ISSN 0280-5316
ISRN LUTFD2/TFRT--5734--SE

Visual Servoing with Time Delay

Rémi Aguesse

Department of Automatic Control
Lund Institute of Technology
December 2004

Department of Automatic Control **Lund Institute of Technology** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* December 2004 *Document Number* ISRNLUTFD2/TFRT--5734--SE
Author(s) Rémi Aguesse	*Supervisor* Rolf Johansson and Tomas Olsson at LTH in Lund Nicolas Andreff from the IFMA in France *Sponsoring organization*

Title and subtitle

Visual Servoing with Time Delay (Robotseende vid tidsfördröjning)

Abstract

The purpose of this thesis was to examine the effects of time delay on a visual servoing system. We analyse the effect of time delays on a linear model of a robotic arm joint, as well as an ABB irb2400 robotic arm with 6 degrees of freedom. This master thesis is consequently divided into two separate parts.

The first concern of the project is to deal with the design of the visual servoing system. In this part we explain how the image processing works and which are the things we have to pay intention. We speak also about the design of the process (controller, dynamic of the robot, etc). This is the preliminaries for experiments but also important to have a good simulation environment, with a model as close as possible of a true robot system.

In the second part we analyse the effects of time delay on the system. We start with the simple model of one joint to have a base to work and after we repeat the analysis using the full model of the robot to see if the behaviour is the same. The real time behaviour is simulated using the TrueTime, which simulates a real time kernel with complex models of the timing of the controller.

All this experiments have been simulated for an easier installation of the experiment protocol (no hardware problems). Another reason is that we had a robot simulator already implemented.

Keywords

Visual servoing, time delay, stability analysis, real time control

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316			*ISBN*
Language English	*Number of pages* 56	*Recipient's notes*	
Security classification			

Contents

Acknowledgments

Being an MSc student in the Blaise Pascal University of science and in third year in my engineer school: L’Institut Français de Mecanique Avancée, France, I was given the chance to carry out my masters thesis in the Department of Automatic Control in the University of Lund, Sweden. I would like to thank Professor Rolf Johansson from the Department of Automatic Control in the University of Lund and Mr Nicolas Andreff from the IFMA for giving me such an opportunity. The experience has exceed my expectations.

Through the course of this project I was involved in a work procedure which had as a purpose the combination of different fields of research, like vision, robotics, and time delay to achieve an ultimate goal, explained in this report. For thus I have principally worked with Tomas Olsson, current PhD students in the University of Lund, at the Department Of Automatic Control. I would like to express to him my sincere gratitude for his help and his co−operation.

I would like to thank Professor Rolf Johansson who has guided me through the whole course of this project and whose ideas made me overcome even the hardest problems I have encountered. I also appreciate the support that Leif Andersson and Dan Heriksson have given me to use programs useful for my work and for that I would like to thank them.

Finally I would like to thank all the other people of the department with who I have passed six very pleasant months whereas the summer weather was bad and learn the rules of Inne Bandy which is an exhausting but interesting sport.

1 Introduction

Visual servoing and time delay in the robotic domain are both problems which have been studied for many years. Indeed, the visual servo in robotics is useful in many applications, such as industrial robotic positioning or distant surgery. And the time delay is one of the biggest problems to solve to have an efficient robot. We will now attempt to give an introduction to both domains: visual servoing on robotic and time delays.

1.1 Robotic and Visual servo

The first use of industrial robots along with computer aided manufacturing systems is dated from 1960's. Since the research area of robotic has progressed so fast, today robots are often used in industry. Indeed the automotive industry has made large investments on this research domain and therefore has played a crucial part in its development.

From then until now developments have made robot capable of making more and more tasks and therefore useful in more application than before. This is a consequence of developments in the area of computing machine which allow robots to increase its performance as well as range of abilities. However, until now industry use robots with restricted sensor feedback, which limit their field of application. Indeed if we visited automotive factories, we should see that for each task a specific robot is used with it own program and set of sensors adapted to this task, which is not easy to change if we want it make another task, because this set of sensors are most of the time internal sensors such as incremental encoders in joints.

Indeed robot control techniques traditionally use world and joint coordinate systems to determine the position of the robot and the desired positions and trajectories. This seems to work satisfactorily for static environments met in industrial applications. So the robot performs tasks in a well known environment.

Problems appear in the control when such robotic manipulators are used in dynamic environment. That is why, the industry begins today to be interested in more adaptable robots and the computer vision is an efficient sensor for such robot. Indeed with visual information we can have a good idea of aspect of robot environment at any moment and so react consequently. That is why a large part of robotic research is devoted to this field of application, would be this only for humanoid robot applications. But is could be also useful in industry to realize the control of robotic arms, tracking of object etc.

In our project we will focus only on the robotic manipulator (robotic arm) and the task of following a line drawn on a board. To do this we will use visual feedback from a vision system consisting of a camera mounted on the end-effector. In such application the control need a feedback to correct the position of the robotic arm dynamically, because a lot of external factor can modify this position or make it obsolete.

So in a visual servo control this feedback is performed by a digital camera, which implies a complex treatment of images to extract the needed information. Indeed if we made a comparison with the human visual system, the camera is only the eye. So we need a “brain” too to interpret the signal send by the camera. This part of visual sevoing is the subject of numerous publications (for instance: [1]) about different process to detect image features: point, line, object, gradient, etc…

Therefore we will speak about the treatment used in our case. Such process being closely linked to the type of information needed: detecting a line is not the same than finding points or other.

1.2 Time Delay

The problem of time delays has always been important in control. The problem of time delay is also closely linked to robotic problem especially when the robot is used in a control loop. Indeed, in such case the control loop need sensors to measure the effect of the control. On the process, time delay appears because of many factors like communication time, processing, etc.

The more a system needs real time process, the more his efficiency is linked to time delay. We distinguish two types of real time control: hard and soft. The soft real time process requires less precision than others. Indeed such process, which are currently the more used in industry are little sensitive to time delay and can work even if the process is not really periodic or the control is based on obsolete information.

On the other hand the hard real time systems are highly sensitive to any problem in the control loop. That is why such systems require a detailed attention about perturbations and especially about time delay which is inevitable. We can control how the system works in presence of a known time delay. This can be useful to determine the behavior of the system.

In this thesis we will see how our system of robotic arm works under the influence of time delay. For more information about control systems with time delay, see [2], [15] and [3].

1.3 Thesis Outline

This master thesis report is organized as follows:

In the next chapter we will introduce pre-requirements in visual servoing to understand the rest of the thesis. We will explain how we design a camera, a visual servo system and we describe the use of the image jacobian.

The third chapter deals with the formulation of the problem and how its proposed solution. We will see all the problems which are related to the subject. Then I will explain how I have designed the simulation model to make the experiments.

In the fourth part we find all which is related to the experiments with one joint. We will see what happens when we have time delays in the system and when we add some noise.

The fifth part is quite similar to the previous as it is the results of experiments with the full model. This is to see if the behavior of the system is the same than with one joint.

In the last part we deal with the results of the thesis and the future work.

2 Background

2.1 Digital Camera

There are two types of digital cameras frequently used in computer vision. These who send intensity images (encoding light intensity) and these sending range images (encoding shape and distance) acquired by sonar or lasers scanners. Both have the same structure, as seen below. They are constituted by optics, which focuses the light on a photosensitive device (a CCD array) and a frame grabber.

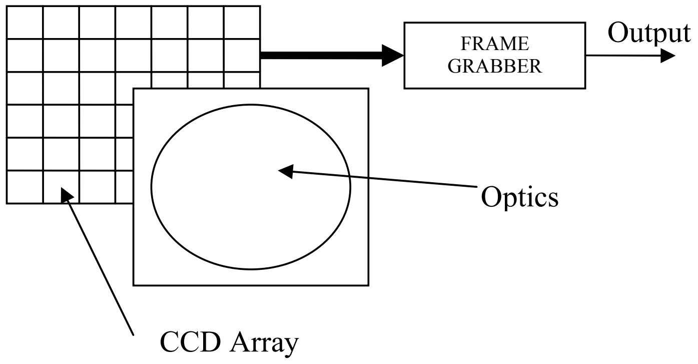


Figure 2.1: Essentials components of a digital image acquisition system

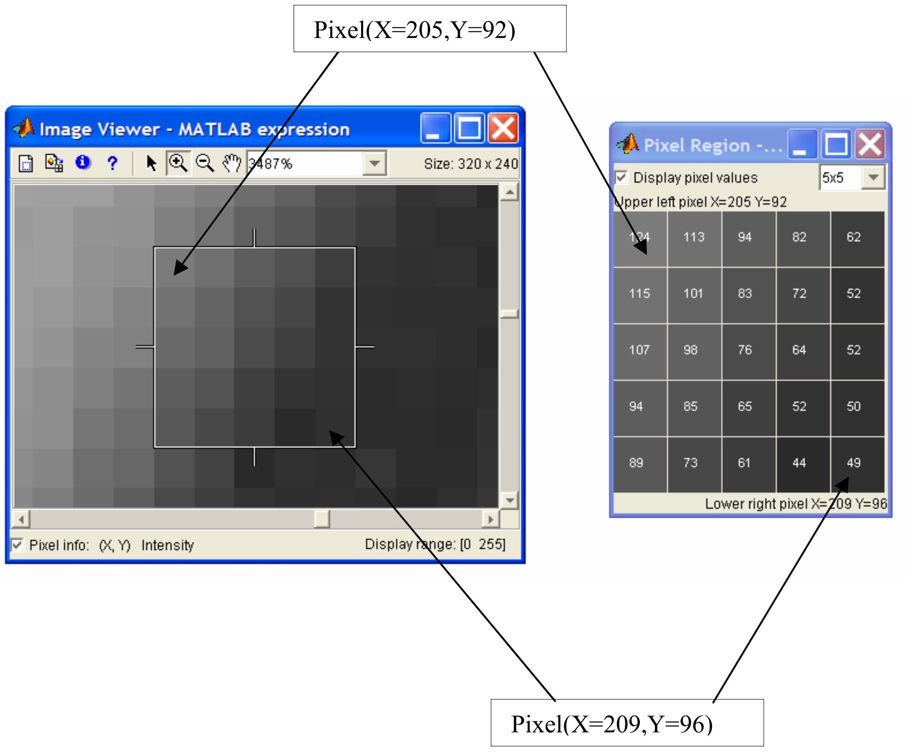
2.1.2 CCD Array

In a CCD (Charged Coupled Device) camera, the physical image plane is a CCD array which is a rectangular grid of photo sensors. . For more information about digital camera, see [1]. Here we used an intensity gray-level camera. That means the sensor is sensitive to light intensity. So each photo sensor stores an energy level corresponding to the amount of light impinging on it. The Output of the CCD array is usually a continuous electric signal, which we can regard as generated by scanning the photo sensors in the CCD array in a given order (line by line) and reading out their voltage.

2.1.3 Frame Grabber

This signal is sent to the frame grabber who digitalizes this signal and translates the amount of energy receive of each photo sensor on an integer in the range [0, 255], typically 0 is black and 255 is white. After this the frame grabber stores the digitalized signal in a memory buffer. The frequency of the camera determines the refresh rate of this buffer.

So digital cameras send matrix of integer corresponding to the scene viewed (figure 2.2 illustrate this). Each entries of this matrix are called pixel (an acronym for picture elements). This matrix is arranged in 2 dimension u and v, where u goes from top to bottom and v goes from link to right. So the pixel (1, 1) is the top link pixel.



Upper left pixel X=205 Y=92

124	113	94	82	62
115	101	83	72	52
107	98	76	64	52
94	85	65	52	50
89	73	61	44	49

Lower right pixel X=209 Y=96

Figure 2.2: Digital Image and corresponding 2D array of numbers

2.2 Camera Model

The aim of such models is to link the position of scene points with that of their corresponding points in images. Indeed a camera transforms a point in 3D space on a point in 2D space by a geometric projection, so we need to know this transformation to work with information sends by camera. Many models exist that are more or less complex and consequently more or less realistic. For more information about others camera models see [4].

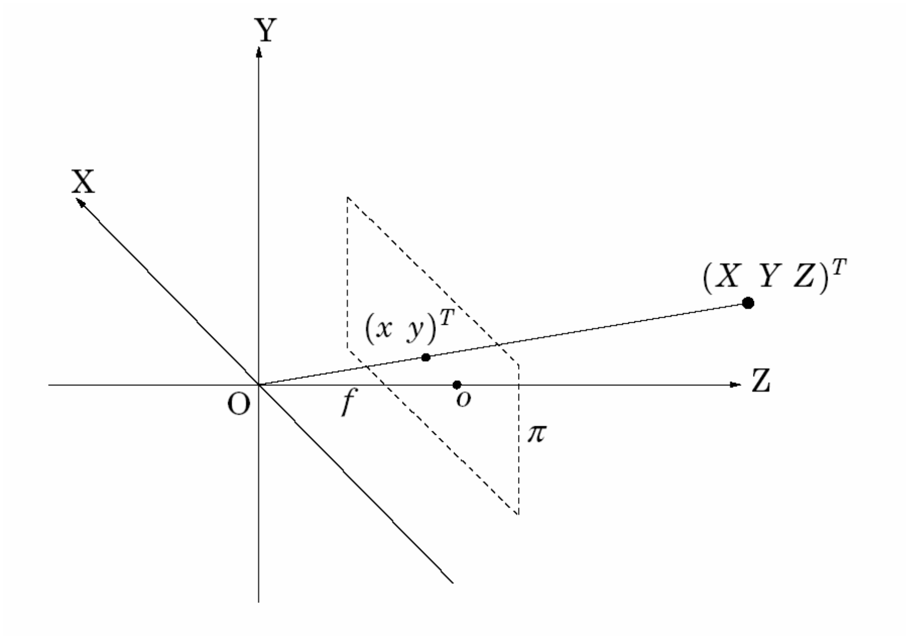


Figure 2.3: The pinhole camera model

In our case we use the perspective or pinhole model, which is the most common geometric model of an intensity camera. The perspective camera model consists of a point O, called the centre of projection, and a plane ∏, the image plane (which is a CCD array in digital camera). The origin of the camera centred coordinate system is in O, see Figure 2.3. The distance between O and ∏ is the focal length f. The line perpendicular to ∏ that goes through O is the optical axis, and the intersection of this line with ∏ is called the principal point o (which is the most often the centre of the CCD device). The projection equations for a point [*X Y Z*] in Cartesian space into coordinate [x y] in the perspective camera are given by:

$$x = f \cdot \frac{X}{Z} \tag{2.1}$$

$$y = f \cdot \frac{Y}{Z} \tag{2.2}$$

This can be written using homogeneous coordinates as:

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad (2.3)$$

Where λ = Z is the depth of the imaged point in the camera.

