ISSN 0280-5316
ISRN LUTFD2/TFRT--5675--SE

Vision Based Robotic Grasping
Tracking of a Moving Object

Michail Bourmpos

Department of Automatic Control
Lund Institute of Technology
September 2001

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
September 2001

Document Number
ISRN LUTFD2/TFRT--5675--SE

Author(s)
Michail Bourmpos

Supervisor
Rolf Johansson, Johan Bengtsson, LTH.
Mathias Haage Datavetenskap, LTH.
Martin Clark Imperial College

Sponsoring organization

Title and subtitle
Vision Based Robotic Grasping Tracking of a Moving Object. (Robotseende för att följa och gripa ett rörligt objekt).

Abstract
Real-time vision allows the direct steering of a robotic manipulator. In this thesis we use a stereo rig mounted on an industrial robot (ABB IRB-6/2) which gives us information about the end effector position of a second robot (ABB IRB-2000/3). This second robotic manipulator performs the task of tracking a moving object, which in our case is a rolling ball.

The first concern of this project is to deal with the different kinds of problems occurring in such a system. After measuring the time-delays in our system we can establish the degree in which they effect it. We can then use these results for the actual control loop. More specifically a Kalman predictor scheme is implemented to produce estimates of coordinates for requested times, based on the time delay analysis. This Kalman predictor also helps us when we have lost track of either the robotic manipulator or the rolling ball.

Finally we perform the actual experiment of tracking the rolling ball. The experimental setup consists of the two robots and the stereo rig mentioned above, as well as a metallic ball rolling on a metallic board. The IRB-6 is mounted with the stereo rig and is set in a pre-decided position to be able to observe the movement of the rolling ball throughout the whole of its course. The other robotic manipulator performs the task of tracking the rolling ball, using as feedback the data received from the stereo rig.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

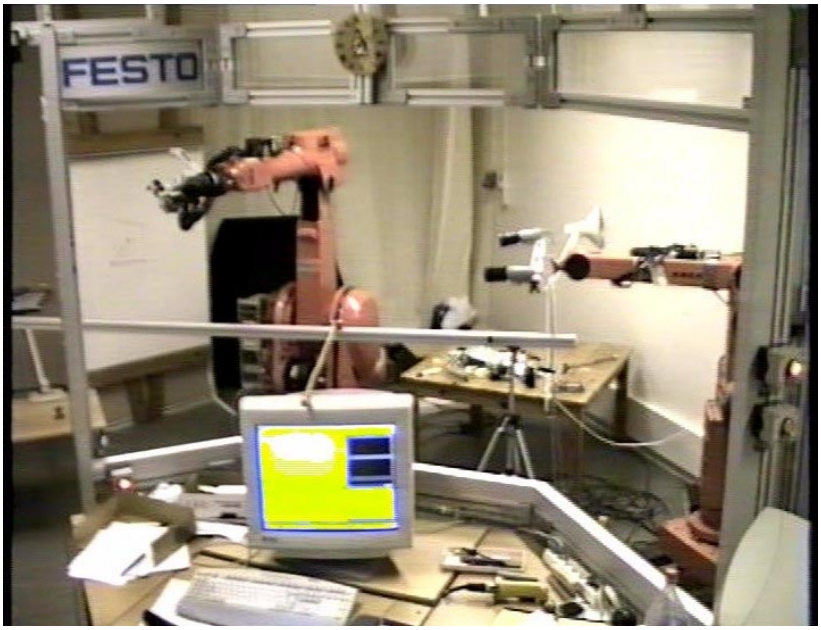
Language
English

Number of pages
75

Recipient's notes

Security classification

VISION BASED ROBOTIC GRASPING
TRACKING OF A MOVING OBJECT



Michail Bourmpos

Department Of Automatic Control
Lund Institute Of Technology
September 2001

ACKNOWLEDGEMENTS

Being an MSc student in the Department of Electrical and Electronic Engineering at Imperial College Of Medicine And Science, 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 Dr Martin Clark from the Department of Electrical Engineering at Imperial College Of Medicine And Science for giving me such an opportunity. The experience proved to be invaluable to me.

Through the course of this project I was involved in a team work procedure which had as a purpose the combination of different fields of research, like vision and robotics, to achieve an ultimate goal, explained in this report. This team was mainly consisted by Johan Bengtsson and Mathias Haage, current PhD students in the University of Lund, at the Department Of Automatic Control and the Department of Computer Science respectively. I would like to express to them my sincere gratitude for their help and their 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 Anders Robertsson and Leif Andersson have given me and for that I would like to thank them.

Finally I would like to mention the people that have made possible the whole MSc course and consequently this thesis project for me, my parents. Without their emotional and economical support I couldn't possibly have come this far. I hope the result matches their efforts and their expectations.

Contents

1 Introduction

The two basic fields of research, which have been combined for the development of this master thesis, are robotics and vision. Both fields have seen rapid development over the past few years. We will now attempt to give a brief historical introduction about robotics and vision as well as a few basics about these two research fields.

1.1 Robotics

The use of the industrial robot along with computer aided manufacturing systems (CAD) begun in the 1960's. From then until now rapid changes in the research area of robotics have been made. The automotive industry has made large investments on the robot industry and therefore has played a crucial part in its development.

Hardware and software developments in the area of computing machines have made modern robots capable of taking part in far more applications than before. Their performance has increased as well as their range of abilities.

Robots with restricted sensor feedback are limited in the ways they can behave. However this is the way they are currently used in industry. The needs for motion descriptions and operator interactions clearly show that robot control needs its own techniques, see [13].

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. However most natural settings, which are not structured and not easy to model, are bound to cause problems in the control of such a robotic manipulator. In this master thesis besides the

world (Cartesian) and the joint coordinates we shall also use image coordinates, which are introduced from vision.

Although robotics today include a lot of different aspects of human motion, like moving of arms, legs, mobility, tracking etc, in our project we will focus only on the robotic manipulator (robotic arm) and the task of tracking a moving object. To do this we will use visual feedback from a vision system consisting of a rig with two non–calibrated cameras mounted on it (stereo rig)

1.2 Vision

When in robotics we require a feedback control scheme for a task involving manipulation of three–dimensional objects, an easy way to acquire feedback information is through an external sensor. A robotic manipulator may use as feedback the information coming from different kinds of sensors. Some of these sensors require direct physical contact, like force sensors, contact switches and others, like ultra–sound sensors, infrared sensors, laser and digital cameras do not. Feedback from computer vision systems enables the use of robots in non–structured accuracy environments, where the work area is not limited.

Computer vision involves the capturing, understanding and processing of images, see [2]. In the part of the image processing we can encounter many problems. For example it is hard to distinguish objects of different material or even of different geometrical shape because their image could be the same. Therefore it is difficult to interpret images using surface models.

A problem occurs when we try to determine the position of a three–dimensional object using a two–dimensional image. The coordinates that give us the actual depth of the object are difficult to be determined. An example of human processing of an image regarding the depth of an

object is called 'pictorial depth cue'. A 'cue' may be the most familiar size, interposing or occlusion, shade or shaded area, size related to the horizon line, motion and motion parallax, binocular perception (stereoscopy). Furthermore, the determination of all three coordinates and not just depth in Cartesian space, from X and Y coordinates in image space is also a hard task to accomplish.

Stereoscopy is the study of corresponding images for the recreation of three–dimensional coordinates. Depth is calculated from the disparity between at least two images.

In our experiments stereo vision is providing the information for the position of a rolling ball as well as of the position of the robotic manipulator which tracks it. The information acquired from the cameras are analysed and interpreted with 3–D vision techniques into image and Cartesian coordinates, see [2]. These coordinates are used as feedback for the control scheme that acts on the robotic manipulator.

1.3 Earlier Projects

Base for this project have been a few earlier projects and thesis assignments carried out in the Department of Automatic Control, in the University of Lund. First of all I should mention the most recent work of Luis Manuel Conde Bento and Duarte Miguel Horta Mendonca, called 'Computer Vision and Kinematic Sensing in Robotics', [7]. This project involved visual servoing with cameras of a moving feature point (the center of a cross). The goal was to keep the feature point centred inside the images captured by two cameras, mounted on a robotic manipulator. However, in this project the cameras were initially calibrated, something that was not done in this thesis assignment.

Even though the above mentioned experiment used calibrated cameras, it was a good experimental setup for measuring variable time

delays occurring in the system. Furthermore, it proved to be a good base for the experiment that was carried out in this work.

Another project that has provided useful information about our experimental setup and also with a way to provide some safety for the actual experiment, was that of Tomas Olsson on 'Vision–supported Force–controlled Robotic Grasping', [8]. This work involved the grasping of a marker through vision feedback. From this project we obtained information about the position of the cameras with hard–eye calibration and with respect to the robotic manipulator IRB2000 and so we were able to estimate the position of the robot in image coordinates by measuring its joint angles. This estimation is only used when we lose track of the robot TCP, something that happens rarely in our experiment.

Older projects that were very helpful to understand the nature of the problems involved in such an experiment were the doctoral dissertations of Klas Nilsson on 'Industrial Robot Programming', [11], of Johan Nilsson on 'Real–time Control Systems with Delays', [6] and of Anders Robertsson on 'Observer–Based Control of Nonlinear Systems', [10]. Finally I should mention the master thesis work of Johan Bengtsson and Anders Ahlstrand with the title 'A Robot Playing Scrabble Using Visual Feedback', [9]. All the above projects that were carried out in the Department of Automatic Control in the University of Lund, have been providing useful ideas and the theoretical base for the realisation of our experiment.

1.4 Thesis Outline

This master thesis report is organised as follows:

In the next chapter we introduce the problems of time delays and occlusions in our system. We will explain what kind of time delays we

have and we will present the results of our measurements. Other problems encountered in this project will also be mentioned.

Chapter three describes a possible solution to the time delay and occlusions problem, using a Kalman filter for purposes of prediction. In order to explain in detail the compensation for the time delays, the introduction of time stamps in our data is required and so a few comments on time stamping will also be made. Solutions given to all other problems encountered will also be thoroughly explained.

In chapter four we give an overview of the system, its architecture and all software components we have used in the experiment. We will briefly mention the characteristics of the two robotic manipulators and the cameras used. We will also list and describe the different modules needed for the control of the movement of both robots. Then the experiment of tracking a moving object is described. The vision and control algorithms are also discussed.

In chapter five we present our results for the performance of our time delay algorithm and for the tracking of a moving object experiment. Finally we include our conclusions and our suggestions for future work

2 Problems Formulation

2.1 Time Delays

Real–time control systems, like the one shown in figure 2.1, are inevitably effected from the time delays occurring. These time delays are introduced from the communication network existing between each node of the system.