In This thesis we assume this model of a camera transformation is enough realistic. Indeed in real camera there are some other problems (such as rectangular pixel, non orthogonal array…) that we do not model here and this make the model more complex. So we obtain the normalized perspective projection system below:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} X \\ Y \end{pmatrix} \qquad (2.4)$$

2.3 Image Based Visual Servo

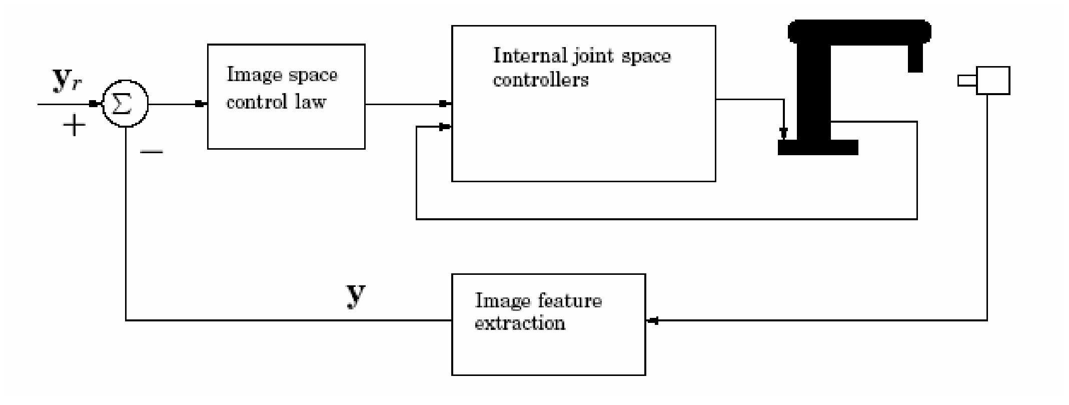


Figure 2.4: Image based visual Servo

2.3.1 The Control Law

In an image based control servo, see fig. 2.4, the control is defined directly in image space quantities. That is why we define the control error as:

$$e = y_r - y \qquad (2.5)$$

Where yr and y are vectors in image space coordinates. y_r is the desired (final) position of the end-effector in the image and y is the measured current position. So a simple control law that would drive the error e to zero is:

$$\dot{y} = K \cdot e \qquad (2.6)$$

In which K is a constant gain. The output $\dot{y}$ of this controller is an image space velocity vector, containing the desired velocity of the end-effector in the image. But the control signal sent to the robot is defined in task space. In this space the velocity screw is define as a 6-vector of translations and angular velocities. Therefore it is necessary to relate differential changes in the image feature parameters to differential changes in the position of the end effector. That is why we introduce the image jacobian J_v.

2.3.2 The Image Jacobian

So let r, which is a vector, represent coordinates of the end-effector in some parameterization of the task space, and $\dot{r}$ represent the corresponding end-effector velocity. Let y represent a vector of image feature parameters and $\dot{y}$ the corresponding vector of images feature parameter rate of change. If the image feature parameters are point coordinates these rates are image plane point velocities. This image jacobian is a linear transformation that maps end-effector velocity to image feature velocity:

$$\dot{y} = J_v(r) \cdot \dot{r} \qquad (2.7)$$

Where J_v is a matrix of partial derivation like below:

$$J_v(r) = \left[\frac{\partial f}{\partial r}\right] = \begin{pmatrix} \frac{\partial f_1(r)}{\partial r_1} & \cdots & \frac{\partial f_1(r)}{\partial r_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_k(r)}{\partial r_1} & \cdots & \frac{\partial f_k(r)}{\partial r_m} \end{pmatrix} \qquad (2.8)$$

The equation (2.7) can be solved using the least square method assuming that $J_v(r)$ is of full rank. This gives the velocity screw $\dot{r}$ that will minimize $\left|J_v(r) - \dot{y}\right|^2$. The velocity screw is usually expressed in the camera coordinate system. In order to generate the correct trajectories for the robot we also need to transform the velocity screws to the robot base coordinate system. Therefore we need the estimations of Cartesian camera robot transformations, obtained from the calibration of the system. This jacobian image is also called interaction matrix.

2.3.3 The image jacobian of point

The most common image jacobian is based on the motion of points in the image. So suppose that this end-effector is moving with angular velocity Ω(t) and translational velocity T as below:

$$\dot{r} = \begin{pmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{2.9}$$

Let p be a point rigidly attached to the end-effector. The velocity of the point p, expressed relative to the camera frame, is given by:

$$\dot{p} = \Omega \cdot p + T \tag{2.10}$$

To simplify notation, let p=[X, Y, Z]^T , so the time derivative of the coordinates of a point on the end-effector, expressed in the camera frame can be written as:

$$\dot{X} = Z\,\omega_y - Y\,\omega_z + T_x \tag{2.11}$$

$$\dot{Y} = X\,\omega_z - Z\,\omega_x + T_y \tag{2.12}$$

$$\dot{Z} = Y\,\omega_x - X\,\omega_z + T_z \tag{2.13}$$

Differentiating the projection equations (2.4) with respect to time and using these expressions we get:

$$\begin{aligned} \dot{u} &= \frac{d}{dt}\left(\frac{X}{Z}\right) = \frac{\dot{X}Z - X\dot{Z}}{Z^2} = \frac{Z\omega_y - Y\omega_Z + T_x}{Z} - \\ &- X\frac{Y\omega_x - X\omega_Y + T_z}{Z^2} = [X = uZ,\ Y = vZ] = \\ &= \omega_y - v\omega_z + T_x/Z - uv\omega_x + u^2\omega_y - uT_z/Z \end{aligned} \tag{2.14}$$

$$\begin{aligned}\dot{v} &= \frac{d}{dt}\left(\frac{Y}{Z}\right) = \frac{\dot{Y}Z - Y\dot{Z}}{Z^2} = \frac{X\omega_z - Z\omega_X + T_y}{Z} - \\ &- Y\frac{Y\omega_x - X\omega_Y + T_z}{Z^2} = [X = uZ,\ Y = vZ] = \\ &= u\omega_z - \omega_x + T_y/Z - v^2\omega_x + uv\omega_y - vT_z/Z\end{aligned} \tag{2.15}$$

So we can write this system on matrix form:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} 1/Z & 0 & -u/Z & -uv & 1+u^2 & -v \\ 0 & 1/Z & -v/Z & -1-v^2 & uv & u \end{pmatrix} \begin{pmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} : \tag{2.16}$$

So the first matrix is the jacobian image of a point in image frame. If the features are a group of point or others object, we can stack each jacobian to build the jacobian of all features.

3 Method

The main goal of this part is to determine how we will proceed to experiment on the subject of our thesis. Therefore we will see first the problem formulation, to see what subject implicate. Just after, we will speak about materials and tools used to make experiments. Finally I will see you how I have proceed to prepare my experiments.

3.1 Problem Formulation

3.1.1 Time Delay

Real-time control systems are inevitably affected from the time delays occurred. Therefore when we take into account time delays, we can represent real-time control systems by a closed loop system like the one shown in the figure 3.1 below, according to this publication [2] about real time control with delays.

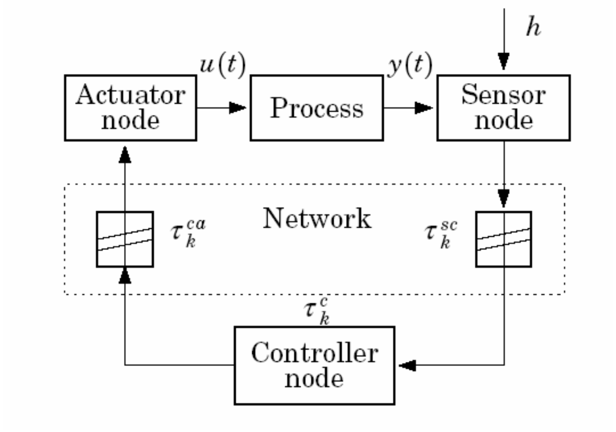


Figure 3.1: Time delays in system

With such representation we can easily see that the different time delays occurred are the following:

-The communication delay between the sensor and the controller

-The computational delay within the controller

-The communication delay between the controller and the actuator

In a visual servoing system where the sensor is a digital camera, we can easily identify the first of these delays between sensor and controller as the time it takes for a captured image from camera to become available for processing. Whereas the computational delay is the consequence of time needed to perform the algorithm of image processing. The last time delay can be identified as the time taken by the computer to send the controller output to the robot.

So the control delay of the system, the time from when a measurement signal (image here) is sampled to when it is used in the actuator (the robot), is equal to the sum of these three delays. The problem is this delay is not constant; he is varying in a random fashion. Indeed the communication time depend on the quantity of data to send and on the type of network used too. Typically Ethernet is non deterministic, so we can not be sure of the moment when the data is usable. And even on a clock synchronized network, the length of sent data modify the delay.

Thus we will make experience to see the influence of time delay on the control. Therefore we will see the link between the time delays and the robustness of our system. We should experiment several representation of delay.

3.1.2 Visual servo

A simple representation of a visual servo control loop is this shown by the following figure (fig. 3.2). This is a closed control loop, where the sensor is a digital camera.

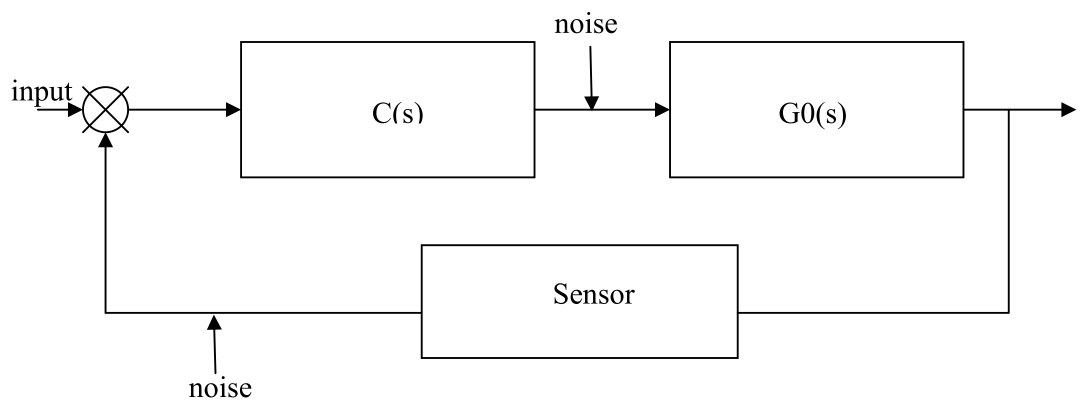


Figure 3.2: visual controller structure

In this transfer loop, the bloc G0 is the transfer function of the process. In other words this is an equation which represents the behaviour of the robot when this one is excited by a command. The bloc C is the transfer function of the controller. But if we want the simulation more realistic we can modify this closed loop to add noise in the data send by the sensor or the controller.

Indeed in real system there are always some noises that disturb the signals. These noises are the consequence of many factors like:

-error in calibration of the camera

-precision of the camera

-leak of precision in the image processing

-friction in the joint of the robot

So in our experiment we have to represent these noises, if we want the simulation is the more close to the real system. Thus we will start our experiments by the setup of one joint control loop. After these first experiments we will make simulation with a full model of the robot and we will see if the results are these expected.

3.1.3 Image processing

As we have seen above a digital camera need a treatment of the image to extract information from the data. These data are only value computed in a matrix. So we need to interpret the arrangement of these data in the matrix. Thus we can know where are the seeking features (points, line, etc…) in the image.

Therefore we need to make an algorithm using this value to determine if there are features in image. This algorithm is depending on which feature we used and what is the task made by the robot. So we need to create a new algorithm to each application of a robot.

The problem of such algorithm is they spend time. Consequently the global time delay is depending of this algorithm. That is why I have paid attention particularly to the velocity of this algorithm. The algorithm has to be enough fast to allow the system works correctly.

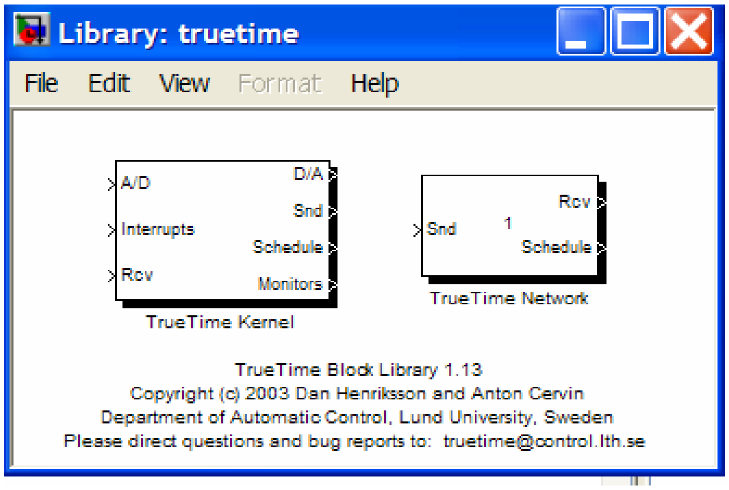
3.2 Tools

3.2.1 Matlab/Simulink

To design and make simulation of the real system I have used as main tool The Mathworks Solution called Matlab/Simulink. This program allow to program functions with a really simple interface and intuitive programming language. With Simulink we have the possibility to create many kind of simulation system and run them. Indeed this module of Matlab proposes a graphical environment in which we build the system with block sorted by their properties.

3.2.2 Truetime

Truetime is a Matlab/Simulink-based toolbox which facilitates simulation of the time delays and network characteristics. The last version of this toolbox has been developed in the Lund Department of Automatic Control by Dan Henriksson and Anton Cervin in 2003, on a base of an early version developed in 1998. For a detailed description of how to use the simulator, see [5]. This simulator is constituted by a library of two simulink block as we can see on the figure 3.4.



TrueTime Block Library 1.13
Copyright (c) 2003 Dan Henriksson and Anton Cervin
Department of Automatic Control, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se

Figure 3.3: The Truetime Block Library (The display and Monitor outputs display the allocation of common resources (CPU, monitors, network) during the simulation.)

This both blocks allow building easily simulink models containing time delays. Indeed the first, the TrueTime kernel block executes user define tasks and interrupt handlers representing: I/O task, control algorithm and network interface. This block takes into account user defined execution time of task and transmission time of messages. Furthermore TrueTime allows simulation of context switching and task synchronization using events. Thus it allows both periodic and event started task.

All the inputs and outputs are assumed to be discrete time signals except the signals connected to the A/ D converters which are the data used by the user-defined task. The display and Monitor outputs display the allocation of common resources (CPU, monitors, and network) during the simulation.

In fact the kernel block simulates a computer with a real time kernel, A/D and D/A converters, a network interface and external interrupt channels. So this kernel maintains the characteristics who define a real-time system such as interrupt handlers, monitors, timers and events so called jobs. For definitions of functions used in this kernel see the user manual [6].

The second block allows simulating a local area network with all his characteristics. Thus Six models of network are currently supported: CSMA/CD

(Ethernet), CSMA/AMP (CAN), Token Bus, FDMA, TDMA, and Switched Ethernet. In this simulation block, the propagation delay is ignored, since we are in a local area network.

This block is event-driven and executes when messages send by the Kernel block enter or leave the network. When a node tries to transmit a message, a triggering signal is sent to the network block. When the simulated transmission of the message is finished, the network block sends a new triggering signal on the output channel. These messages contain data transmitted by the network between each node.

3.2.3 Robot

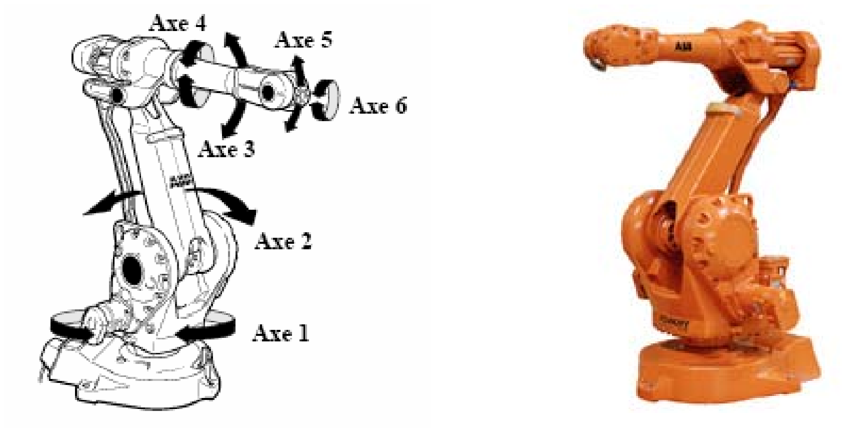


Figure 3.4: Robot Irb 2400 with 6 degree of freedom

In Lund department of Automatic Control there are two robots. The one used for my experiments is the robot Irb 2400 which is an industrial robotic arm of ABB. This robot is coupled with a new grey level camera with high refresh frame rate (100 Hz). This is the digital camera A 602f of Basler. The Camera is connected on a computer by a Firewire link (IEEE 1394 2000) which is a high performance serial bus. The picture format is 320 x 240 pixels. We assume the camera is hold by the end-effector of the robot. For further information see the reference [7] and [8].

3.2.4 Winrobot

Winrobot is a graphic simulator of robot and his environment. We can see an example of environment with the figure 3.5. This program has been developed here in the Lund Department of Automatic Control by Tomas Olsson. With such program the simulation of my system has been greatly simplified. Indeed this is a program write in C++ which can be used under Matlab environment thanks to the functions wrote in both syntaxes. So these functions allow all we need for a robotic simulation:

- Definition of robot

- Definition of object

- Moving the robot (in his joints frame)
- Placement of object and robot in the space
- Definition and positioning of camera
- Attaching camera to the robot or object

This program is based on OpenGL library which allows creating a 3D environment and creating object on it from files. You can See below the environment of the simulator.

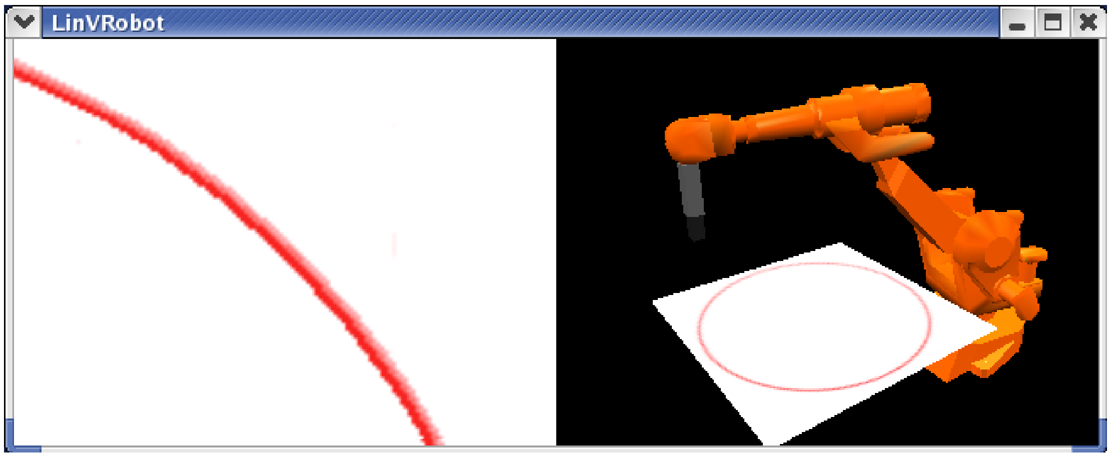


Figure 3.5: Winrobot environment example with two cameras. The first, on the left is attached to the end effector. The second is fixed in the space to see the environment.

3.3 Model and Design of the robot

Once the tools chosen I have created my simulation system. This has taken time because of my inexperience in matter of image processing and time delay. I had to read several publications to understand what do these both field of study mean. Thereafter I could start the implementation of my system, which I will explain in this chapter.

3.3.1 Image processing

As I said before the image processing is an algorithm which depends closely on the feature searched in the image and the task we want the robot is able to do. For recall the task defined in the problem formulation is the following: the robot has to follow a line drawn on a board. Here when we said robot, we want the end effector follows the line as accurately as possible.

First of all we have to know what the nature of the sought feature is and what his characteristic in an image are. So to follow a line we have to know where the robot is in comparison with the target line. It is the same problem to find the line position in comparison with the end effector. Therefore, I have decided to find the closest point of the line to the center of the camera, which is assumed as the reference of the end effector.

This problem which is really simple for a human being is much harder for a computer. Indeed for a computer each pixel of an image are the same (only their values are different) he can not associates a pixel to a feature working without his neighborhood. Thus when a tried my first algorithm, it was a disaster because I had not taken into account this fact. I had assume a pixel as member of the line only by his value and defining two threshold limit to avoid the pixels who did not have the same "color" than the line.

$$A(x,y) = \begin{cases} 1 & \text{if } 25 < \text{Pixvalue}(x,y) < 120 \\ 0 & \text{else} \end{cases} \quad (3.1)$$

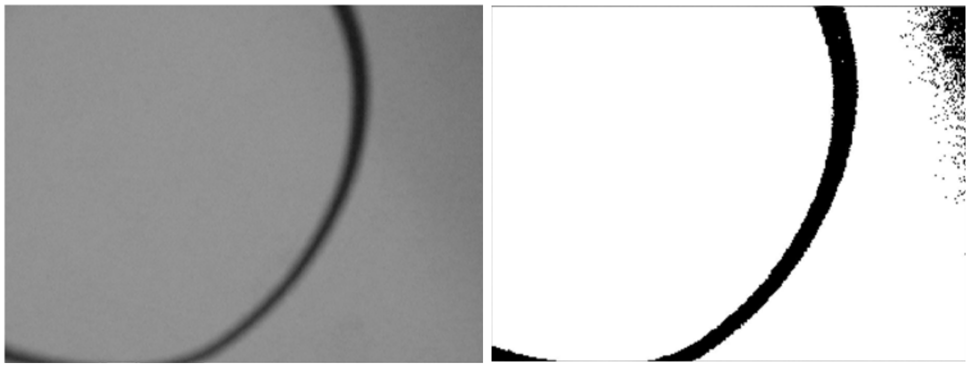


Figure 3.6: First result of simple algorithm (we can view see noise on the right of the image)

Where A(x,y) is the function which determines if the pixel (x,y) belongs to the line. It depends of the gray level B of the pixel. The thresholds are relatively extended because of the non uniformity of the line color. The result was the following, see figure (3.6) we can see that some noisy pixel are considered as line member which is catastrophic for the efficiency of the algorithm.

In order to eliminate these “noisy” pixels I have had resorts to a calculation with the neighborhood matrix of each pixel see below. This matrix was a 5x5 array of pixel and I checked if each pixel of the matrix was seen as line member. But this was not enough, because big region of noisy pixel stayed assumed as line member as we can in the figure 3.7.

$$A(x,y) = \begin{cases} = 1 \quad if \quad 25 < Pixvalue(x+i, y+j) < 120 \quad \forall i,j \in [-2;2] \times [-2;2] \\ = 0 \quad otherwise \end{cases} \tag{3.2}$$

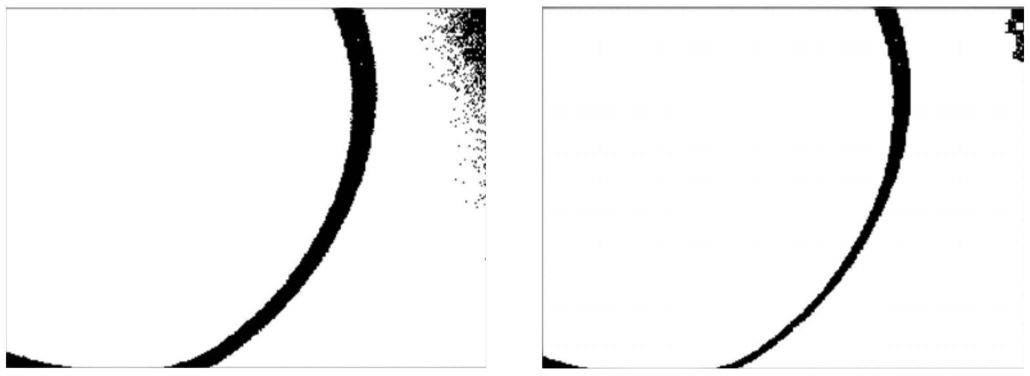


Figure 3.7 result of advanced algorithm

And the other problem was: when a black element (pixel value =0) near a white (pixel value =255) there is always a gradient, many pixels have an intermediary value between 0 and 255. So when there is a border between black and white elements in the image, there were many pixels of this border assumed as line member.

At this time I have observed a property of my line: The gradient of grey level in the orthogonal direction of the line was really small (due to the drawing of the line made with a large red felt). Thus the pixel value rises up slow from background to the center of line, instead of noisy region where the value rises up really fast.

That is why I have change the algorithm as below, see equation (3.3). Thus I have had pretty good results without implementing really complex algorithm, which would slow down the speed of computing. To see the results go to the figure 3.8.

$$A(x,y) = \left\{ \begin{array}{c} \begin{array}{l} = 1 \\ \end{array} \begin{array}{c} if \quad 25 < Pixvalue(x,y) < 120 \\ and \frac{7}{8} \cdot Pv(x,y) < Pv(x+i, y+j) < \frac{8}{7} \cdot Pv(x,y) \quad \forall i,j \in [-2;2] \times [-2;2] \end{array} \\ = 0 \quad otherwise \end{array} \right\} \quad (3.3)$$

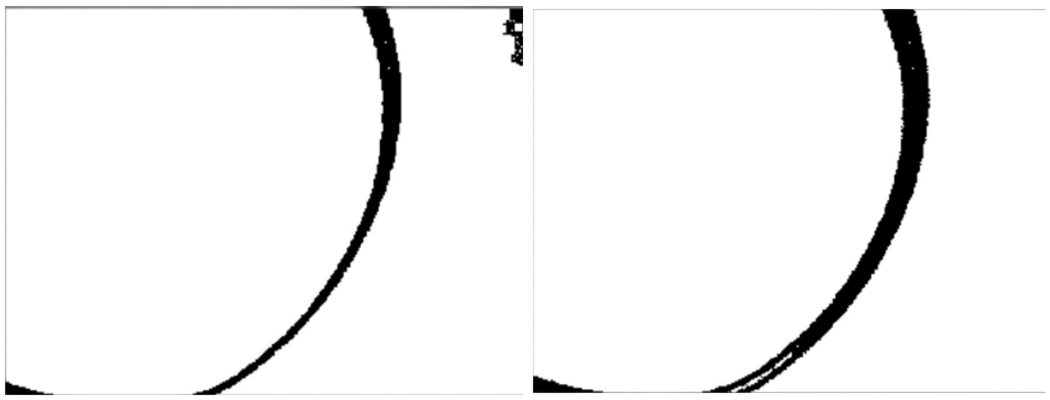

Previous algorithm Final algorithm

Figure 3.8 : result of final algorithm

Having the point members of the line, it was easy to obtain the closest point from center of image, by searching the one with the smallest norm. With this point known we can now determine the direction to follow. With this intention we have to calculate the tangent of the line in the found point. But the line not having any physical reality in the image, this is only a list of point that was not so easy.

So I have used the mean line square method (see reference [9]) to obtain the orientation of the direction vector. Thus I create the list of coordinate point which are in the neighborhood (a 25x25 frame) of the point found earlier and which are member of the line. In this neighborhood we can assume the line as straight and we apply the mean square method to the lists of abscises and ordinates.

To avoid the singularity when the line is horizontal I see the value of the old orientation to see if the slope is included in the interval [-1,1] or not. In the first case I invert the coordinates x and y (I search for the least squares line with the following equation: x=a.y+b), then I invert the value of the slope obtained to have the good value for the tangent ; in the second case I use the usual method of least squares line with normal Cartesian equation. Finally I have to calculate the cotangent of the slope value to obtain the value of the orientation. With the orientation we have two directions so by making a comparison with the older direction we can determine which one is the good (this is the one which form the smallest angle with the older direction).

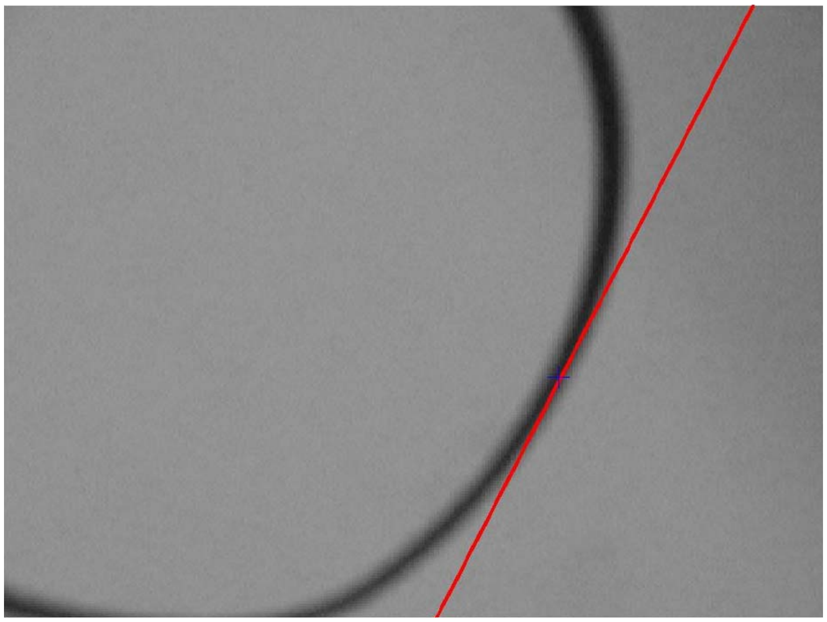


Figure 3.9: result of image processing

Here we can see the orientation calculated by the algorithm (red line) is pretty good according to the image and the closest point too.

The last problem was the number of data to scan. Indeed in an image of 240x320 pixels we have to make the treatment for 76800 data which is too much large to have an efficient algorithm. Therefore I have reduced the field of study of the image to a small frame around the point found in the precedent application of the algorithm. The time cost reduction is huge (with a frame of 25x25 pixels the number is not any more but of 625 data, the cost time is so divided by 120!).

This algorithm has been made with numerous functions to be understanding, but unfortunately it was too slow (velocity of 10 Hz) in comparison to the frame rate of the camera to be implemented in a real time control loop. So I have to make only one big function less readable but impressively faster (500 Hz in Matlab, which is not compiled program, so not as fast as C++ programs). You can read the entire code in the appendix of the image processing function.

3.3.2 Robot Configuration for experiments

The application of visual servoing in this thesis to study the time delay is the following of a line drawn on a board. Therefore the configuration of the robot and his environment is like on the figure 3.10.