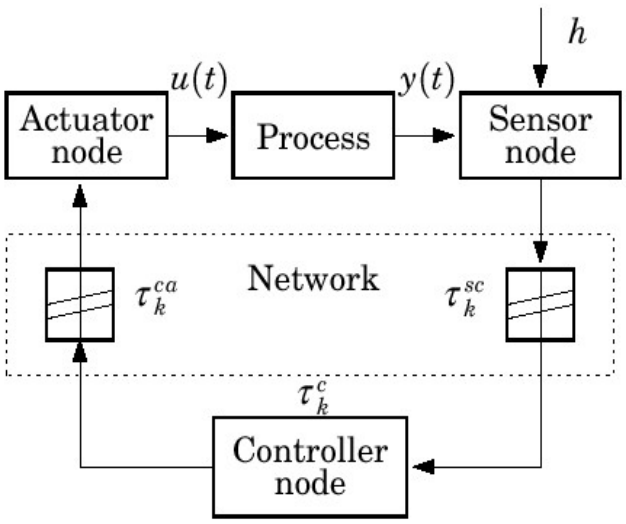


Figure 2.1 : Time delays in a system

In such a system the different time delays occurring, see [6], 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

For our system we can easily identify the communication delay between the sensor and the controller as the time it takes for a captured from the cameras image to become available for processing. The computational delay of the controller then should be the image processing time plus the communication time between the computer that performs the image processing and the computer that performs the control algorithm, plus the computational delay of the control algorithm itself. The time it takes for the computer that simulates the controller to communicate and send the controller output to the robot can be identified as the communication delay between the controller and the actuator.

We will examine these delays by first measuring the time it takes to receive an image from the cameras (communication delay between sensor and controller) and finally by measuring the total communication delay, which from now on will be referred to as system lag.

Receiving Images From The Cameras

The hardware setup of the cameras is consisted of the inner capabilities of the cameras as well as the IEEE1394–1995 network connecting them to the PC which receives and processes the images. The hardware definitely effects the total time delay introduced by this communication. However, we will now focus only on the actual measurements and the observations that may be derived from them. The description of the hardware setup will be described in detail in the chapter regarding the experimental setup for this project.

By injecting a few lines of code in the C++ code that performs the communication between the PC and the cameras, we note down the

actual time when a request for an image is made and the corresponding time when the image is received. Because such a time is expected to be the size of milliseconds we keep the actual times for a hundred such images to be received. With these measurements we build the following histogram, regarding the time delay of one image to be received:

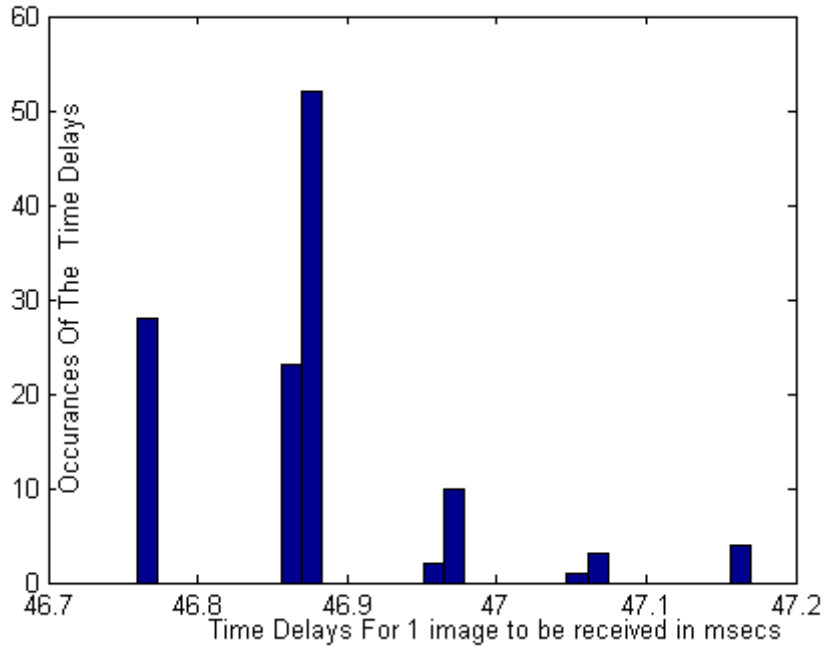


Figure 2.2 : Histogram for time delays of one image, from moment of request until moment of reception

We then calculate the mean of the time delays occurring in the system for a hundred of these images to be received, which is 4678 msecs and the standard deviation for these time delays, which is 9.38. The mean and variance calculated, assuming Gaussian distribution for the image acquisition time, can be divided by a hundred to obtain the mean and variance for the time delay of one image to be received. The new mean and standard deviation corresponding to the one image time delays are 46.87 msecs and 0.0938 respectively.

The time delays occurring are variable for each camera and for each sample. However the overall mean and variance are almost the same and that is why we present one general measurement for both cameras.

Finally we should mention that this time is actually the time from when the request for the image is made until the image is received.

Communication Delay Between Controller and Actuator

For time delays occurring in our system, no measurements were needed. The computer simulating the controller and the robots communicate through a software package called matcomm. This software, which will be analysed later on in the experimental setup chapter, has known performance of 1 msec per sent data package

System Lag

In order to obtain the measurements for the complete system time delay we should first synchronise the two clocks of the two computers involved in the experiment. After synchronisation is achieved we calculate the offset of the two clocks which will later on be used in our time delay computation. A few lines of C++ code were injected into the existing code running in the Windows NT machine, so that any information sent to the UNIX machine would carry with them a time stamp. This time stamp would be the image acquisition time according to the Windows NT machine clock.

The UNIX machine receives the information sent by the PC and the controller action is simulated. When the actual controller is fed with the information received from the Windows NT machine, the actual time delay is calculated. The time of the UNIX machine is taken and then the PC time that was transferred along with the data is subtracted from it.

The offset of the two clocks is then subtracted from the result and the final time delay is saved. The time delays can be formulated as follows:

$$t_d = \{Current\ UNIX\ time - Synchronization\ UNIX\ time\} -$$
$$-\{Transferred\ Windows\ NT\ time - Synchronization\ Windows\ NT\ time\} =$$
$$= \{Current\ UNIX\ time - Transferred\ Windows\ NT\ time\} - \boldsymbol{Offset} \quad \textbf{(2.1)}$$

where,

$$\boldsymbol{Offset} = \{Synchronization\ UNIX\ time - Synchronization\ Windows\ NT\ time\} \quad \textbf{(2.2)}$$

The results of our measurements can be seen in the next figure:

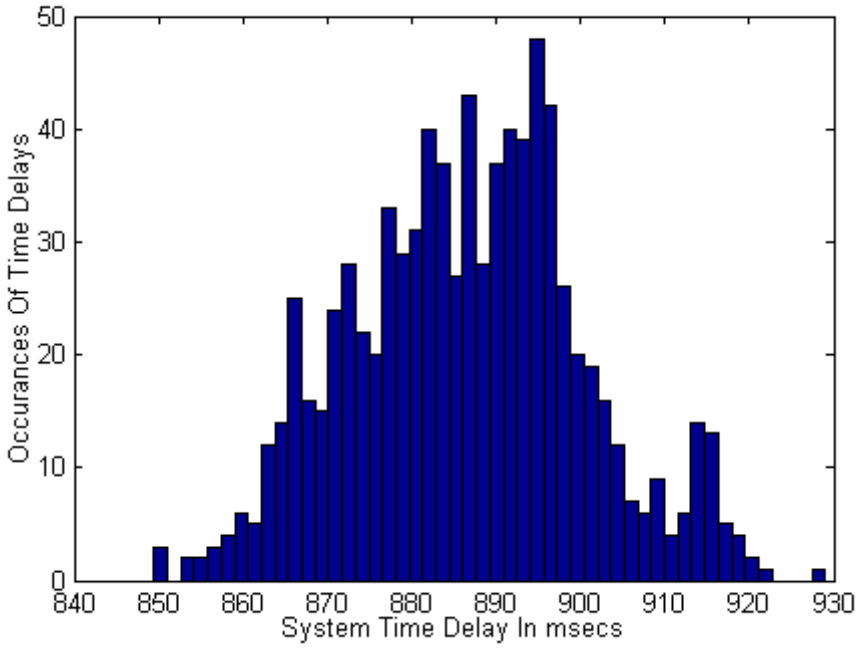


Figure 2.3 : System lag with the old vision software

The mean of the time delays occurring in our system is 886.56 msecs and the standard deviation is 13.819.

Due to the fact that the original image processing code used was rather slow, thus making the time delays occurring in our system quite large, we decided to use another image processing software. This software was also built in C++ and was created by Mathias Haage, a current PhD student at the University of Lund. New measurements were taken and are presented in the histogram of figure 2.4.

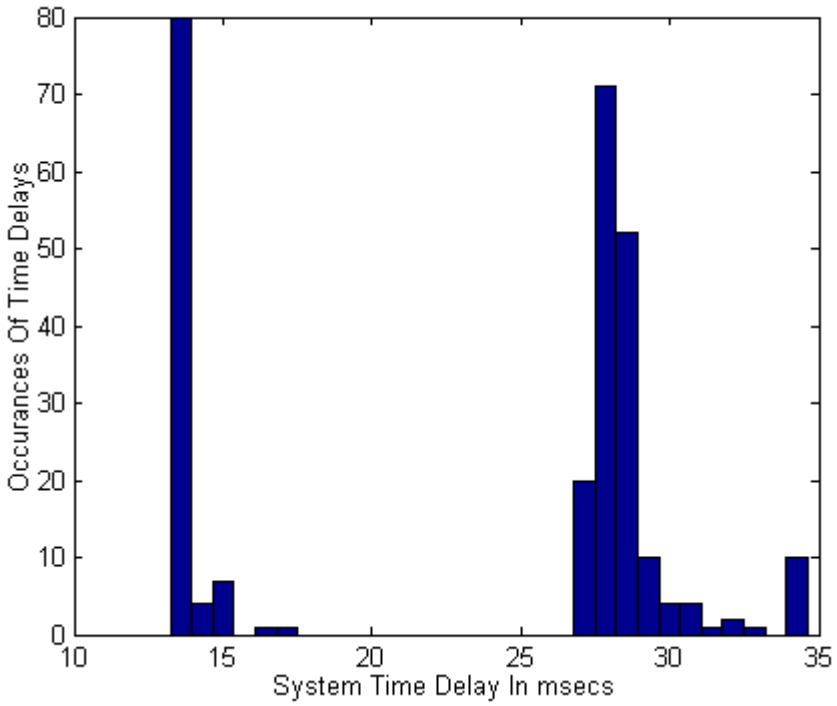


Figure 2.4 : System lag using the new vision software

We can see that the new time delay measurements reveal that the new software has improved the performance of our system, in terms that the time delays are now much smaller. The new mean of time delays occurring in the system is 23.5117 msecs and the new standard deviation is 7.1841.