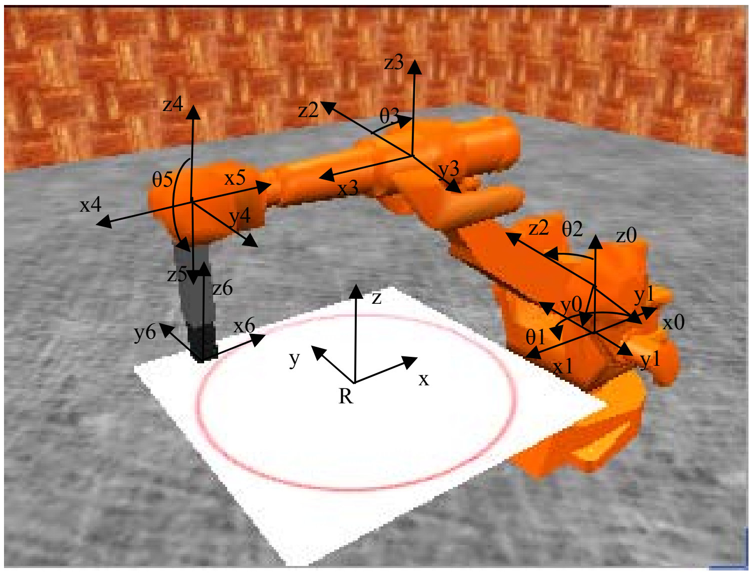


Figure 3.10: Experiments environment configuration

We can see the board is placed horizontally, and centered on the origin of the world space R {X, Y, Z}, the basement of the robot being moved along X with the coordinates system R0 {x0, y0, z0}. Then we place coordinates systems in each joints like shown in the figure 3.9. With such parameters for the robot we can obtain the transformation matrix between the coordinates in end-effector base and the coordinates in the joint base

But our robotic task is really constrained. So with our experiment we can add few hypotheses that will simplify the future calculations and will explain how this task works. The hypotheses are the following:

- The end-effector move in a horizontal plan
- The camera is oriented vertically
- The camera is rotated to have always the same orientation as the world space
- Consequently the end effector is only in translation (no rotation)

These hypotheses can be translated into equations about angle and velocity in joints. Indeed this implicate that joint 4 is locked, joints 1 and 6 are linked as joints 3 and 5. The obtained simplifying equations are the following:

$$\dot{\theta}_4(x,y) = \theta_4(x,y) = 0 \tag{3.5}$$

$$\theta_6(x,y) = 180 - \theta_1(x,y) \quad \text{and} \quad \dot{\theta}_6(x,y) = -\dot{\theta}_1(x,y) \tag{3.6}$$

$$\theta_5(x,y) = 90 - \theta_3(x,y) \quad \text{and} \quad \dot{\theta}_5(x,y) = -\dot{\theta}_3(x,y) \tag{3.7}$$

So we have easily the position relation of the end effector as below (3.8), taking into account this equations and the transformations matrix. The calibration of the robot was not a goal of the theses, thus I take the lengths send by the documentation of the robot.

$$\left\{\begin{matrix} x = 1 + \cos(\theta_1).(l_1 + l_2\sin(\theta_2) + l_3\sin(\theta_3) + l_4\cos(\theta_3)) \\ y = \sin(\theta_1).(l_1 + l_2\sin(\theta_2) + l_3\sin(\theta_3) + l_4\cos(\theta_3)) \\ z = l_2\cos(\theta_2) + l_3\cos(\theta_3) - l_4\sin(\theta_4) \end{matrix}\right\} \tag{3.8}$$

We can now calculate the inverse cinematic model of our robot. This model is the relation between the velocities of the end-effector in the world coordinates system and the velocities of the joints. This model is like below:

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{pmatrix} \frac{\partial(x)}{\partial\theta_1} & \cdots & \frac{\partial(x)}{\partial\theta_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial(z)}{\partial\theta_1} & \cdots & \frac{\partial(z)}{\partial\theta_6} \end{pmatrix} . \begin{bmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_6 \end{bmatrix} = J_v(\theta). \begin{bmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_6 \end{bmatrix} \tag{3.9}$$

Actually the velocity of the end effector, according to preceding hypotheses is a two components vector (Tx and Ty) and is linked with only three joint velocity (the three others being null or compositions of the others see equations (3.5),(3.6) and (3.7)). So the cinematic model is the jacobian matrix 2x3 you can see in the equation (3.10):

$$J_v(\theta) = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \tag{3.10}$$

with

$$a = -\sin(\theta_1).(l_1 + l_2\sin(\theta_2) + l_3\sin(\theta_3) + l_4\cos(\theta_3))$$
$$b = l_2\cos(\theta_1)\cos(\theta_2)$$
$$c = \cos(\theta_1)(l_3\cos(\theta_3) - l_4\sin(\theta_3))$$
$$d = \cos(\theta_1).(l_1 + l_2\sin(\theta_2) + l_3\sin(\theta_3) + l_4\cos(\theta_3))$$
$$e = l_2\sin(\theta_1)\cos(\theta_2)$$
$$f = \sin(\theta_1)(l_3\cos(\theta_3) - l_4\sin(\theta_3))$$

So to obtain the joint velocity in function of end effector velocity we have to use the pseudo inverse of the jacobian matrix calculate just above (see below). So we have the value for the first three joints and the last three are calculated with the simplification equations (3.5), (3.6) and (3.7).

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = J_v{}^+(\theta).\begin{bmatrix} T_x \\ T_y \end{bmatrix} \tag{3.11}$$

3.3.3 Controller

The controller is the bloc in the simulation model who sends the velocities in joints to the robot. To compute this it uses the features sends by image processing to obtain the error which is in our case the vector between the closest point of the line and the center of the camera. To correct this error we use a simple proportional control law. Indeed As we can see in the following paragraph we assume the joints as uncoupled. That means if a joint moves; it does not change the behavior of over joints. And in such case it was shown that this law is good enough. But to follow the line we have to add to this correction a velocity vector. The direction (θorient) of this velocity vector is send by the orientation calculated in the image processing see chapter 3.3.1 for further explanations. All of this means that the control law is the following:

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = J_v{}^+(\theta)\begin{bmatrix} Tx \\ Ty \end{bmatrix} + T_f \cdot \begin{bmatrix} \cos(\theta_{orient}) \\ \sin(\theta_{orient}) \end{bmatrix} = J_v{}^+(\theta)\cdot Kp \cdot L_T{}^+.(s - s_0) + T_f \cdot \begin{bmatrix} \cos(\theta_{orient}) \\ \sin(\theta_{orient}) \end{bmatrix} \tag{3.12}$$

In the equation (3.12) we can see the matrix $L_T{}^+$, this is the inverse of the jacobian matrix which allow the transformation from the velocities of the end effector to the velocities in the camera space of the point found with the image processing. This jacobian is a simpler version than one seen in the background. Indeed our robot moves in an horizontal plan and only in translation. So the matrix is reduce to a 2x2 matrix (this only the first two columns of the full jacobian matrix :

$$L_T = \begin{bmatrix} \frac{\lambda}{z} & 0 \\ 0 & \frac{\lambda}{z} \end{bmatrix} \tag{3.13}$$

3.3.4 Robot dynamic

The study of the dynamic of a robot is not the goal of this thesis. So I have used the dynamic model, which has been already determined for the robot Irb 2400. This dynamic is a linear model determined with the ARMAX method. So the dynamic is the following equation (3.14) in discrete space:

$$G_0(z) = \frac{B(z)}{A(z)} \tag{3.14}$$

With:

A(z)= z^3 - 0.3722 z^2 + 0.1181 z – 0.3462

B(z)= 0.01622 z^2 + 0.007641 z + 0.3771

Discretization period: 0.004 sec.

This dynamic is applicable to a joint of the robot. We assume that the dynamic is the same for each joints; this implicates each joint is not linked with the others. According to experiments on the robot we can assume than the error created by the difference between the ARMAX model and the real non linear model is small in comparison with the errors in signals; therefore we do not take into account this error for our results. For explanations about ARMAX method sees the following references: [10]

4 Preliminary Experiments

This part is dedicated to the study of only one joint. For this study we will start to setup the model of one joint. After doing this thing, we will see the influence of the time delay on the control. Finally we will add to the control loop some noise on the data to simulate the more realistically the comportment of the joint.

4.1 Experimental Setup

Since we have already defined the design of a control loop system, we will now setup the corrector to have a good response time. This allows making further experiments. In this part we will assume that the good condition of experiment is a system that response time to have 5% of the final signal is about 0.5s. So we simply see the response of the system to a step input and we choose the corrector consequently. As seen before the corrector is a simple proportional gain. This is the only one value to setup. You can see below the output of the system with few value of corrector.

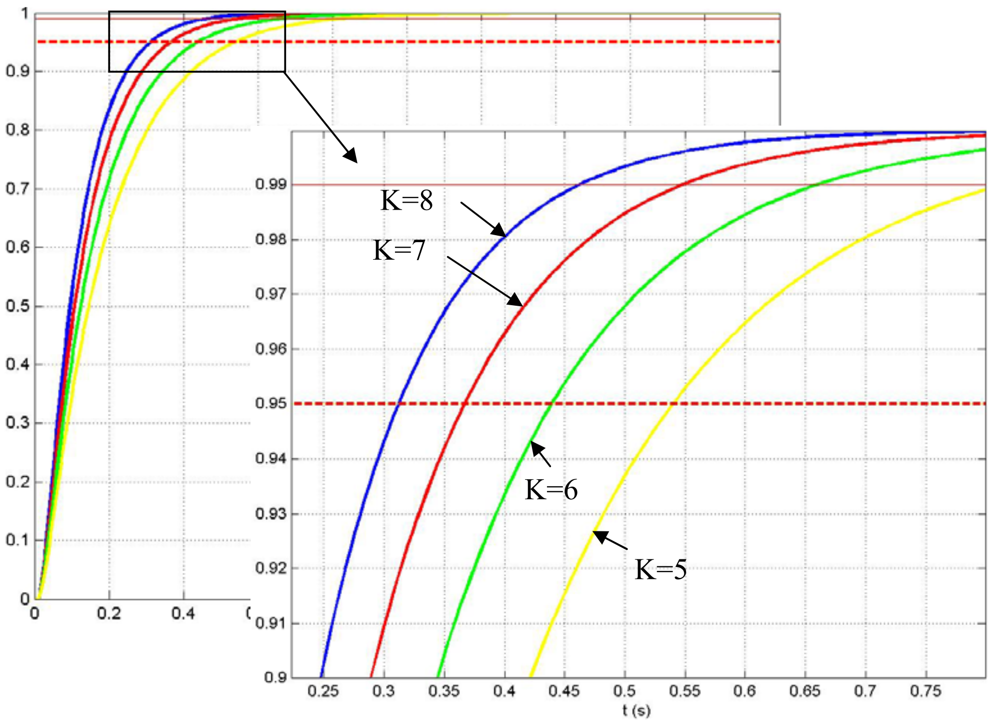


Figure 4.1: output of the system with few corrector for a step input

So according to the figure 4.1 we have chosen to setup the corrector to six, in fact it is the smallest integer which take the response time of the system smaller than half a second. By choosing a bigger value the system should be faster to reach the final value, but the system should be more sensitive to the noise as we will see later. With a smaller value the system should not be enough dynamic to allow the control of the trajectory of a robotic arm, the response should be to much slow.

4.2 Results with Time Delay

Now we will add to the system delays. Indeed as we have seen before the real systems have always time delays and this have impacts on the response of the system to the input. So we will start to add a constant delay (the simplest we can design) and finally we will see the effect of a distributed time delay (with a Poisson law).

4.2.1 Constant delay (transport)

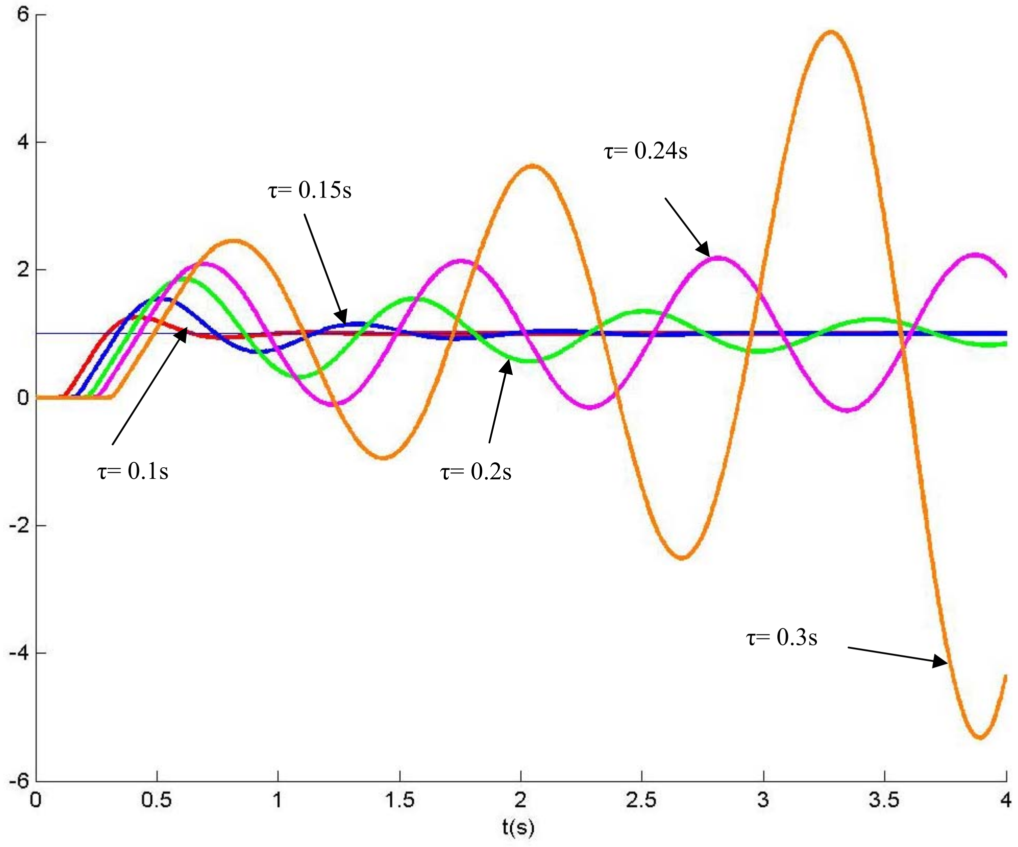


Figure 4.2: response to a unitary step with few transport time delays

As we can see in the figure above, first of all the delay introduce an overflow of the output signal. We can also see that the introduction of the smallest delay (0.1 s) does not change really the dynamic of the system. Indeed the response time of the system to reach the ninety five percent of the signal is quite similar to the signal without delay (about 0.1s more). In the other hand the bigger the delay, the slower the system is: with a delay of 0.15 s the response time is approximately 2 s.

The figure 4.2 shows also that the system with a delay of 0.25s oscillates with a period of 1s. That means the signal has reached the limit of the stability. Indeed with a bigger delay (0.3s) the system becomes unstable. So with a smaller delay the system is stable by definition, but if the delay to close to this limit of stability, the system is not satisfying because he oscillate too much to be assumed as a step. This problem occurred when we had a dynamic input as sinus or a pulse signal. Indeed if the system does not stabilize as fast as the frequency of the signal, the output has no more common point with output as we can see below. This depends on the nature of input too: for instance a pulse is more sensible than a sinus to these delays.

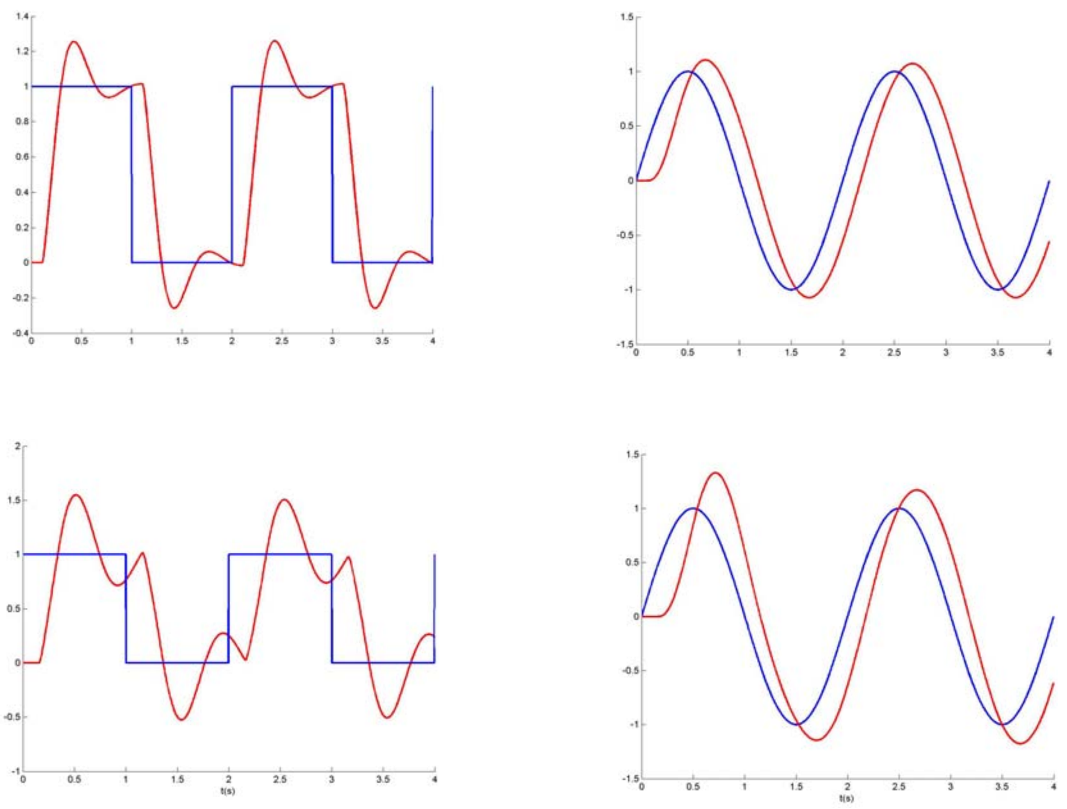


Figure 4.3: outputs (red) of the system for inputs (blue). The upper figures are the results with a delay of 0.1 s and the ones just above are the results with a 0.15s delay

As we can see with the figures 4.2 with a delay of 0.1s the results are quite similar to the inputs (in the pulse case we assume easily it is a response to a pulse). In the other hand with a bigger delay the pulse response has no more common point with the input whereas in the sinus case the output stays quite similar to the input. This shows as said before the input type is important in the instability problem. Indeed with a sinus the signals move softly; consequently the error to correct too. So the delay which introduces error is less important than in the case of a pulse that is a non continuous signal what implies abrupt changes in the value of the error to correct.

4.2.2 Constant Delay (computation)

The problem with of the first study above is that the delay work as a simple transport of the signal, without loss of information. In fact all the information send by the corrector to the process (our joint) are received by the process, but with a delay. It seams that information are old but after a delay of initialization the process receive the control as a continuous signal. This is good to simulate the latency in the system as the transport delay in the network. But it does not work in the case of a computation time of corrector. Indeed in such cases the computer can work only on one value, so if information are send to the computer whereas this one has not finished to treat old information, the new one are lost.

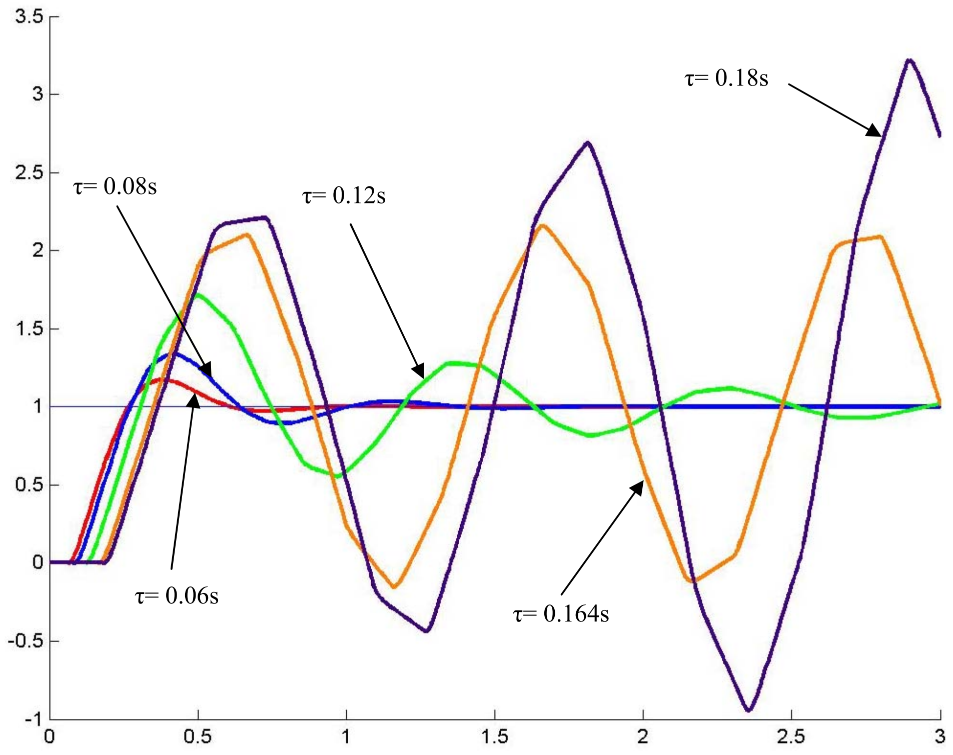


Figure 4.4: response to a unitary step with few computation time delays

To simulate such case we have to use Truetime, because Matlab can not treat this easily, it allows only the transport delay and no the computation time. Except this difference we have used the same protocol of experiments. So we can see with the figure 4.4 that the behaviour of the system is quite similar to the transport delay case with some difference meadows. Indeed first of all the system is less stable, as we can see the values of delay are smaller than in the first case. For instance the limit of stability is reached for a delay of 0.164s versus 0.24s for the transport delay case. Secondly we can see the bigger the delays are the more numerous discontinuities in the derivative function are. This is explained by the loss of information: with a huge delay none the less the information are old but many of them are missing too so the behaviour of the system is less smooth.

For the case of dynamic input in the system, the results are quite similar to thus with transport delay so we do not need to remake this experiment. With pulse signal the system stays more sensible to delay than in the sinus case (the systems stays to work good for bigger delays than with pulse input for the same frequency).

4.2.3 Distributed delay

Now we have seen the effect of a constant delay on the system, we will see if the results are the same with a random delay. To do this we will design the delay has a variable time distributed according to a Poisson law. Indeed in many works about time delay in system using network (see [2], [11] for further information) the delay is designed like this, because it is the more realistic model which is simple to implement. The distribution function is like below (equation 4.1) and we can see also repartition distribution for few main values with the figure 4.5.

$$\lambda = \text{Mean value}$$

$$p(x=k) = e^{-\lambda}.\frac{\lambda^k}{k!}, k \in \{0,1,..,N\} \qquad (4.1)$$

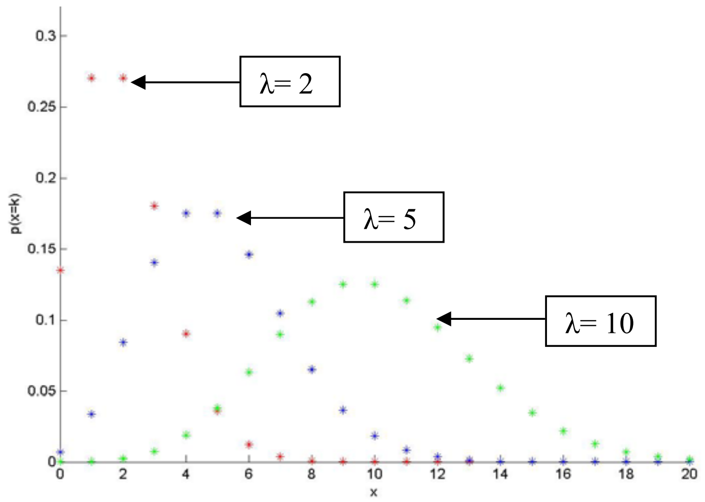


Figure 4.5: distribution repartition according to a Poisson law with three mean values

As we can see in this equation this distribution law is discrete with unitary gaps. So in this case the value of the delay x can only be an integer. So if we want a gap between two consecutive value of x smaller than 1, we have to customize the final distributed value of x and the distribution function consequently. So for instance to have a mean value equal to 0.2s (which is a little bit more than the delays obtain in the preceding experiments) and a gap equal to 0.1s, we have to set the mean value of the distribution to 2 and divide the value of x by 10. In the same way to have the same mean value and discrete rate of 1/100s, we setup the mean value to 20 and divide x by 100.

As we can see below in the figure 4.6, for the same mean value the smaller the discrete rate is the less the value of delay moves away from the average value. So with a discrete rate of 0.1s the delay has a big probability to be equal to 0 contrary to the case with a discrete rate of 1/100s or 1ms with which the probability to have a delay smaller than .15s is almost null (with a mean value equal to 0.2s). So we will see the influence of these differences in our system.

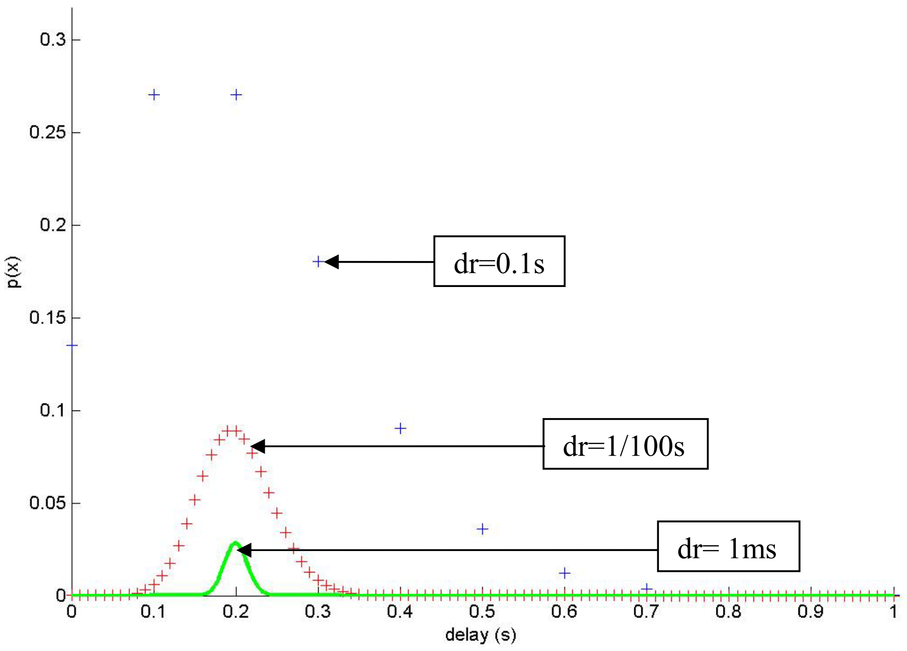


Figure 4.6: delay distribution for few discrete rates

So we will made an experiment for a mean value of 0.12s (which is delay how does not create instability in the case of a constant delay, see first experiments). We will do this for three discrete rate values (0.1s, 1/100s, 1ms). The first figure (4.7) shows the output of the system for a unitary step input. The second (4.8) deals with the delays during three experiments.

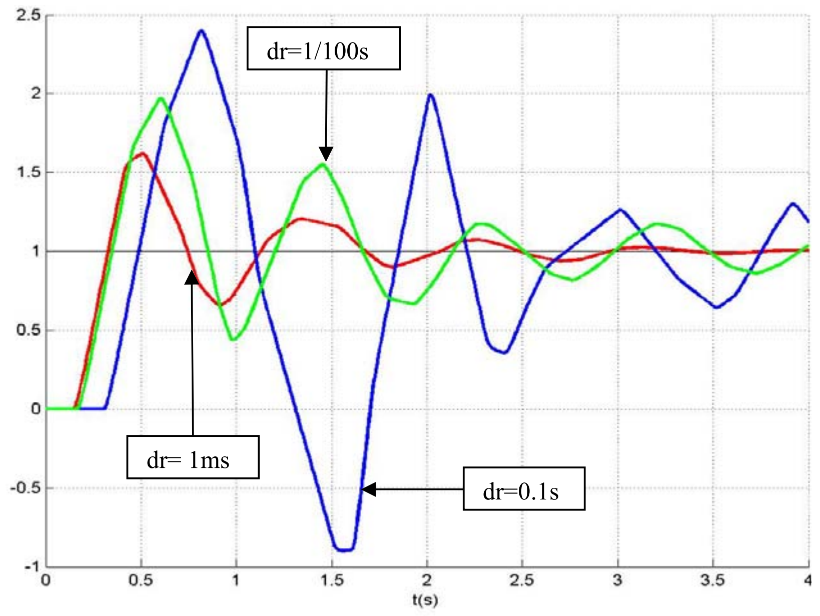


Figure 4.7: response of the system for three distribution of delay (large, normal, and narrow)

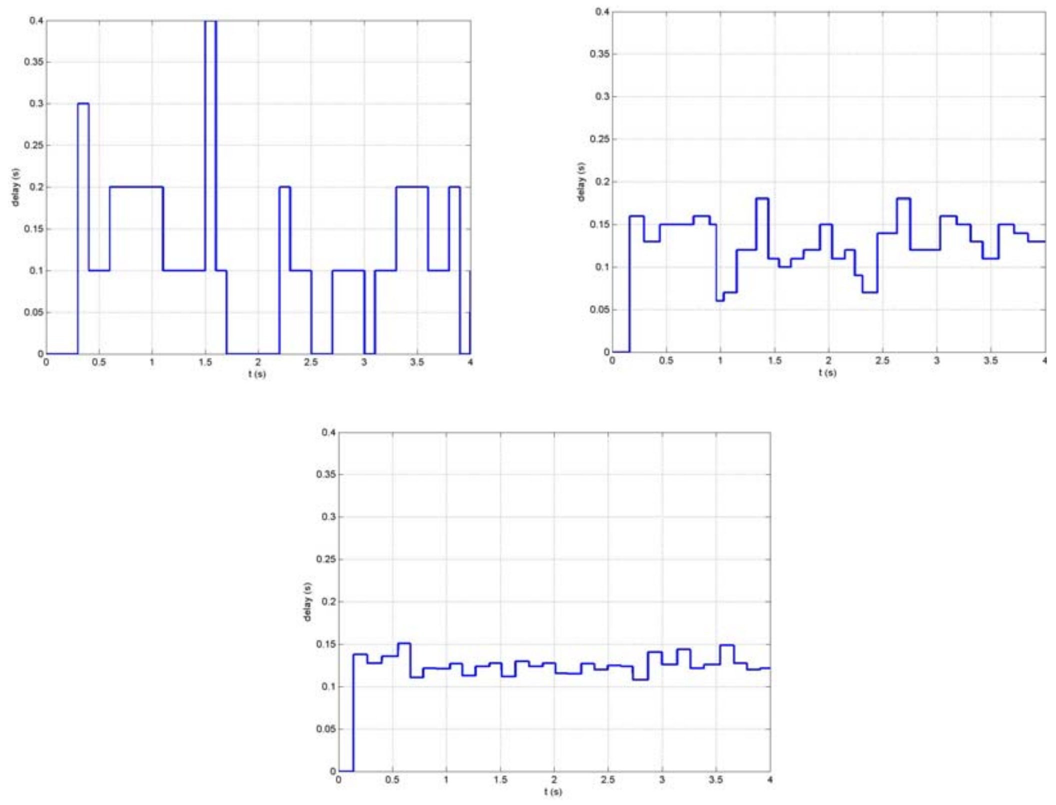


Figure 4.8 : Delays variation during the simulations for the three discrete rate.

With these experiments we can concluded that the interval of variation for the delay has an effect on the stabillity. Indeed with the same mean value for the interval of variation the simulation with the narrowest interval is more stable than the others. So the result of the first experiments (with the larger interval from 0 to 0.4s) is a bad response to the step even if sometime the delay is null. In the other hand the last experiment with all the delays close to the mean value is more fast and the response is almost the same than in the case of a constant delay equal to the mean value of the distribution as we can see below. With a constant delay the system is a little bit faster but this is conform to the previous results.