At this point it would be useful to explain why we have used the latter measurement as the system lag and not include the time required for communication with the robot or even the computational delay introduced from the simulation of the controller. To justify our selection we will refer to the fact that both times for the above procedures are rather small. It takes around 1msec to communicate with the robot and another 5msecs to perform all computations for the simulation of the controller. Furthermore, we are much more interested in the error introduced by the time delay of our feedback scheme, since this time delay is much larger. This is the only part of our code where we might be able to improve performance. The time required to perform the tasks we have left out of our system lag measurements are rather small and there is really not much space to improve. However, there might be something we can do to improve performance of our system and compensate for the calculated system lag, as we shall discuss in the next chapter. Then it will become obvious that the time delay measurement required for our compensation algorithm involving the Kalman predictor is exactly the one we are currently measuring

2.2 Clock Synchronisation

In the above determination of the overall time delay of the system the synchronisation of the clocks between the two different computing machines used was necessary. The two machines used were:

a) a PC system running Windows NT which performed the reception of the images and their processing

b) a UNIX system that simulated the controller and communicated with the robot.

The problems arising in such a task were the precision of the two clocks and the communication time between the computers. First of all the Windows NT machine has a clock with precision of more than 1msec, while our UNIX machine had precision much less than a millisecond. It is obvious that when dealing with time delays of a few hundreds of milliseconds the error in synchronisation of size of 1 msecs is not so large and so we decided to live with such an error.

The idea for synchronising the two clocks was to send via matcomm the clock of the Windows NT machine to the UNIX machine and then, at the same instant to keep the time of the UNIX machine. Both times are kept in milliseconds. For the Windows NT time we must say that we are referring to the absolute time from the beginning of the application 'Camera Server', which is used for communication with the cameras and image processing. For the time kept in the UNIX machine we have used the absolute time of the machine in milliseconds. With these two times and the measured mean time for communication between the two computers we calculate the offset between the two clocks. This offset is used in our calculations of the overall system time delay.

It is obvious that using the mean time for communication introduces another slight error in our synchronisation procedure, but this is of size less than a half of a millisecond and so it is neglected.

All the above procedure was realised by injecting a few lines of code both in the software running in the Windows NT machine (C++ code) and in the one running in the UNIX machine (MATLAB SIMULINK models).

2.3 Visual Occlusion

Another problem we had to face due to the nature and the setup of our experiment was that of occlusion. The robot TCP follows the rolling ball and is always closer to the cameras. In this case it is possible that the

ball is occluded by the robotic manipulator for a few samples. Inevitably we lose track of the ball and the question of what to do in such a situation arises.

Occlusion is a standard problem when dealing with systems that use visual feedback. The simplest solution would be to assume that for these few samples a normal linear interpolation would be enough. As we shall see later on this idea was not used. The fact that we would have to compensate for the time delays mentioned above with a prediction model, made it obvious that we could use the same predictor to overcome the occlusions problem as well. In the next chapter, when we will describe the predictor created, we will see how effective this approach has been to the problems caused by occlusion.

2.4 Noisy Data – Loss Of Tracking

The next problem we had to face was somehow similar to that of occlusion. The image processing algorithms used was sensitive to noise caused by lighting conditions. When dealing with the tracking of the robotic manipulator this noise was rather small and so no compensation was needed. However the noise introduced in the tracking of the ball in the images was quite large. Figure 2.5 presents the information sent by the image processing algorithms regarding the position of the ball in image space coordinates.

As we can see there are lots of jumps in the ball trajectory both in X and Y image coordinates, with quite large amplitude. The expected trajectories, based on the physical model of the ball motion are smooth trajectories, like the ’ideal trajectories’ shown in the figure. The high amplitude of the noise introduced created many problems in the correct controlling of the robotic manipulator and therefore in the performance of the system.

Besides this high amplitude noise in our data we have experienced and some cases where due to lighting conditions we completely lose track of the rolling ball.

In all the above mentioned cases we should find a solution that would allow us to send correct data to the controller and therefore improve the performance of our system. This solution would once again involve the Kalman predictor.

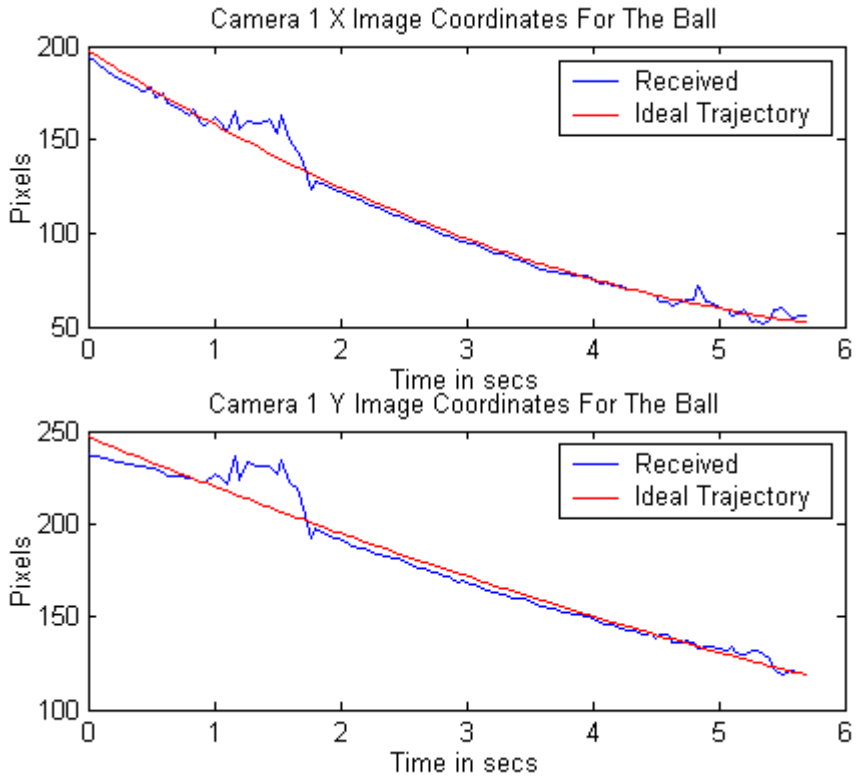


Figure 2.5 : Noisy data received and ideal trajectories

2.5 Singularities

A problem that was rather unexpected when we started working on the project, but we had to face after all, was that of singularities. With this term we refer to the different positions that the robotic manipulator can take in order for the robots TCP to achieve a position in Cartesian space,

see [1], [5]. As we can see in the following figure there might be several positionings of the robot for one position of the robotic end effector.

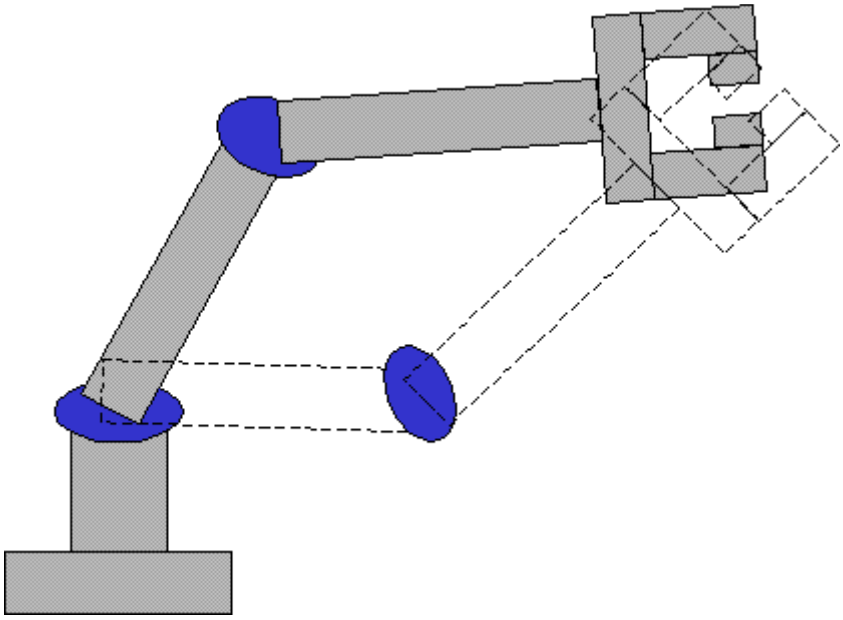


Figure 2.6 : Different positionings of the robot for acquisition of the same Cartesian point

The calculations done in our trajectory generation algorithm require the use of transformation from the robots TCP position in Cartesian space into the robots angle values in joint space and vice versa. While the second transformation is unique the first might have multiple solutions. Both kinds of transformation are required since our control algorithms operate in Cartesian space while the robot system requires trajectories generated in joint angles for its motion.

For the transformation from joint angles into Cartesian coordinates of the TCP forward kinematics has been used, see [1]. The MATLAB function performing this task is called **forward2400.m** and was created by Anders Robertsson. This function uses the joint angles of the robotic manipulator in combination with the length of the tool attached on the robots end effector to calculate the Cartesian coordinates of the robots TCP, with axis the base of IRB–2000.

For the transformation from Cartesian coordinates of the TCP into joint angles the process of inverse kinematics is used, see [1], [5]. We should note that these calculations are done through a MATLAB function called **invkin2400.m** also created by Anders Robertsson. This function uses the theory of inverse kinematics in combination with the Cartesian coordinates of the robots TCP and the length of the tool attached in the end effector of the robotic manipulator to calculate a possible solution for the values of the joint angles for the robotic manipulator.

The solution provided by **invkin2400.m** is calculated having no information on the current position of the robot and so it is possible at certain points in Cartesian space to create large jumps in our movement. Exactly this situation was the one we had to face. At three specific areas in our movement we noticed rapid movements of 180 degrees in joint 4, which is interpreted as the movement of the arm of the robot. This singularity areas should be avoided at any cost, in order to keep the movement of the robotic manipulator in our experiment as smooth as possible.

2.6 Constant Orientation Of The End Effector

The last problem we had to face regarding the positioning of the robot with respect to the stereo rig was that of constant orientation of the gripper. This requirement emerged from the necessity to track the end effector at all times. Based on the above need it became obvious that we should not only be able to control the position of the robotic manipulator but the orientation as well.

The image processing software, in the case of tracking the end effector of IRB–2000 , works as follows. A black tape surrounded by white background which is placed on the gripper of IRB–2000 is checked by the user on the image display of the software's graphical user interface(GUI).