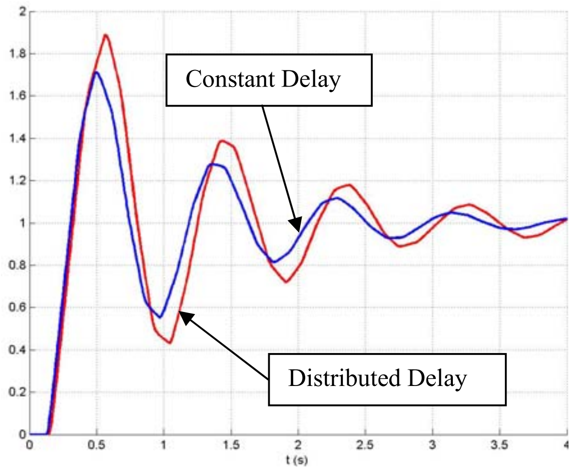


Figure 4.9: comparison beetwen constant delay (0.12s) and narrow repartition (0.1s-0.15s)

4.2.4 Simple Stability Criterion

In the reference [11] or [14] about Time Varying Delay Systems A criterion of stability is developed allows to know if a system with a delay is stable or not by knowing the maximum delay. So for a plant P(s) and a controller C(s) with a varying delay Δ, the theorem is the following: the system is stable for any time-varying delays defined by:

$$\Delta(v) = v(t-\delta(t)), \quad 0 \le \delta(t) \le \delta\max \tag{4.2}$$

if

$$\left|\frac{P(j\omega).C(j\omega)}{1+P(j\omega).C(j\omega)}\right| < \frac{1}{\delta\max.\omega}, \quad \forall\omega\in[0,\infty[. \tag{4.3}$$

So it is the same to say that magnitude of the transfer function of the closed loop system has to be below the criteria line. We will apply this theorem to our system for a constant delay of 0.12s and 0.18s (we have seen before that in the first case the system is stable contrary to the second case). In these cases the system is the following:

$$C(s) = \frac{6}{S} \tag{4.4}$$

$$P(s) = \frac{133.3501\,(s^2 - 407.7s + 1.063 \cdot 10^5)}{(s + 60.72)(s^2 + 204.5\text{s} + 2.328 \cdot 10^5)} \tag{4.5}$$

So we obtain the following magnitude diagrams:

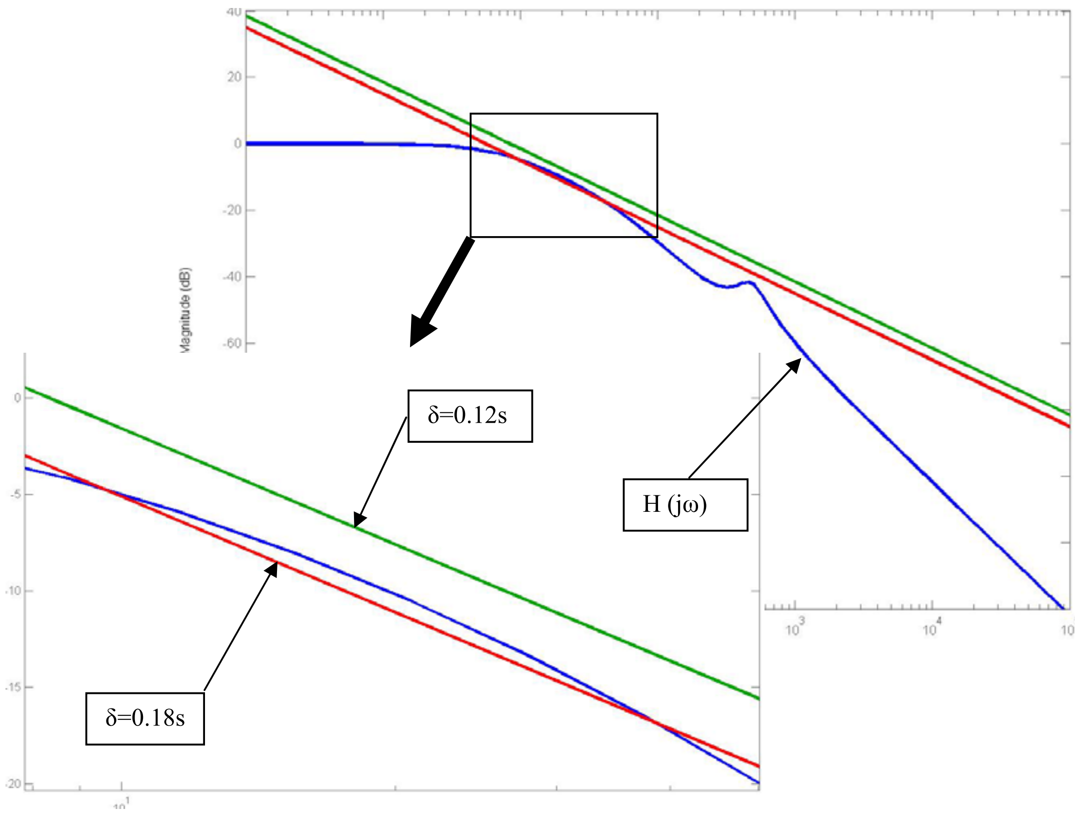
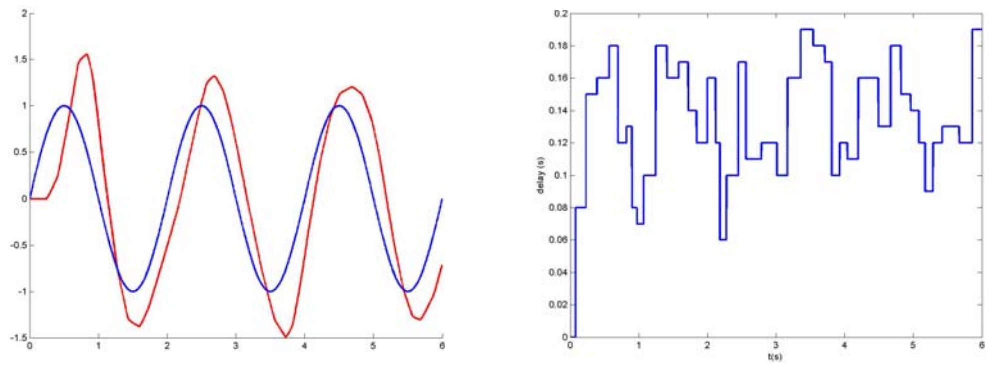


Figure 4.10 : magnitude of bode diagram for system with constant delay

As we can see the results are conform to the conclusion obtained in previous experiments. Indeed with the criterion of stability we have found that with a delay of 0.12s the system is stable but not with a bigger delay equal to 0.18s : the red line corresponding to the delay of 0.18s is below the system transfer function for frequencies ranging between 10 and 100 Hz approximately which is the area of frequencies where our system work (indeed to fresh rate of the digital camera is about 100 Hz). On the other hand, the green line corresponding to a 0.12s delay is always above the Bode diagam gain of the system for any frequencies.

But if we use a system with varying time delay, where the delay is 0.14s for instance and can have a maximum upper than 0.18s this criteria will not be check whereas the system is stable has shown before. Indeed as we can see below with an average delay equal to 0.14s and a maximum delay of 0.19s the system remains stable whereas according to the criterion the system has to be unstable. So this criterion is good to be sure that the system will be stable but in few case we can have a system stable whereas the criteria said no (but in this case the system is close to the limit and so the response can be all the same bad with a pulse input for instance).



System response

Simulation delays

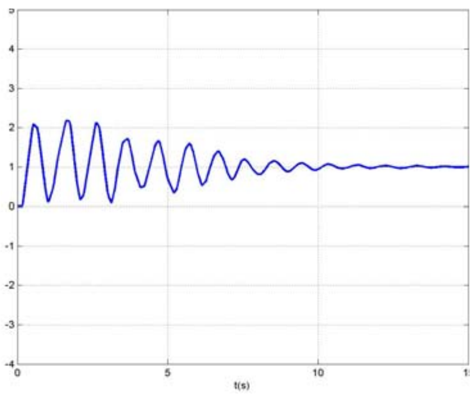
Figure 4.11: response of the system and delay during the simulation

So with such criteria we can approximates the limit of stability for a system. Indeed we can inverse the problem of the criterion. In fact by viewing the bode magnitude diagram of the transfer function we can find the equation of the line with a slope of – 20 dB per decade which is tangent (and always upper) to the system bode magnitude. This can be interesting to setup easily a system and be sure it will be stable.

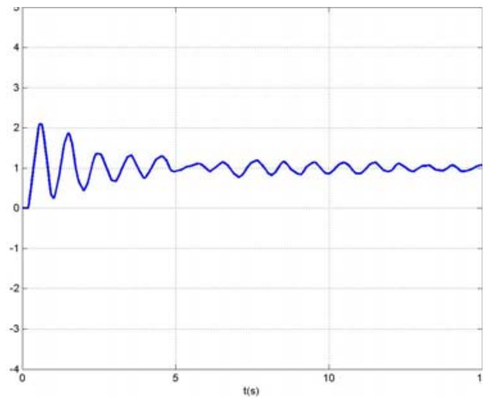
4.3 Noise Addition

As we have said in the formulation of problems, in real system we can not cancel all the noise created by errors in measurement or in the processing. In fact the digital camera used to compute the control is never perfect and is by definition a dicrete device (the smallest precision we can have is the tall of a pixel). Further more errors can also be introduced by the control process.

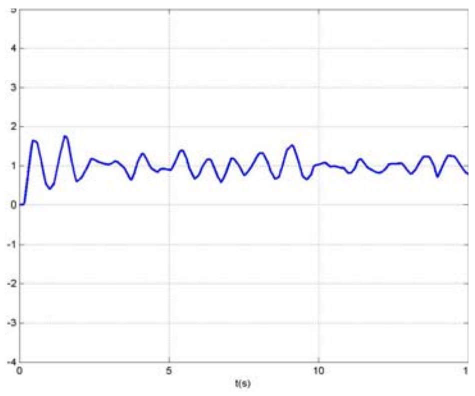
Consequently to have a better model of the real process we will add now noise in the control loop (in the return branch).The most often we assume the error created by this problem can be design as a white noise.We will see now the effect of this white noise on the system. For that we simulate three noise with different variance value (typically 0.001, 0.01 and 0.1), with a unitary step input.



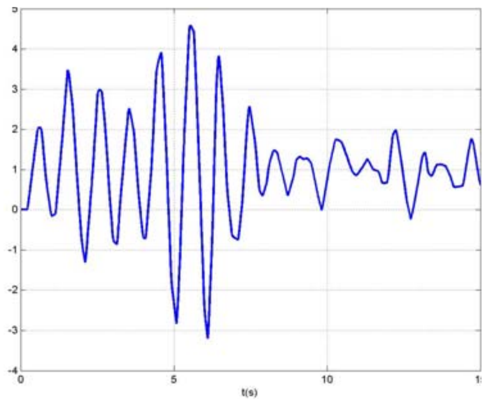
No noise



Var=0.001



Var=0.01



Var=0.1

Figure 4.12: step responses for few white noise and an average distributed delay equal to 0.14s

As we can see bigger the variance of the noise is, more instable becomes the system. Indeed without noise the system reachs the value of the step in 10 s aproximately. But with a smal variance (0.1% of final value) the system oscilate closely to the final value. For bigger variance the response has no more common point with the input and so the system is really bad.

Consequently we have to pay attention to the noise when we work close to the limit of stability of the system. If we work with a big delay we have to be sure that the system stay stable even with noise, so an accurate sensor and computation which limit the noise can be necessary. In fact with a small delay the problem of noise is less important, as we can see below (figure 4.11). Without time delay the white noise (even with an high variance) has a very small influence on the system: the output stay close to one. Bigger the delay is more the noise can be a problem for stability.

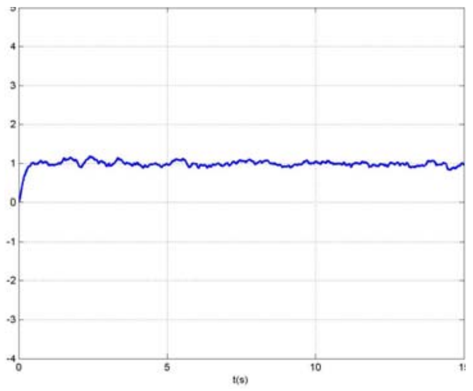


Figure 4.13: step response without time delay and with white noise (var=0.1)

5 Experiments with full system

Now we have study the behaviour of a system with one joints we will see if the results obtained in this fourth part are applicable to the case of a system with more joints as a robotics arm. We will start by watching the behaviour of the system without time delay to have a base to compare. The second part deals with the problem of time delay with a complex visual servoing system.

5.1 Experiments without delay

In the goal to have a base of comparisson, we will see the behavior of the system for three kind of curve : a circle, a square and a unspecified curve. The data we have to study are the following : the velocities of the end effector, the norm of the error (in pixel) between the centre of the camera and the closest point found by the image processing and finally the trajectory of the end effector to see if this follow as good as possible the curve.

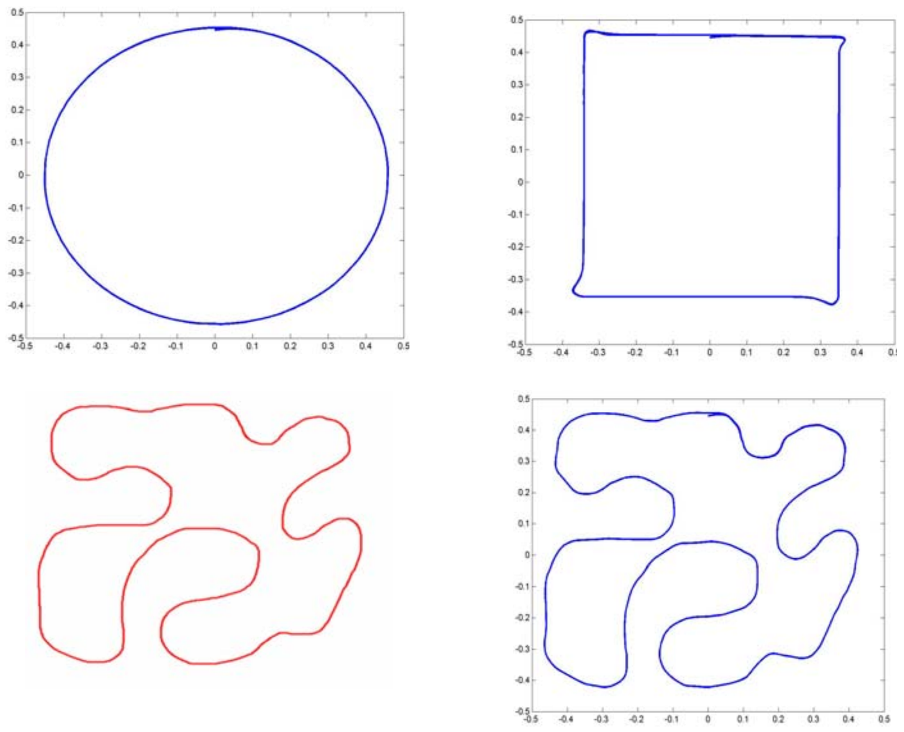
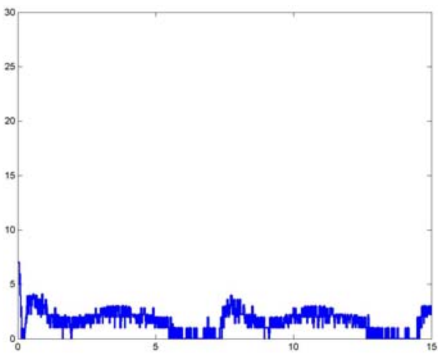
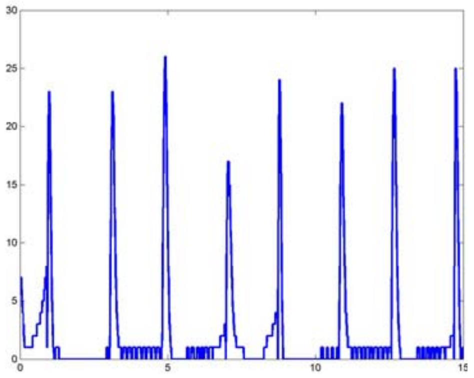