From that moment on that black tape is traced. It easily understood that different orientations in combination with light reflections make it harder and sometimes impossible for the image processing software to keep track of the end effector. That is basically the main reason that we desire, as much as possible, a constant orientation of the gripper with respect to the stereo rig. We must emphasise on the fact that this orientation should not necessarily be such, so that the gripper is parallel to the stereo rig or any of the cameras at any time. The exact orientation was randomly decided by the initial conditions of the IRB–2000 joint angles, in such a way that we are able to have a clear view of the tape attached on the gripper in our cameras.

3 Methods

We have mentioned in the previous chapter the degradation of performance in our system due to time delays. In order to improve our system performance we had to compensate for the errors caused by these time delays using a certain technique. The technique we decided to use involved time stamping and the Kalman predictor.

3.1 Time Stamping

With the term time stamped data we refer to various information which are accompanied by a unique time instant, relevant to these data, see [6]. More specifically, in our experiment we transmit image coordinates for the current position of the robotic manipulator and the rolling ball. But when was the transmitted information really valid. The answer comes from the time accompanying these coordinates.

Time stamped data in combination with an accurately synchronised system makes it possible to calculate the exact time when our data are valid and help us predict what the current real values of the variables are.

In order to be able to transfer time stamped data we had first to synchronise the two clocks between the PC and the UNIX system, as we described in chapter 2.2. Immediately after we receive the images form the cameras, we keep the different times that correspond to each of the cameras. At this point we should mention that the two cameras are not synchronised and operate at 30Hz. This means that the time difference between the data extracted from camera 1 and those extracted from camera 2 could be up to something less than 33msecs. It is obvious that a different time stamp for each of the cameras is necessary since if we

decide to run the experiment in 30Hz or even in 20Hz the time difference of 30msecs is very large. In fact the measured time difference between the acquisition times of the two cameras is around 17msecs, which is also rather large for the case of selecting one time stamp for both cameras.

After the images are processed and the desired coordinates of the robotic manipulator and the rolling ball are extracted, they are saved in an array which also holds the time instants these images were received (time stamping). We should note that this time differs from the time the images were captured, due to transmission delays along the FireWire network used for communication between the cameras and the PC. This time is approximately 6msecs and considering this error to be systematic, we subtract 6 msecs from our timestamps. This way we actually send the time the image is captured.

The decision we made to keep a different time stamp for each camera lead us to the creation of two different predictor schemes. The Kalman predictor seemed the most appropriate tool in our situation.

3.2 The Kalman Predictor

The methods described here are explained analytically in [4], [10], [12]. Let us consider a general estimation problem where the state space model of our system is :

$$\begin{aligned} x_{k+1} &= A\cdot x_k + B\cdot u_k + v_k \\ y_k &= C\cdot x_k + D\cdot u_k + e_k \end{aligned} \qquad \textbf{(3.1)}$$

with $E(e_k)=0, E(v_k)=0$ and $E\{v\cdot v^T\}=R_1$, $E\{e\cdot e^T\}=R_2$

Then the Kalman filter for prediction of x based on the data at time k is:

$$\begin{aligned} \hat{x}_{k+1|k} &= A\cdot \hat{x}_{k|k-1} + B\cdot u_k + K_k\cdot(y_k - C\cdot \hat{x}_{k|k-1}) \\ \hat{y}_k &= C\cdot \hat{x}_{k|k-1} + D\cdot u_k \end{aligned} \qquad \textbf{(3.2)}$$

where

$$K_k = A \cdot P_k \cdot C^T \cdot (R_2 + C \cdot P_k \cdot C^T)^{-1}$$
$$P_{k+1} = A \cdot P_k \cdot A^T + R_1 - A \cdot P_k \cdot C^T \cdot (R_2 + C \cdot P_k \cdot C^T)^{-1} \cdot C \cdot P_k \cdot A^T \quad \textbf{(3.3)}$$

and $P_0 = R_0 = E\{x_0 \cdot x_0^T\}$

The way our problem is formulated we can clearly see that the recursive equations for the calculation at each step of the Kalman gain K_k are of no use, since the variance, correlation and autocorrelation properties of the noise are unknown. So from the above general case we would like to extract simpler equations that apply for our system. The first step towards the extraction of those equations is identifying a good model to match our system.

Identifying The Model

As we mentioned in chapter 2 the basic need for the creation of a predictor came from the time delays, noise and occlusion that effect the ball trajectory in the image space. So the model which we would like to identify should map the model of the ball movement in X and Y image coordinates, counted in pixels.

To obtain the model equation we used the subspace model identification (**SMI**) toolbox for MATLAB, created by Professor M. Verhagen and Dr B. Haverkamp, TU Delft, The Netherlands, see [14]. Our system would have no inputs and would predict the ball movement using information obtained by the identified initial conditions of the state. The initial condition of the ball when entering the image, would however always be different, since the initial velocity and position of the motion differs from experiment to experiment. This proves that we

shouldn't apply just a static state–space model from prediction but use the Kalman gain to adapt to each different motion.

First, using the command **dordpo.m** we retrieve information about the order of the linear time invariant (LTI) state space model we are about to identify. This command also acts as a pre–processor for the command **dmodpo.m** used next, which does the actual estimation of the A and C matrices of the state space model, as well as of the Kalman gain K. After **dmodpo.m** the command **dac2db.m** could have been used for the estimation of matrices B and D, but since our model is assumed to have no input there is no need for such calculation. Finally we estimate the initial conditions of the states of our model, using the command **dinit.m**. We should note that the Kalman gain estimated is a static gain, non–updateable and is calculated in an optimal sense.

The data fed into the identification algorithm came from our image processing code and therefore were not noise–free. In order to make sure that we feed our identification algorithm with the correct data, we process our data in the following way. We calculate from the information provided from the image processing a curve, which goes through the original data, using the least squares criterion. This way we are able to determine a smooth curve that matches as much as possible the real trajectory of the ball in the image and in a way we filter out the noise which is added in our image processing, mostly due to lighting conditions. This last curve holds the data that we input to our identification algorithm, although we still use the real time instants of our measurements.

The above procedure is followed twice, once for the data received from camera 1 and another for the data of camera 2. Then the identification algorithm is executed and the model matrices are estimated. The form of the two predictors is:

$$\begin{aligned} \hat{x}_{k+1|k} &= A\cdot\hat{x}_{k|k-1}+K\cdot(y_k - C\cdot\hat{x}_{k|k-1}) \\ \hat{y}_k &= C\cdot\hat{x}_{k|k-1} \end{aligned} \quad \textbf{(3.4)}$$

, where K is optimally selected.

The state–space models are of second order and the matrices estimated are presented below:

$$\text{Camera 1:} \quad A_1=\begin{bmatrix} 0.9918 & -0.0004 \\ -0.0025 & 0.9846 \end{bmatrix},\ C_1=\begin{bmatrix} -0.1872 & -0.3210 \\ -0.3119 & 0.1886 \end{bmatrix}$$
$$K_1=\begin{bmatrix} -0.1122 & -1.1497 \\ -0.2454 & -0.0274 \end{bmatrix} \quad \textbf{(3.5)}$$

$$\text{Camera 2 :} \quad A_2=\begin{bmatrix} 1.0005 & 0.0059 \\ -0.0086 & 0.9880 \end{bmatrix},\ C_2=\begin{bmatrix} -0.2499 & 0.2574 \\ -0.2492 & -0.2636 \end{bmatrix}$$
$$K_2=\begin{bmatrix} -0.3607 & -0.3904 \\ -0.0742 & -1.3857 \end{bmatrix} \quad \textbf{(3.6)}$$

The two Kalman predictors calculated are inserted in the MATLAB code of our experiment and we shall know describe how they are used for online prediction.

Using The Model On–Line

The most difficult part about running the predictors online was to decide when we should update our predictor using the Kalman gain. The update of the model takes place through the term $K\cdot(y_k - C\cdot\hat{x}_{k|k-1})$. Our

problem was located in measuring the real output y_k which we want to use in order to calculate the error of the prediction, $y_k - C \cdot \hat{x}_{k|k-1}$. Direct noise–free measurement of the real output, which in fact is the X and Y image coordinates of the ball in both cameras, was not feasible.

We somehow should be able to determine if the time–delayed, noisy measured output should or should not be used to calculate the real output. After such a decision is taken we update the state using as y_k the value that is calculated through the information given by the cameras and the image processing. More specifically the two last measurements are splined (interpolated) and the time delay measurements are used to establish the exact output at the desired time instant (figure 3.1). That value is used in the adaptation mechanism of our Kalman predictor.

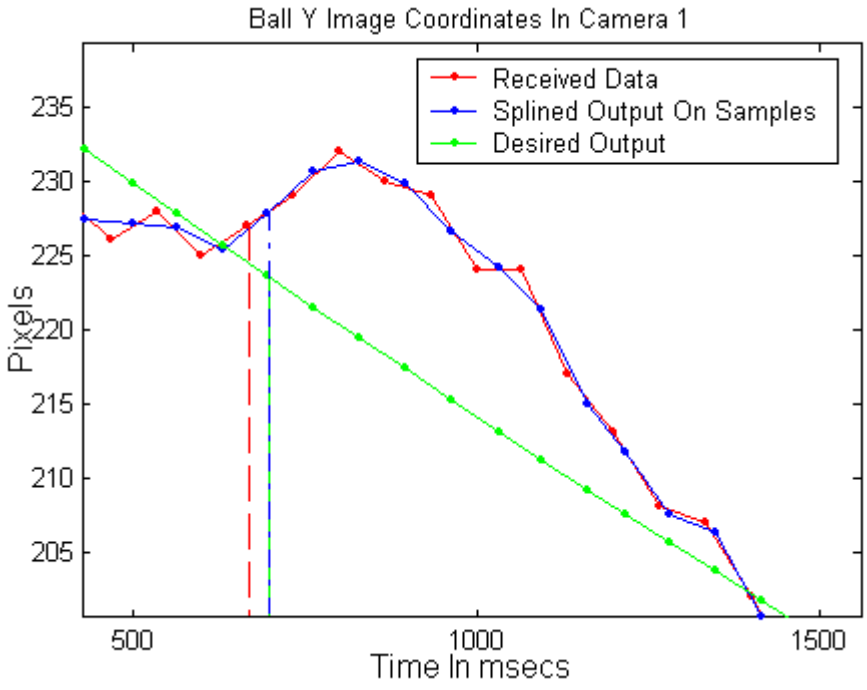


Figure 3.1 : Received time–delayed data, splined data on sample times and desired output

Due to the fact that these splined real outputs can only be estimated once the next measurement has arrived, the calculation of the next values for the state is done prior to the next calculation of the output. So instead of having equations of the form of **(3.4)** we do our calculations in the form:

$$\begin{aligned} \hat{x}_{k|k-1} &= A\cdot\hat{x}_{k-1|k-2}+K\cdot(y_{k-1}-C\cdot\hat{x}_{k-1|k-2}) \\ \hat{y}_k &= C\cdot\hat{x}_{k|k-1} \end{aligned} \qquad \textbf{(3.7)}$$

The results of these calculations will be presented in chapter 5.

3.3 Size Measurements

We have mentioned in the previous chapter that in order to determine if the data provided by the image processing and the cameras are good enough to actually use in the control and in the prediction algorithm, we should have a distinguishing criterion. A simple idea which gave a solution to this need was the use of the size of the object calculated from the image processing algorithm.

Our image processing algorithm is capable to determine the size of the object we are tracking. Using that measurement and setting a threshold for its value we have a good way of determining how good our image is, how much has noise altered our data and in the end if we can or can not use these data. The lower bound of the size measurement of the ball was set at 150 pixels and we can see from the following image that this threshold could prevent us from using data altered by noise up to 20 or 30 pixels. We should note that 20 pixels, when the whole image has a resolution of 240x320 pixels, is rather large error and in our tests we easily notified the disorientation of the robotic manipulator when using these data in the control loop.

This way of establishing how valid our data really are turned out to be extremely useful both in the control loop and in the prediction algorithm. Data that were discarded from the size measurement test were not used in order to update our predictor scheme. In those situations the predictor goes into a feedback–free operation, described by the following equations:

$$\begin{aligned} \hat{x}_{k|k-1} &= A \cdot \hat{x}_{k-1|k-2} \\ \hat{y}_k &= C \cdot \hat{x}_{k|k-1} \end{aligned} \qquad \textbf{(3.8)}$$

It's easy to notice that in these cases the calculation of y_{k-1} is not possible and therefore the adaptation mechanism takes no action at all.

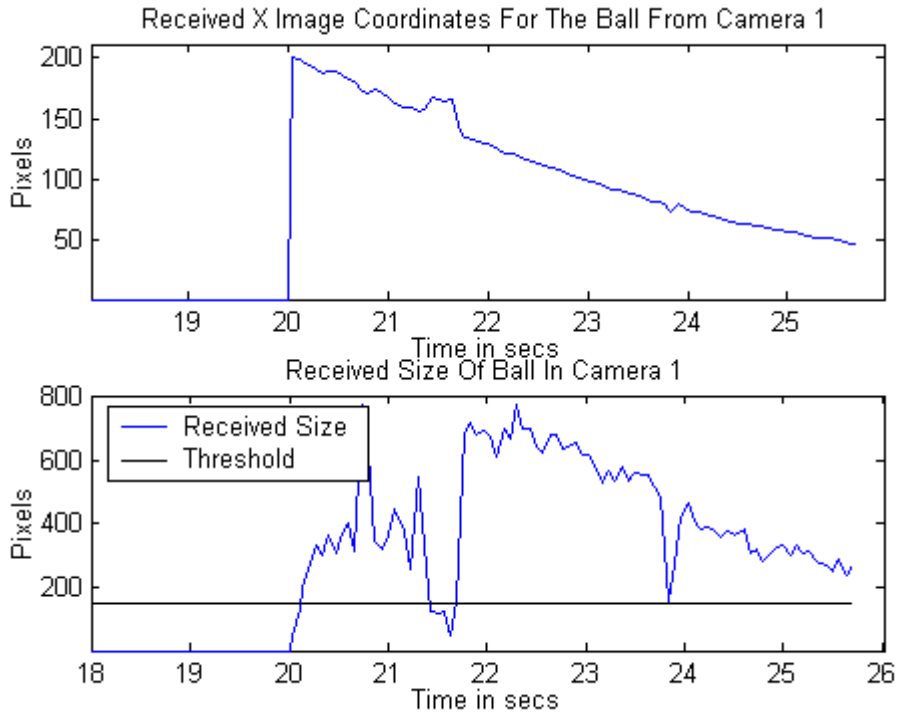


Figure 3.2 : Jump in received X image coordinate and relevant low size

3.4 Locking Joint 4

The solutions that we managed to come up with for the problems we faced with the singularities were the following two. The first concerned the rewriting of the inverse kinematics algorithm using past data information which would enable us to choose the shortest possible move in terms of joint angle rotation. This idea, although it seemed more attractive had the problem that it needed far more information from the system in order to operate and would take up much more time for its execution. Furthermore the time that we had in our hands didn't allow us to get involved in such a time–consuming process.

The second solution which we decided to implement was based on the observation of the kind of singularities we faced in our experiment. It seemed that our real problem was focused on the singularities involving joint 4 of the robotic manipulator, which corresponds to the arm's rotation and takes values in between $\pm 180^{o}$. Since we wanted to prevent this rapid movement of joint 4, see figure 3.3, we decided to lock joints 4, 5 and 6 and do our calculations using the arm's end as base. This was very easy to implement since the only thing needed was to measure the distance between the arm's end and the end effector of the robotic manipulator. This turned out to be $100mm$ and was directly used in the existing inverse kinematics script to calculate the position of the arm's end. The tool length was measured 438mm and so we used this value in the inverse kinematics script in order to get the actual position of the robot TCP in Cartesian space.

From the inverse kinematics script we obtain each time the joint angles for the robotic manipulator IRB–2000 using as tool length $-100mm$. From these values we only use the values of joints 1, 2 and 3.

The values for joints 4 ,5 and 6 are always preset. We keep joint 4 always at -90^{o} and, as we will describe in the next chapter, instead of locking joints 5 and 6 in a preset value we will change those values with respect to the value of joint 1, to accomplish constant orientation of the gripper, throughout the entire movement.

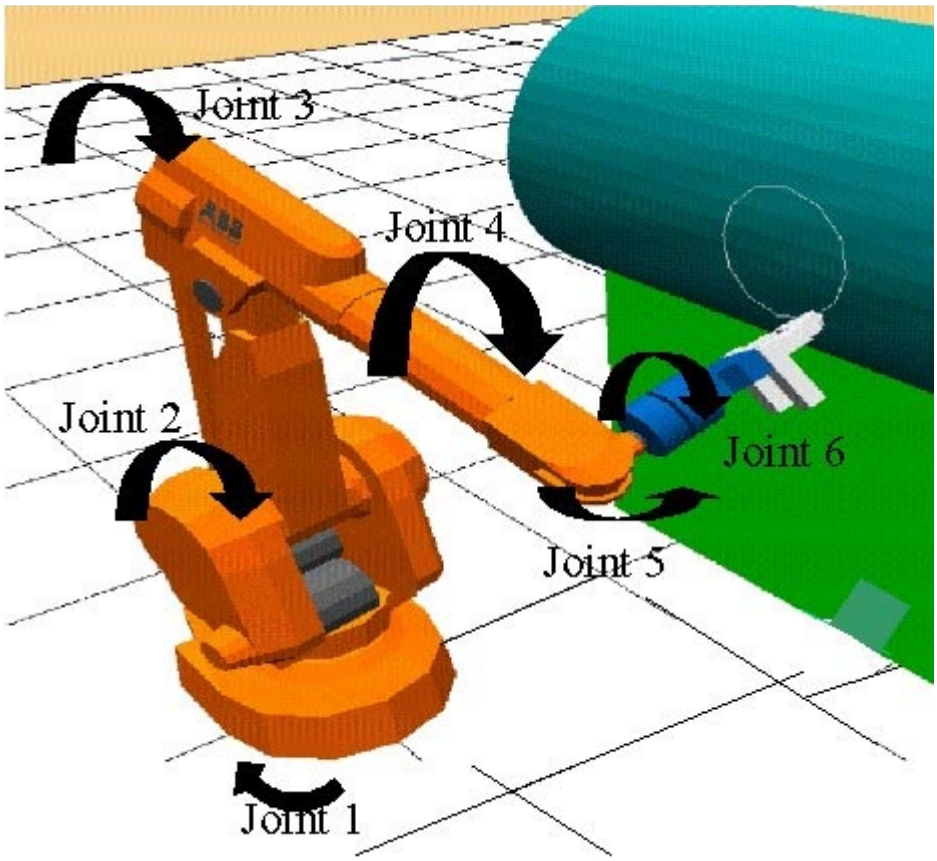


Figure 3.3 : Joints of robotic manipulator IRB−2000

The actual controlling is done with respect to the end effector of the robotic manipulator. Since joint 4 is locked and joints 5 and 6 are moving slowly it is easy to see that a desired motion of the end effector corresponds to the exact same motion of the robotic arm's end. The fact that joints 5 and 6 are slowly moving produces slight errors. These errors are even less when repeating the calculations for a desired position using

as start position our first calculations. The procedure will become clearer when we will describe the whole experiment in detail, in chapter 4.

3.5 Calculation Of Joints 5 And 6

After locking joint 4 the next step was to be able to keep the gripper always almost vertical to the cameras so we can achieve good tracking. The way the tracking of the gripper is done through our image processing software made this task absolutely necessary.

By a series of experiments we have gathered enough data to examine what the movement of joints 5 and 6 should be with respect to joints 1, 2 and 3, see figure 3.3. After all joints 5 and 6 would be enough to achieve constant orientation of the gripper, since joint 4 has already been locked. It came out that joints 5 and 6 should move linearly with respect to joint1 (figure 3.4).

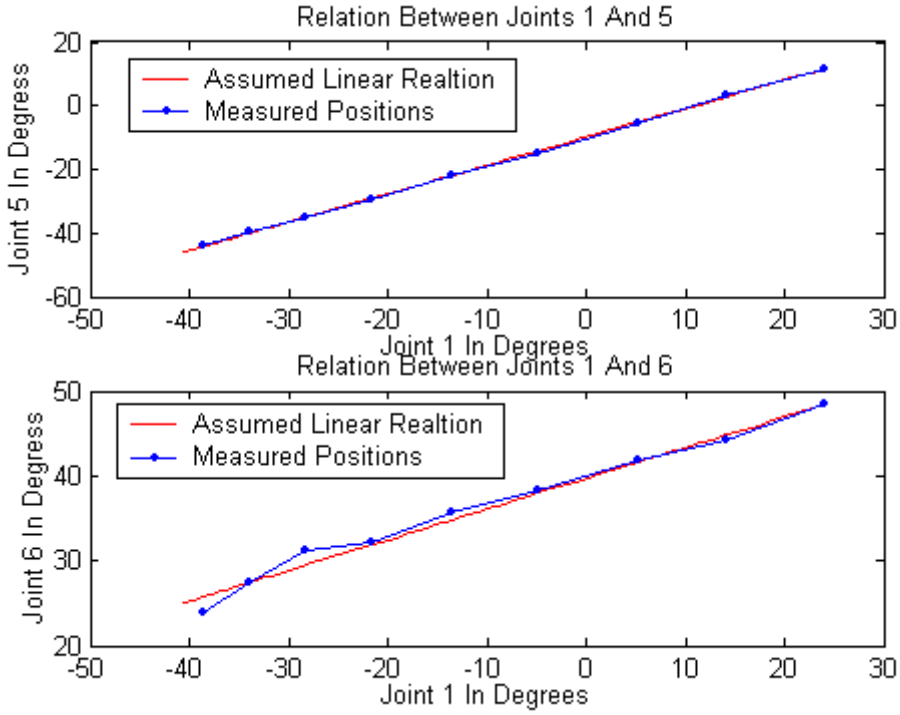


Figure 3.4: Linear relation between joints 1 and 5 and between joints 1 and 6

Using least squares, see [4], [12] we managed to determine what the best coefficients for the description of the movements of joint 5 and 6 with respect to joint 1 should be. The final equations describing these motions are presented below:

$$\begin{aligned} J_5 &= 0.8908 \cdot J_1 - 9.7785 \\ J_6 &= 0.3615 \cdot J_1 + 39.7231 \end{aligned} \tag{3.9}$$

Finally, the calculations of the joint angles for achieving a position in Cartesian space are made through inverse kinematics for joints 1, 2 and 3, with equations (3.9) for joints 5 and 6 and by setting joint 4 to -90^o .