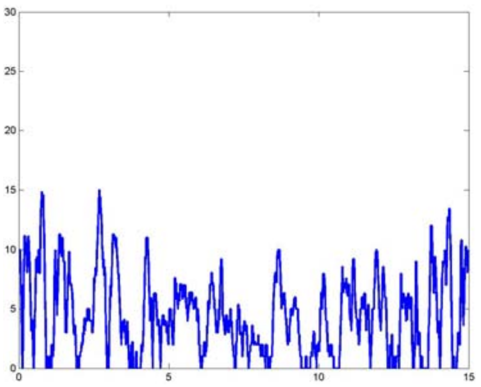
Figure 5.1: trajectories of the end effector without delay for a cercle, squarre and unspecified curve defined on the bottom left



Circle

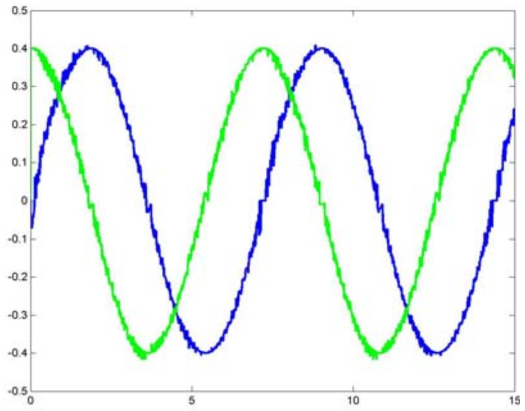


Square



Random Curve

Figure 5.2 : error of position from the center of camera to the curve



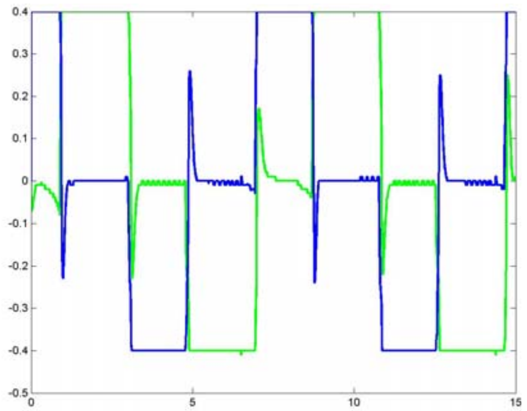
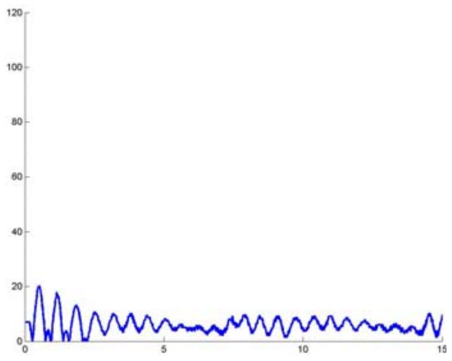


Figure 5.3: control value for the circle and square cases.

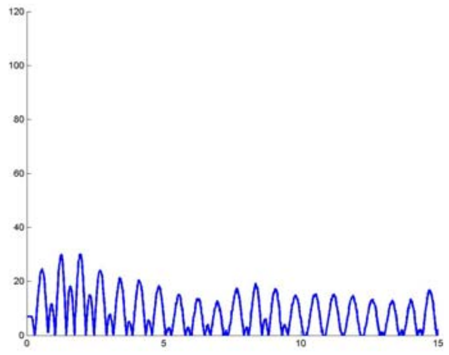
As we can see with this first three figures (5.1, 5.2, 5.3) the results of the control are really good and close to the line to follow. Indeed the trajectories of end effector for circle and unspecified curves look like the original curves. In fact the error (measured as the diference between the line and the center of the camera) is never upper than 5 pixels in the circle case and 15 in the other. And the control (in x and y) for the circle seems to have no error only a really small noise.

In the square case the response stays good too, but in angle the response is not the same than the drawn curve. The error is however small (never bigger than 15 pixels). And the most part of time the error is smaller than the other curves (about 1 pixel). The system has the same behavior than in the experiments with only one joints. The big error when the end effector changes its moving direction is the consequence of angle, so the transition is not soft, and this creates a perturbation to correct.

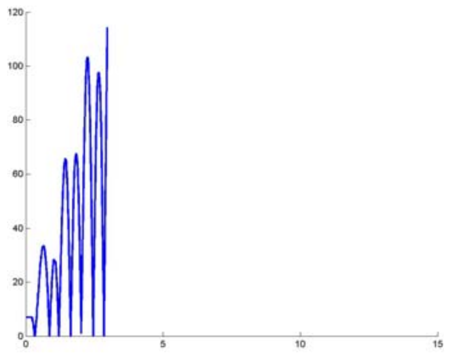
5.2 Experiments with delay for circle curve



Delay=0.12s



Delay=0.14s

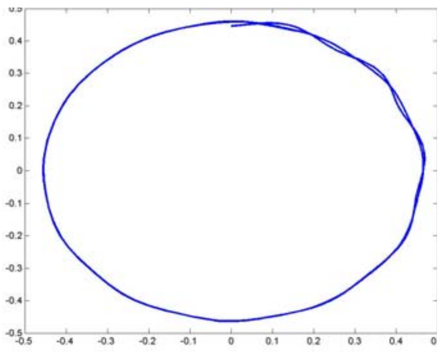


Delay=0.18s

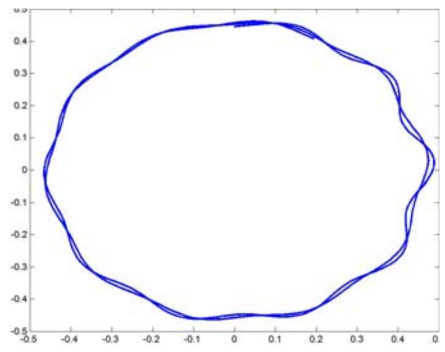
Figure 5.4: Error for the circle line with three delays.

In this experiments we have drawn a circle and the end effector follows this circle. The velocity to follow the line is fixed to 0.4 s. We have setup the regulator as in the case of one joint. So we will see if the delay have the same effect in this case than with one joints and a step as input. In the previous experiments we have found the limit of stability is equal to 0.165s. Consequently we have made two experiments with a bigger and a smaller delay to see what happens. This results are in the figure 5.4. we can see the system stay stabe for a delay equal to 0.12s and is unstable with a delay of 0.18s. Is seems the system have the same behaviour than in one joint case.

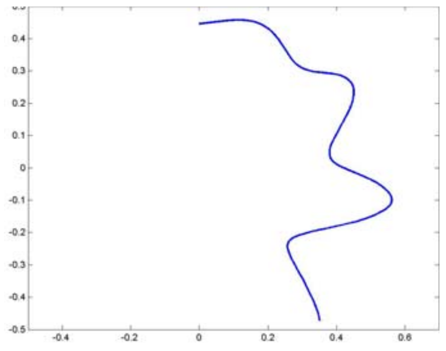
With one more experiment to reach the limit of stability for the full system, we have obtain that the maximum delay is equal to 0.14s. This results is really smaller than the previous (about 0.16s). This can be explained by the dynamic input : the controller correct the error between the line and the end effector, but the control has also another part to follow the curve, so this part create more error and consequently reduces the maximum delay.



Delay=0.12s



Delay=0.14s



Delay=0.18s

Figure 5.5 : trajectories of the end effector following a circle with delays

The trajectories show in the first case the beginning oscillation are reduce with the time contrarely to the case with 0.14s delay: the end effector oscillate all the time around the curve. In the case with 0.18s delay the curve is lost after 3s of simulation because of oscillations which grow up.

5.3 Experiments with delay for square curve

Now we do the same as before but for the square case. In the previous experiments without time delays we have found the system was less stable in this case than with a circle trajectory. So the limit has to be smaller than before and as we can see below (Figure 5.6) this is true because the delays used are equal to 0.08 and 0.12 s respectively. And we can see with a 0.12s delay the system is unstable, when the end effector has to change the direction to move, the system oscillate too much and the image processing loses the direction to follow. Indeed the end effector goes back after the second direction change.

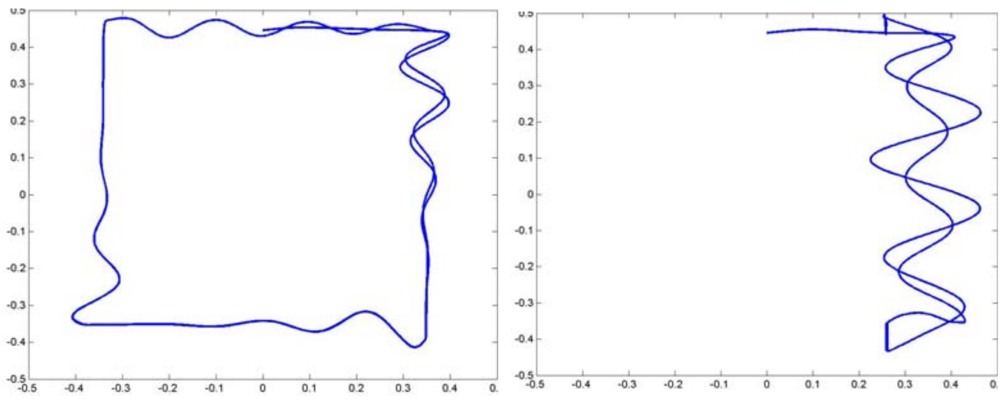

Delay=0.08s Delay=0.12s

Figure 5.6: trajectories of the end effector following a square with delays

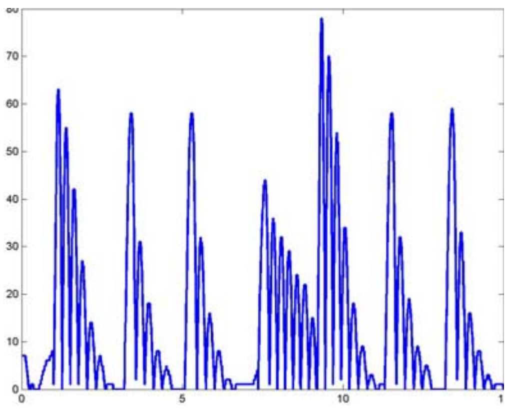
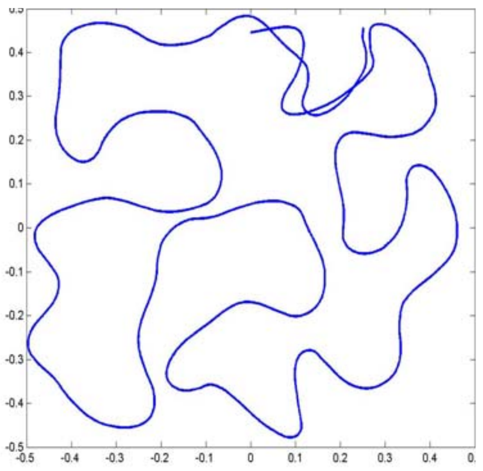


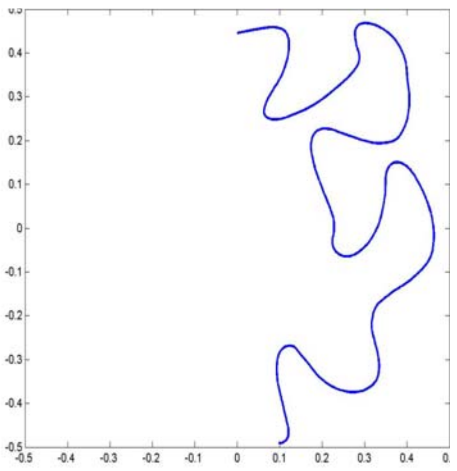
Figure 5.7: : Error for the square line with 0.08s delay.

The figure 5.7 shows for the 0.08s delay the maximum error is bigger than in the case of the circle but this error is reduced to zero after a transition phase whereas in the circle case the error oscillate all the time. So in the case of a square line the system become unstable if the system has not enough time to reach the curve before a change of the direction to follow. This is that happens with a 0.12s delay: the system continues to oscillate when the direction changes.

5.4 Experiments with delay for unspecified curve

Now we use a unspecified curve to see a more general case. Indeed the trajectory does not to have to be necessary a square or a circle. This curve is the one I have use to create the image processing.





Delay=0.12s

Delay=0.13s

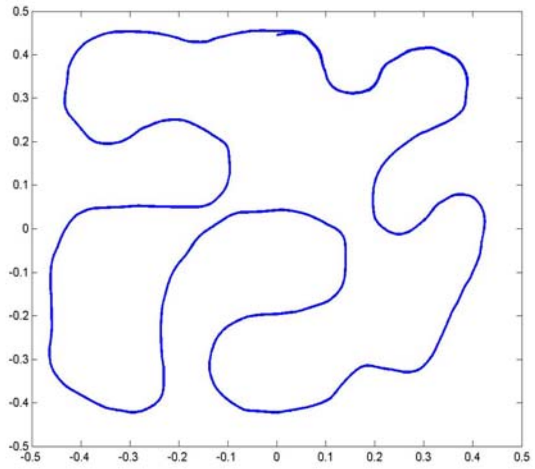


Figure 5.6: trajectories of the end effector following a unspecified curve with delays and the drawn line to compare

The results match we have seen before. Indeed the curve has no angle so it is normal that the maximum time delay is bigger than with the square case (this angle creates errors which are bad for stability). But this curve is less regular than a circle so it seems normal the maximum time delay is smaller. We can see even if the system has no problem to follow the line with a delay, the response is bad (the response has no more common points with the curve to follow, because the system oscilllate too much.The limit of stability is close to 0.13s. Indeed with such delay the end effector can not follow the full curve. After few seconds the system lose the line because of oscillations.

5.5 Experiment without velocity

In this experiment we setup the delay to 0.17s which a little bit bigger than the limit of stability in the case of one joint and we see which are the results if we do not try to follow a line (we setup the controller without the part of the controller to follow the line) and have a smal error in the beginning of the simulation . As we can see the system is unstable (this is not the case for smaller delay). This results are similar to the case with one joints (limit delay of stability equal to 0.165s).

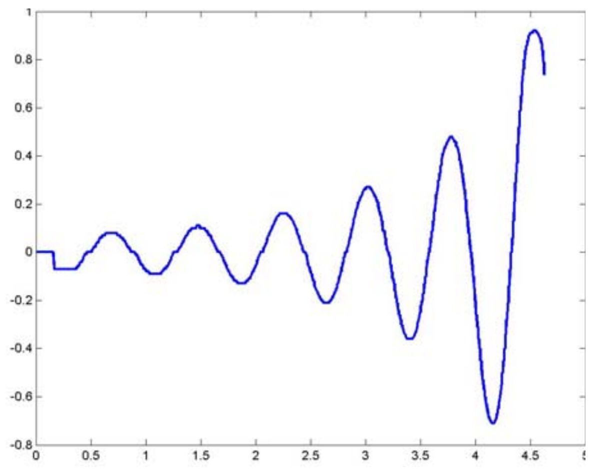


Figure 5.7: control value for the static case.

6 Conclusion and Discution

6.1 Discution

The first part of my work was the modelisation of the experiments environment. For this I have used an existing robot simulator to avoid the problem we can encounter with real systems such as mechanical or electronical problems, numerous noises ,etc. Indeed with a simulation we can choose our model, so we can simulate the noise that interest us time delay in my case and not the others. We can see clearly the effect of the simulated noise (easier than with a true system).

First of all I have paid attention to the image processing to be sure, it will fast enough for the system with a frequency of 100Hz for the camera. But after the first designing of this process the frequency of the image treatment was 10Hz because of the amount of data to treat (240x320 bytes for the full image) and the use of matlab (which is not an optimizated language). So after a study of the slowing part the code, I modificated them and reduced the part of image used in the process to obtain finally a processing time equal to 2ms.

After that I have made the full model of the experiments and started to set the corrector of the command to have a good enough response time of the system, then I observed the behaviour of the system with a delay; and I have seen that the limit of stability of the system in response to a step was reached for a delay of 0.25s. But with dynamic input the delay's limit is reduce to 0.1s (beyond this limit the output has no more common point with input with square signal for instance).