4 Tracking Of A Moving Object

4.1 Experimental Setup

For the acquisition of the time delay measurements we had to perform an experiment, which should use visual feedback on a robotic manipulator. The task of the robotic manipulator was at this point of no importance to us, since we were interested only in the time it takes from when we capture an image to the point when we use the information provided from the image to control the process. For this we have used an existing experimental setup created from Luis Manuel Conde Bento and Duarte Miguel Horta Mendonca. The experiment involved a stereo vision system, which was attached to the robotic manipulator IRB–6 and another robotic manipulator, the IRB–2000, which performed the task of positioning above a still object.

For the experiment of this project, the setup used still consists of these two robotic manipulators. This time, while the IRB–6 was holding the stereo rig and kept still, the other robotic manipulator performed the task of tracking a rolling ball.

The robot software involved uses the open robot architecture, which allows the system to be connected easily to different devices and programs. We can see the actual hardware configuration of the open architecture, see also [11], for the IRB–2000 robot controller in figure 4.1.

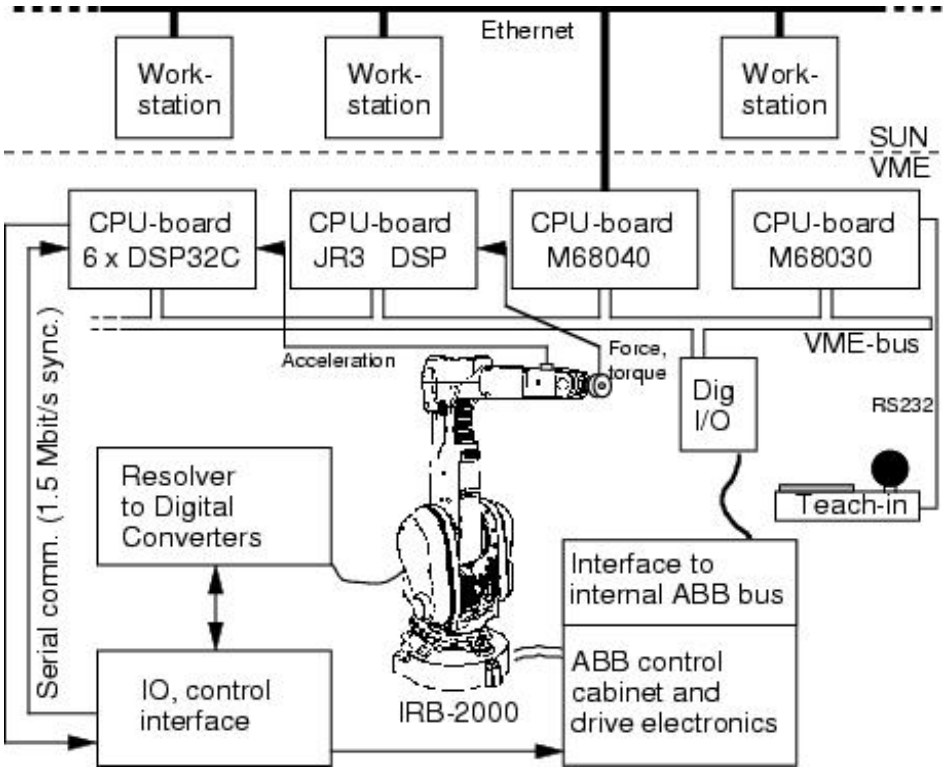


Figure 4.1 : Hardware Configuration of the open architecture robot controller

ABB IRB–2000/3

The characteristics of ABB IRB–2000/3 (Figure 4.2), to which from now one we will refer to simply as IRB–2000 are mentioned in the next few lines. This robotic manipulator has 6 degrees of freedom (DOF) and a precision of 0.1mm. Joints 2 ,3 and 5 are revolute joints, while the other three joints, 1, 2 and 6 are cylindrical joints. The fact that this robot has 6 degrees of freedom allows it to reach all points in the possible work area with arbitrary orientation, see [1], [5].

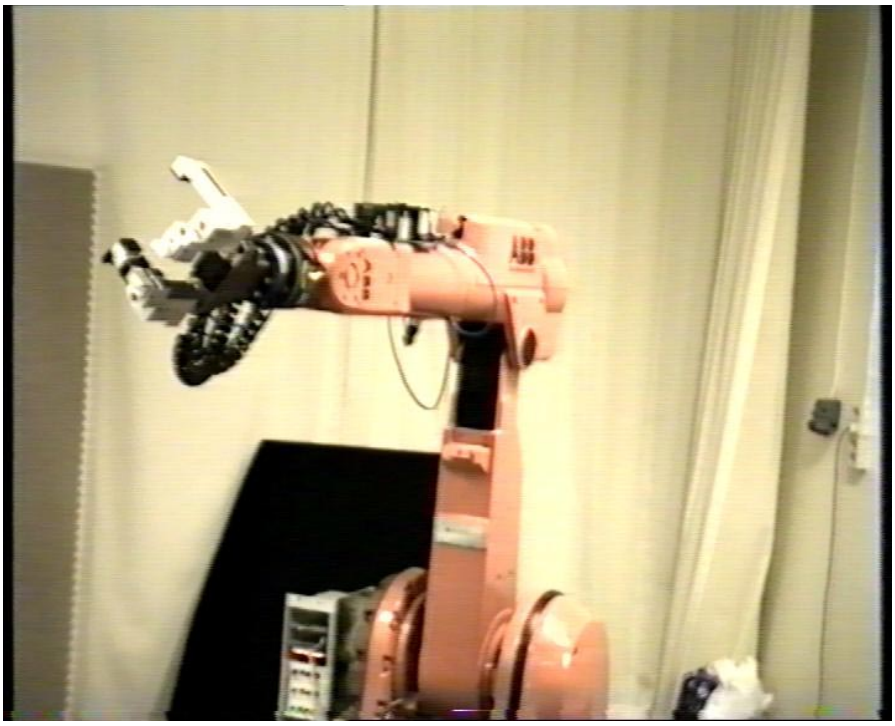


Figure 4.2 : The ABB IRB−2000/3 robotic manipulator

ABB IRB–6/2 And The Stereo Vision System

The ABB IRB–6/2 robotic manipulator (Figure 4.3), which from now on will be simply referred to as IRB–6, has 5 degrees of freedom (DOF). The Cartesian precision of this robotic manipulator is 0.2mm. Joints 1 and 5 are cylindrical joints, while joints 2, 3 and 4 are revolute joints.The stereo vision system consists of two SONY DFW V–300 **cameras** attached to a rig, which is mounted on the end–effector of the robotic manipulator IRB–6 (Figure 4.4). For this stereo rig we assume that the distance and the angle between the cameras are unknowns. The cameras use the IEEE 1394 high performance serial bus to send non–compressed YUV digital data and allow us to perform control functions such as colour tone, brightness, picture quality, white balance and automatic gain control (AGC). However, all the above are preset in our software and have

certain values appropriate for the lighting conditions of our experimental setup.

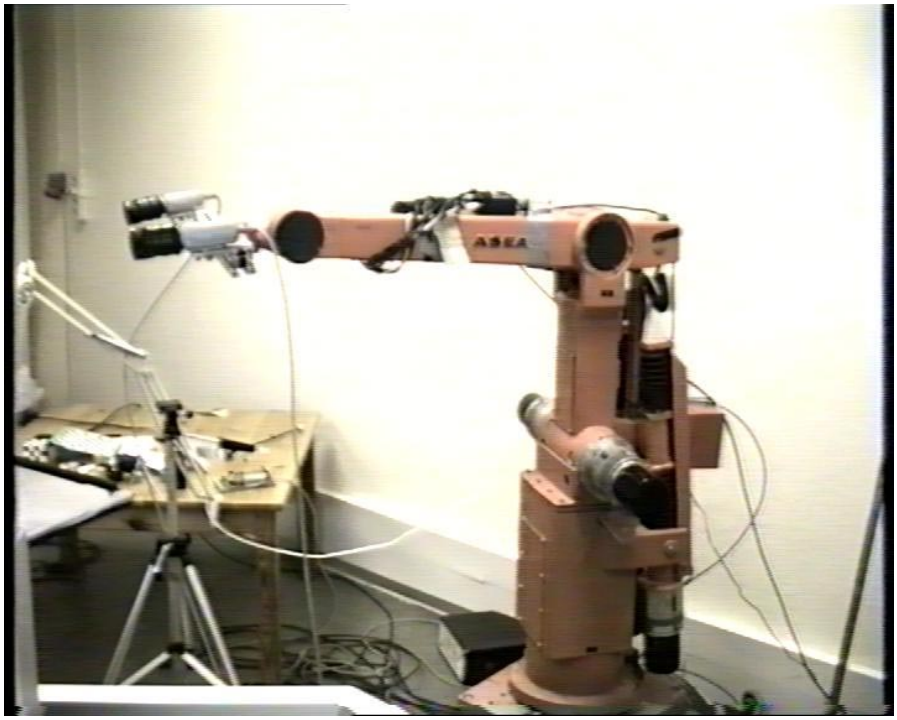


Figure 4.3 : The ABB IRB−6/2 robotic manipulator

The **camera communication** is done by **Fire–I**, which is a standard IEEE 1394–1995 high performance serial bus. The data transmission rates are 100, 200 and 400Mbps and this serial bus has the capability to allow true plug and play (we can add new devices with the system switched on). The camera signals are actually transmitted at 200 Mbits/sec, which means we have a transmission rate of 30 images per second between the cameras and the PC running the image processing software. The picture format is 320 x 240 pixels.

The **cables** used to transmit the images from the cameras are commercial cables capable to sustain data rates up to 400Mbps.

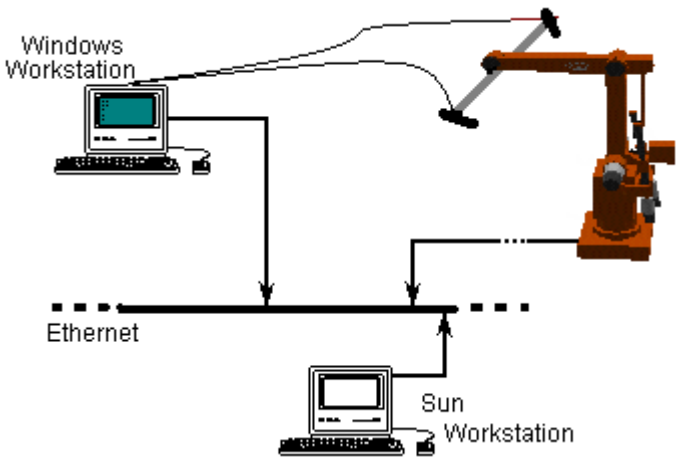


Figure 4.4 : The vision system

Control System And Software

The **control system** used is almost the same for both robots, with slight changes due to the different degrees of freedom the IRB–6 and the IRB–2000 have. It consists of three different modules, see [9], as we can see in figure 4.5. We will now briefly describe these three modules:

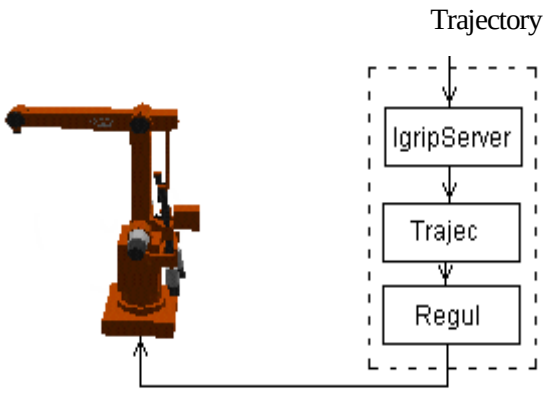


Figure 4.5 : Control system

- The **IgripServer** module communicates through a socket to an application, usually Matlab, which generates the trajectory. The IgripServer receives pre–calculated trajectories from the application and sends them to the Trajec module.

- The **Trajec** module is able to calculate a trajectory or use a pre–calculated trajectory. It then calculates the velocity and the acceleration references for every joint. Trajec sends the position, velocity and acceleration of each joint to the Regul module, which will perform the motion.

- The **Regul** module controls the robot and uses cascaded PI controllers for each joint. The velocity and acceleration are feedforwarded.

The communication between computers within the network consists of the following different types of data transmissions:

- Data transmitted between a SUN workstation running on a UNIX operating system and a Windows NT station

- Data transmitted between the SUN workstation and the IgripServer module

For the above two types of data transmission we have used a software package developed in the University of Lund, Department of Automatic Control, called **matcomm**. This software uses the TCP/IP protocol to transmit data between computers, in a network where different systems might exist. The reason to use matcomm is that it is an easy and fast way to connect with another computer (or module) without the necessity of setting all necessary parameters. Data are sent, in an array format, through matcomm, using sockets.

The final setup is shown in the following figure:

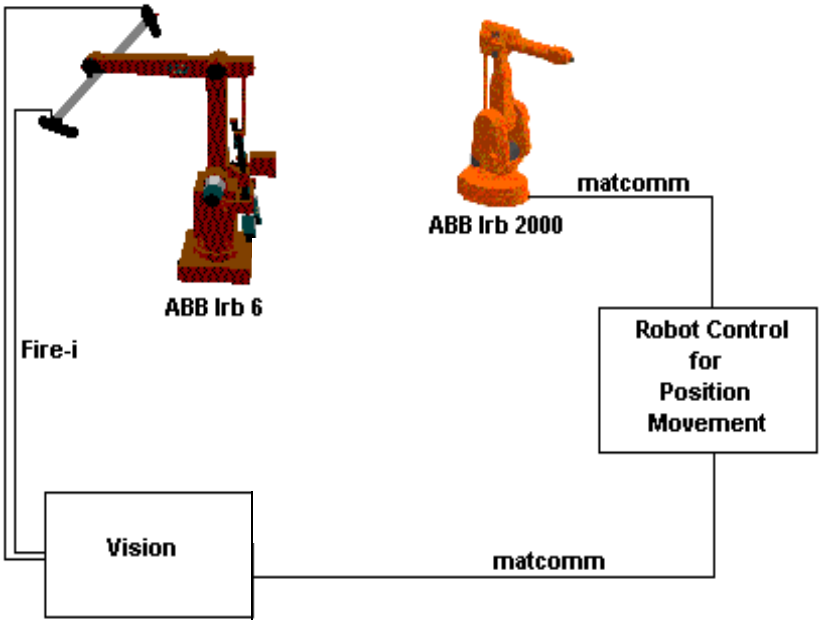


Figure 4.6 : Systems setup

The cameras mounted on the IRB–6 robotic manipulator provide us with images that are processed in the Windows station. Then data are sent to the SUN workstation which runs the control algorithm for the second robotic manipulator, the IRB–2000.

4.2 Vision

The vision software in this experiment is divided into three major parts. In the first part the reception of the images from the cameras and the time stamping of those images is done. The second part consists of the image processing algorithms and the third part involves the communication with the controller and the transmission of the data. The architecture of the vision part in this experiment can be seen in the following figure:

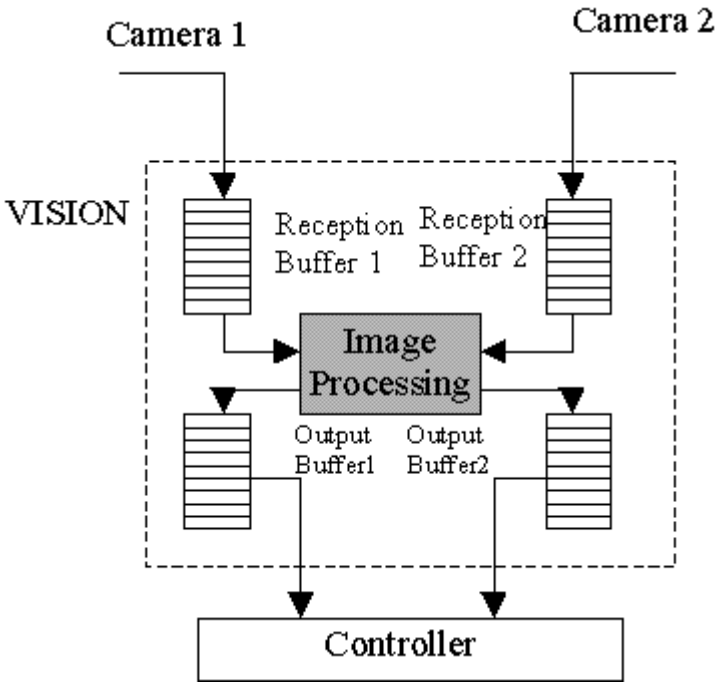


Figure 4.7 : Vision system architecture

The image processing is divided into two different parts of C++ code. The first part is about the locating and tracking of the robot TCP and the second part is about the locating and tracking of the ball. The images are processed based on the YUV422 specifications, where Y represents the luminance of the images and U and V the colour.

For the detection and tracking of the TCP the following steps are followed:

- A global threshold is applied to provide us with a binary image

- The segmentation of the image based on a 4–connected algorithm for finding regions follows. At this stage calculations of the center of the regions and of the size of the regions take place.

- After the region is selected from the end user by simply clicking on the image of our GUI, the tracking commences. The tracking of the TCP is performed by finding the closest region to the selected region of the previous image, with some constraints on size and displacement of the center points.

The second part of the processing applied to the received from the camera images involves the detection and tracking of the rolling ball. This is done by detecting motion in consecutive images. The steps followed in this procedure are:

- The command for the commence of motion detection is given by the end user by simply pressing the up button in the keyboard. A green light is highlighted on the bottom of the image processing window in the GUI.

- The luminance differences between pixels in consecutive images are calculated, through the formula:

$$dL(x,y,t)=L(x,y,t)-L(x,y,t-G) \qquad \textbf{(4.1)}$$

, where G is a constant set to 5 and represents the number of frames that we should look back in order to calculate this luminance differences.

- We binarize the above calculated differences using as threshold: $dL(x,y,t)>L_{thr}=30$.

- Using the binary luminances we segment the images and eventually we find all regions within a certain radius from the last image centerpoint. We now calculate the new centerpoint and the new size of the ball in pixels, using the found regions.

We should note that throughout the whole algorithm, a constant check is made below the green highlighted line if a new region of size above 20 pixels is detected. This is essential for the first detection of the ball.

Finally, the vision software "listens" to the SIMULINK model's requests and sends the appropriate data based on the code of the request. The requests are made by sending a request code to vision via matcomm. The codes for these requests and the relevant data which are sent can be seen in the following table:

Requests Code	1	2	3
Data Sent	Ping Exchange	Points Coordinates, Timestamps	Current Time

Table 4.1 : Data sent by vision and relevant request codes

The vision software described was created by Mathias Haage, a current PhD student in the Department of Computer Science, University of Lund.

4.3 Control Algorithms

The goal of the control algorithms used is to make the movement of the robot as smooth, fast and stable as possible. The trajectories are

calculated with respect to the initial and the desired position. The desired position is calculated from feedback provided by vision.

Although the calculations made from the data received from vision are using coordinates in image space, these coordinates are transformed in Cartesian space through image properties and with the help of our controller gain. Assuming that image coordinates for X and Y directions and the calculated depth Z are proportional to the X, Z and Y Cartesians coordinates respectively, we can include the constants involved within the controller gain.

The way our system was built made it obvious that we needed three different controllers, one for each Cartesian coordinate. The properties of the controllers should differ from coordinate to coordinate. This tactic was followed in order to lead us to better results, since this way we are able to tune our controllers for every dimension separately.

The controllers used were set to minimise the differences between the robot and the ball feature points for movement in the image plane and the difference between displacements, between corresponding feature points, for movement in depth.

The two kinds of controllers used in our experiments were the proportional (P) and the proportional – integrative (PI) controller. All theoretical facts presented below can be found in analytical forms in [3].

Proportional Controller (P)

A proportional controller in continuous time is described by the following equation:

$$u(t_k) = K * e(t_k) \qquad \textbf{(4.2)}$$

where K is the proportional gain of the controller.