In the one hand when we lose information (occurred when the delay is bigger than the processing time) the delay's limit is reduced (0.16s comparred to 0.25s). In the other in the case of time variing delay the behaviour of the system is approximately the same as with a constant delay, if the distribution is not too large : for instance 0.1s for a 0.15s delay. So a moderate variation of the delay has no real influence on the behaviour of the system.

Indeed with the stabilty criterion from the reference [14] I have obtained the same result because of a large distribution the line corresponding to the maximal delay cut the magnitude bode diagram of the system which is a reason of non stability according to this criterion. But this criterion is not enough accurate because there was some case stable opposing to the result of the criterion. With this criterion we can be sure that a system is stable but not it is unstable.

Finally after this first result I have used the complete system with my image processing and the robot simulator. And Firstly I have improved my algorithm with the simulator and many trajectories without delay, to be sure it was good enough to have no influence on the stability of the systems and fortunatly that was the case (see fig. 5.1) according to the value of the error (about 5 pixels to the circle and 15 pixels maximum for any curve without angle .

Then with delays I have obtained the same results than in the first experiment but the limit was smaller than further. And the system is more stable with a circle trajectory than with any other. This because the system is less excited in this case. In fact the robot end effector oscillate around the line to follow. The bigger the delay, the larger are the oscillations so with a big delay the between two image the robot has so moved that the line is no more in the part of the image which processed, consequently the robot lose the trajectory.

So now it will be good to verify this result with a real system, because we can never be sure than simulation is close enough to the reality to have the same behavior as a real system. And I think also that a more professional solution to the problem of image processing will reduce the effect of delay by an active research of the line in the image and better algorithms. But the mean problem is I have no time to study the effect of predictor on the stability, and see how corrects a system with delay.

6.2 Conclusion

The subject of this thesis has been visual servoing with time delays, and first part of my works was devoted to the image processing. This subject was new for me because all my knowledges about this was theoretical. So first of all I had to read about this subject and proceed by experiments. This was enriching. Indeed I learned how the robotics vision works, with problems of object detection, sampling time, etc.

Indeed I have payed attention to the execution time of the process because the sampling rate of the camera. Indeed We use a camera with a refreshing rate of 100 Hz, to be efficient the process has to be faster than the sampling of the camera. Otherwise we lose images and consequently information about the system state. But it can be hard to have an efficient process with a good execution time : we have to reach the good mix.

This first part done I have worked on the system designing. This part was easier because I have already made project on servoing system. But some problems remained like the design of the delay. Indeed Simulink allow only the simulation of transport delay. Consequently it was hard to simulate computation time delay (to see what happens if we lose information when the delay is bigger than the sampling). This is particularly true when we use time varying delays, the signal is send in desorder. Consequently the results was unusable.

Then I have learned about time delay. Indeed I had never worked on time delay previously. That was interesting to read the works about this subject. After I have made experiments with the models I have created and seen the results. First of all time delays create unstability on the system. So it is interesting to know which is the maximum delay of the system to be sure that the system will work correctly.

But it is not enough, we have to see which is the form of the input of the system. Indeed we have seen that a soft curve is easier to follow than a curve with angle : the angle creates a discontinuity which rises up the unstability. We have seen the maximum delay with a square curve to follow is smaller than in the case of a circle to follow.

I have discovered, too the delay can be bigger if the system has only to reach a point than when the end effector has to follow a curve. There is the same problem with transport time delay and the control time delay. Indeed with the transport there is no loss of information contrarly to the control time delay. So the second time delay is worse than the transport time delay for the stability of a system.

Hopefully the delay can be compensated by many ways as the use of Smith predictors see[12] or use of Kalmans predictors(see[13]), we can use also a feed forward controller. This can be useful often because in system there is allways time delays. So if we can correct them the system will have a better behavior and can be stable even with big delays.

6.3 Future works

First of all it should be interesting to make experiments on time delay with a real robot. Indeed the simulation is not exactly the same as the model, we have design the dynamic of the robot as a linear function whereas it is non linear. And we have design time delay by stochastic methods so it can be better to make simulation with real time delays created by a real network and the computation time of the image processing.

As said before we can compensate the delay by adding predictors. Consequently it can good to work on the predictors for visual servoing with time varying delay for instance. The results of such works should be useful for hard real time system with a small tolerance to the delay.

7 References

[1] E. Trucco and A. Verri. *Introductory Techniques for 3D Computer Vision*. Prentice Hall, New Jersey, 1998.

[2] J.Nilsson. *Real-Time Control Systems with Delays*. PhD Thesis, Departement of Automatic Control, Lund Institute of Technology,LTH Sweden. (1998)
ISRN LUTFD2/TFRT--1049--SE

[3] B. Wittenmark, J. Nilsson and M. Törngren. *Timing Problems in Real-Time Control Systems*.Department of Automatic Control, Lund Institute of Technology and Department of Machine Design, Royal Institute of Techology Sweden

[4] E.Casagrande. Dynamic *Vision Shape From Motion.* Master Thesis, Departement of Automatic Control, Lund Institute of Technology,LTH Sweden. (2003)
ISRN LUTFD2/TFRT--570--SE

[5] D.Henriksson. *Flexible Scheduling Methods and Tools for Real-Time Control Systems*. Departement of Automatic Control, Lund Institute of Technology,LTH Sweden. (2003)
5316 ISRN LUTFD2/TFRT--3233--SE

[6] D.Henriksson and A. Carvin. *TrueTime 1.13-Reference Manual*. Departement of Automatic Control, Lund Institute of Technology,LTH Sweden. (2003)
ISSN 0280-5316 ISRN LUTFD2/TFRT--7605--SE

[7] Web site ABB : www.abb.com

[8] Web Site Basler: www.baslerweb.com

[9] www.efunda.com/math/leastsquares/lstsqr1dcurve.cfm

[10] M.T. Tham. *Dynamic Models for controller design*. Dept. of Chemical and Process Engineering University of Newcastle upon Tyne. (1999)

[11] B.Lincoln. *Dynamic Programing and Time Varying Delay Systems* . Departement of Automatic Control, Lund Institute of Technology,LTH Sweden. (2003)
ISSN 0280-5316 ISRN LUTFD2/TFRT--106--SE

[12] Karl Johan Åström and Björn Wittenmark. *Computer Controlled Systems—Theory and Design*. PrenticeHall, Englewood Cliffs, New Jersey, second edition, 1990.

[13] M. Bourmpos. *Vision Based Robotic Grasping Tracking of a Moving Object*, Lund Institute of Technology,LTH Sweden. (2001)
ISSN 0280-5316 ISRN LUTFD2/TFRT--567--SE

[14] C-Y Kao and Lincoln B. *Simple Stability Criteria for Systems with Time-Varying Delays*. Submitted to Automatica

[15] A. Carvin. *Integrated Control and Real-Time Scheduling*, Lund Institute of Technology,LTH Sweden. (2003)
ISRN LUTFD2/TFRT--1065--SE

Appendix

A Simulink Models:

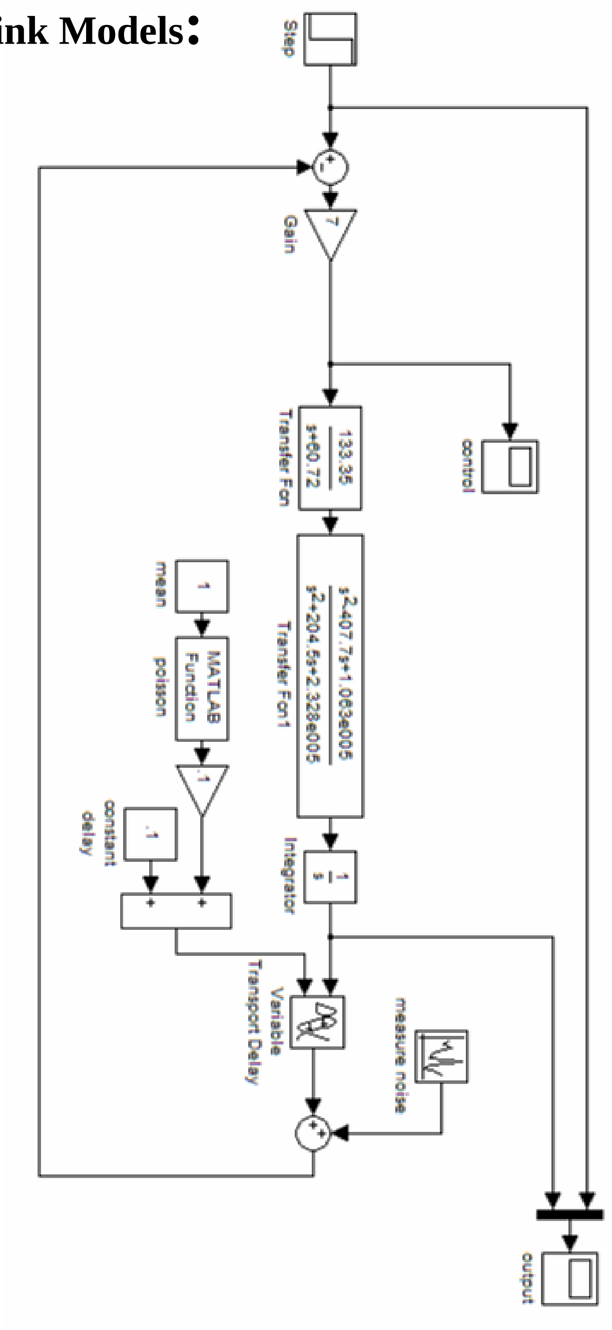


Figure A.1: full simulink model of one joint simulation

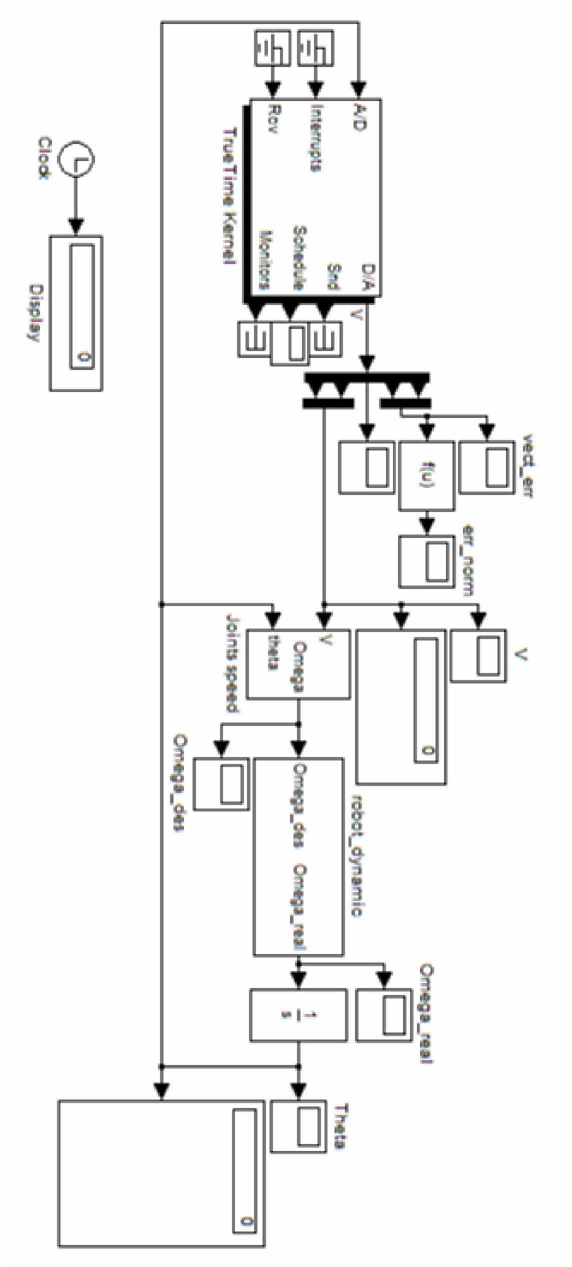


Figure A.2 : full model of the robot

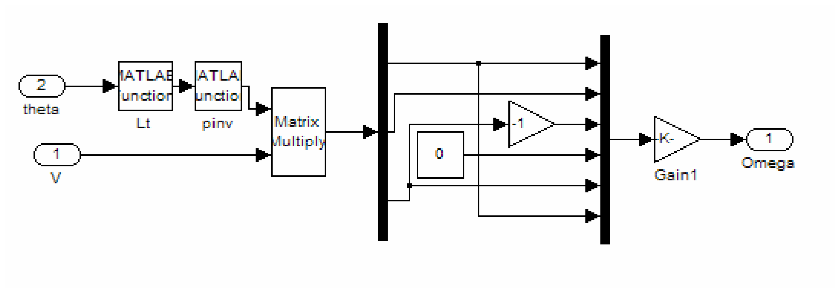


Figure A.3 : transformation bloc from end effector velocities to joints velocity

B Matlab Code of the image processing algorithm:

```
function [point,theta]=image_processing(matrix,old_point, old_theta);

closest_point=[1 1];
%%research of the closest point of the curve
for i=13:1:43
   for j=13:1:43
      % test if the pixel has the good level
      if (matrix(i,j)>25&matrix(i,j)<120)
         member=1;
         m=-2;
         %test if the neighborhood is a part of the line
         while (member==1&m<3)
            n=-2;
            while (member==1&n<3)
               if (matrix(i+m,j+n)<25|matrix(i+m,j+n)<matrix(i,j)*7/8)
                  member=0;
               end
               n=n+1;
            end
            m=m+1;
         end
         % test if the point is closest than the ealier found
         if member==1
            X=old_point-[27 27]+[i j];
            if norm(X-[120 160])<norm(closest_point-[120 160])
               closest_point=X;
               Y=[i j];
            end
         end
      end
   end
end
coord=zeros(442,2);
k=0;
% find the coordinates of pixel owned by the curve in the neighborhood of the closest
point
for i=-10:1:10
   for j=-10:1:10
      if (matrix(Y(1)+i,Y(2)+j)>25&matrix(Y(1)+i,Y(2)+j)<120)
         member=1;
         m=-2;
         while (member==1&m<3)
            n=-2;
            while (member==1&n<3)
```

```
                if
matrix(Y(1)+i+m,Y(2)+j+n)<25||matrix(Y(1)+i+m,Y(2)+j+n)<matrix(Y(1)+i,Y(2)+j)*
7/8
                    member=0;
                end
                n=n+1;
            end
            m=m+1;
        end
        if member==1
          k=k+1;
          coord(k,1)=i;
          coord(k,2)=j;
        end
      end
   end
end
%computation of orientation (Least Square Method)
A=zeros(2,2);
C=zeros(2,1);
% calculate the orientation with two methods to avoid singularity
if(old_theta>45&old_theta<135||old_theta>225&old_theta<315)
      A(1,1)=norm(coord(1:k,2))^2;
      A(1,2)=sum(coord(1:k,2));
      A(2,1)=A(1,2);
      A(2,2)=k;
      C(1,1)=coord(1:k,1)'*coord(1:k,2);
      C(2,1)=sum(coord(1:k,1));
      Ba=pinv(A)*C;
      alpha=pi/2-atan(Ba(1));
   else
      A(1,1)=norm(coord(1:k,1))^2;
      A(1,2)=sum(coord(1:k,1));
      A(2,1)=A(1,2);
      A(2,2)=k;
      C(1,1)=coord(1:k,2)'*coord(1:k,1);
      C(2,1)=sum(coord(1:k,2));
      Bb=pinv(A)*C;
      alpha=atan(Bb(1));
   end
point=closest_point;
% determinate the direction to follow according to the old direction.
if dot([cos(old_theta*pi/180) sin(old_theta*pi/180)],[cos(alpha) sin(alpha)])<0
   theta=mod(alpha*180/pi+180,360);
else
   theta=mod(alpha*180/pi,360);
end
```