In our case we are dealing with discrete time controllers which have the following form:

$$\begin{aligned} x_i(k\cdot h) &= K_x * d_x(k\cdot h) \\ y_i(k\cdot h) &= K_y * d_y(k\cdot h) \\ z_i(k\cdot h) &= K_z * d_z(k\cdot h) \end{aligned} \quad \textbf{(4.3)}$$

The proportional controller is a function of the error, which in our case is the difference between the current position of the ball and the current position of the robotic manipulator's end effector:

$$\begin{aligned} d_x(k\cdot h) &= x_{ball}(k\cdot h) - x_{TCP}(k\cdot h) \\ d_y(k\cdot h) &= y_{ball}(k\cdot h) - y_{TCP}(k\cdot h) \\ d_z(k\cdot h) &= z_{ball}(k\cdot h) - z_{TCP}(k\cdot h) \end{aligned} \quad \textbf{(4.4)}$$

The tuning of this controller was done by trial and error.

Proportional – Integrative Controller (PI)

The second controller used was the proportional – integrative controller. The integral term in this type of controller is used to eliminate the stationary error and has, in continuous time, the form:

$$I(t) = \frac{K}{T_i} \cdot \int_0^t e(s) \cdot ds \quad \textbf{(4.5)}$$

where T_i has time dimension and is called integral time.

It follows that

$$\frac{dI}{dt} = \frac{K}{T_i} \cdot e \quad \textbf{(4.6)}$$

and using the forward differences we get:

$$\frac{I(t_{k+1}) - I(t_k)}{h} = \frac{k\cdot h}{T_i} \cdot e(t_k) \quad \textbf{(4.7)}$$

The equations (4.6) lead us to the following recursive equation for the integral term:

$$I(t_{k+1}) = I(t_k) + \frac{K}{T_i} \cdot e(t_k) \tag{4.8}$$

The PI controller finally takes the form:

$$u(t_k) = K \cdot e(t_k) + \frac{K}{T_i \cdot s} \cdot e(t_k) \tag{4.9}$$

The actual PI controller we have used operates in discrete time. We shall use the "Backward Difference" approximation to go from Laplace transformation into Z transformation. This transformation is:

$$s \simeq \frac{z-1}{T \cdot z} \tag{4.10}$$

, where T is the sampling time.

The equation describing the discrete version of the PI controller is:

$$u(k \cdot h) = K \cdot \frac{(\frac{T}{T_i} + 1) \cdot z - 1}{z - 1} \cdot e(k \cdot h) \tag{4.11}$$

The tuning for this controller as well was done through trial and error.

4.4 Tracking The Moving Object

Initialisation

The experiment is initialised from the MATLAB part with the function **startexperP.m** for the P controller and **startexperPI.m** for the PI controller. These functions open the connection between the SIMULINK model, which simulates the controller, and the vision part running on the PC. Also the communication between the controller and the robot is initiated.

All variables are cleared and receive their initial values. Among them the value of the joint angles which represents the robots initial position is

initialised. A socket is opened in order for the model to be able to receive information from the robot, at any time, about its current position in joint space. The gains and integral times (for the PI case) are then set.

Finally the sampling frequency is chosen. The image processing algorithm runs on 30Hz and so we are capable of running the SIMULINK model on at least 30Hz. However we have decided to use a sampling frequency of 20Hz to eliminate some extra problems caused by noise in such a high (for this experiment) frequency. Now we are ready to run the SIMULINK model.

Initiation Of TCP And Ball Tracking–Initial Position

The experiment from the MATLAB point of view consists of three steps. In the first step the robot TCP goes to a prespecified initial position, in the second step the actual tracking of the ball with the use of the controllers and the predictors is done and in the third step the grasping of the ball is done, in a prespecified position along the Y axis with respect to the IRB–2000 base.

The initial position of the robotic manipulator was selected in such a way that we are able to see all the time through the cameras the image recognition scheme attached to the TCP, which is a black tape surrounded by white paper. The initial position of the robot should also be rather close to the cameras so that we can examine the operation of the controller through a lengthier movement of both the ball and the TCP.

After the robot was acquired the initial position we must indicate in the **Camera Server** software running on the PC which is the black surface, surrounded by a white surface, to be tracked. This is done for both camera images by directly clicking on the black object in the camera window. At that point the surface selected is highlighted in the relevant

image processing window and this way we are certain that we have indeed selected the robot TCP as the object to be tracked. This is actually the first step from a two step procedure to be followed on the software running on the Windows NT machine.

The second step regarding the vision part of this experiment is to turn on the motion detector algorithm running on the PC, after we have first left the ball free to perform its motion. This is done by pressing the up button on the PC keyboard. After the motion detection is on our full attention goes to the UNIX machine, which performs the simulation of the controllers.

Receiving Data – Predictions –Calculation Of Errors

The data received from vision are now processed from the MATLAB function **rcvpoints.m**. We now use the Kalman predictors in order to get from these data, estimates of the current position of the ball and then we calculate, using these predictions, the errors to be fed into the controllers.

For the error regarding the X direction of the images we can clearly observe from the setup of our experiment that it approximately corresponds to the X direction with respect to the IRB–2000 base axis system. So a simple way to calculate the error to be fed to the X coordinate controller, see also [7], is by using the following equation:

$$X_{error} = \frac{(X_{robot\ of\ camera1} - X_{ball\ of\ camera\ 1}) + (X_{robot\ of\ camera\ 2} - X_{ball\ of\ camera\ 2})}{2} \qquad \textbf{(4.12)}$$

For the error regarding the Z direction of the robot Cartesian coordinates we can use the Y measurements from the images, since clearly the Z axis of IRB–2000's base maps to the Y axis of the image plane. So the equation through which we calculate the errors in Z is:

$$Z_{error}=\frac{(Y_{robot\ of\ camera1}-Y_{ball\ of\ camera\ 1})+(Y_{robot\ of\ camera\ 2}-Y_{ball\ of\ camera\ 2})}{2} \qquad \textbf{(4.13)}$$

The depth with respect to the image plane corresponds to the Y axis of the IRB–2000 coordinates, but towards the negative side of the axis. The calculation of the depth error is done through the difference in displacements along the X image axis, according to known vision theorems. Since the mapping is between the Z axis in camera coordinates and the $-Y$ axis in robot coordinates the equation describing the difference in displacements receives a minus sign and so takes the final form of (4.12).

$$\begin{aligned} Y_{error} &= -((X_{robot\ of\ camera1}-X_{robot\ of\ camera\ 2})-(X_{ball\ of\ camera\ 1}-X_{ball\ of\ camera\ 2}))= \\ &= (X_{ball\ of\ camera1}-X_{ball\ of\ camera\ 2})-(X_{robot\ of\ camera\ 1}-X_{robot\ of\ camera\ 2}) \end{aligned} \qquad \textbf{(4.14)}$$

We must point out that the values used for the X and Y image coordinates of the ball are not the values received from vision, but the predicted values from the Kalman filters.

Trajectory Generation

After the errors have been calculated they are sent to the controllers and from there to the MATLAB functions which generates the trajectory to be followed. To generate trajectories we should first calculate the starting and ending positions for this sample time, in joint space. This means that through the known starting positions in Cartesian space we must calculate the end position in Cartesian space, using the outputs of the controllers as the errors in our IRB–2000 coordinate system. After acquiring these positions we should transform these values from Cartesian space into joint space.

The difficulty with the above task is that we want to keep the orientation of the gripper constant and also keep joint 4 locked in order to avoid problems caused by singularities. To keep orientation constant we calculate joints 5 and 6 with respect to joint 1, using the constants obtained through the least squares criterion, as described in chapter 3.5.

The procedure of the calculations is the following:

- Calculate joint angles through inverse kinematics, using tool length equal to $-100mm$, as explained in chapter 3.5

- Set joint 4 equal to -90^{o} and calculate joints 5 and 6 from equations (3.9)

- Calculate using the new joint angles the new position of the robot TCP (using tool length equal to 438mm) in Cartesian space

- Calculate the errors between the desired position in Cartesian space and the position calculated in the previous step . These errors are due to the slight movement of joints 5 and 6 which we have assumed to be still, in order to do our calculations for movement of the arm's end instead of the robot TCP

- Find what slight movement should the arm's end now make (in joint angles) in order to correct our final position, using once more the inverse kinematics script (with tool length equal to $-100mm$).

The last position acquired, in joint angles, is used along with the start current position of the robot to generate an appropriate trajectory.

We have tried using two ways in order to create this trajectory. The first involved using a motion with constant acceleration. This method was not used because it produced quite a lot of problems, such as high velocities and jumps in acceleration. These jumps are translated into shakings in the movement of the robotic manipulator.

The method that was implemented uses trajectories within one sample, which are formed using constant velocity. Through laws of physics regarding motion, from the initial position, the end position and the sample time we calculate the values we should have for the constant acceleration of each joint. This values are used in order to split the movement from our start position to the end positions into 4 smaller movements. By interpolation we calculate the joint angles and velocities for each one of these three intermediate and the final position. All these values are held in an array, the trajectories array, which is sent to the robot through matcomm.

Grasping The Ball

After running the experiment for several times and tuning the controllers in order to get a good behaviour in the movement of the robotic manipulator, we decided to try and grasp the ball. The difficulty in this task is that in order to grasp, the robots TCP which until this point followed the ball from a little bit higher so that we can see both of them in the cameras, should now come exactly in the same height as the ball and surround it. But this way we lose track of the ball in the cameras. This problem was solved with the help of our predictor. The predictor was used for the ball position but this time without using the Kalman gain

and adaptation mechanism, since we have no real data for the actual position of the ball in the image.

Another problem was the timing when the grasping should be done. We decided that we should do the grasping near the end of our motion and assuming that we are in position to grasp. So the grasping command is given when the robotic manipulator reaches a certain point in Y coordinates of the IRB–2000 base axis.

Ending The Experiment

Finally, we sent the robot to a home position and using the MATLAB function **stopexper.m** we close all connections to vision and to the robot. The socket which was opened for the robots position acquisition is closed and all variables are saved for evaluation.

Tuning The Controllers

As we mentioned in chapter 4.3, both the proportional and the proportional – integrative controller were tuned using the trial and error method. The values selected from this procedure for the gains of the proportional controller were :

$$K_x = 0.09, \qquad ,K_y = 0.9, \qquad K_z = 0.025 \qquad \textbf{(4.15)}$$

The outcome of this procedure for the proportional – integrative controllers case can be seen in the following table:

	X	**Y**	**Z**
Gain	0.3	0.83	0.04
Integral Time	30	30	30

Table 4.2 : Results of tuning for the PI controller

The actual outputs for the P and the PI controller can be seen in figure 4.8 and 4.9 respectively.

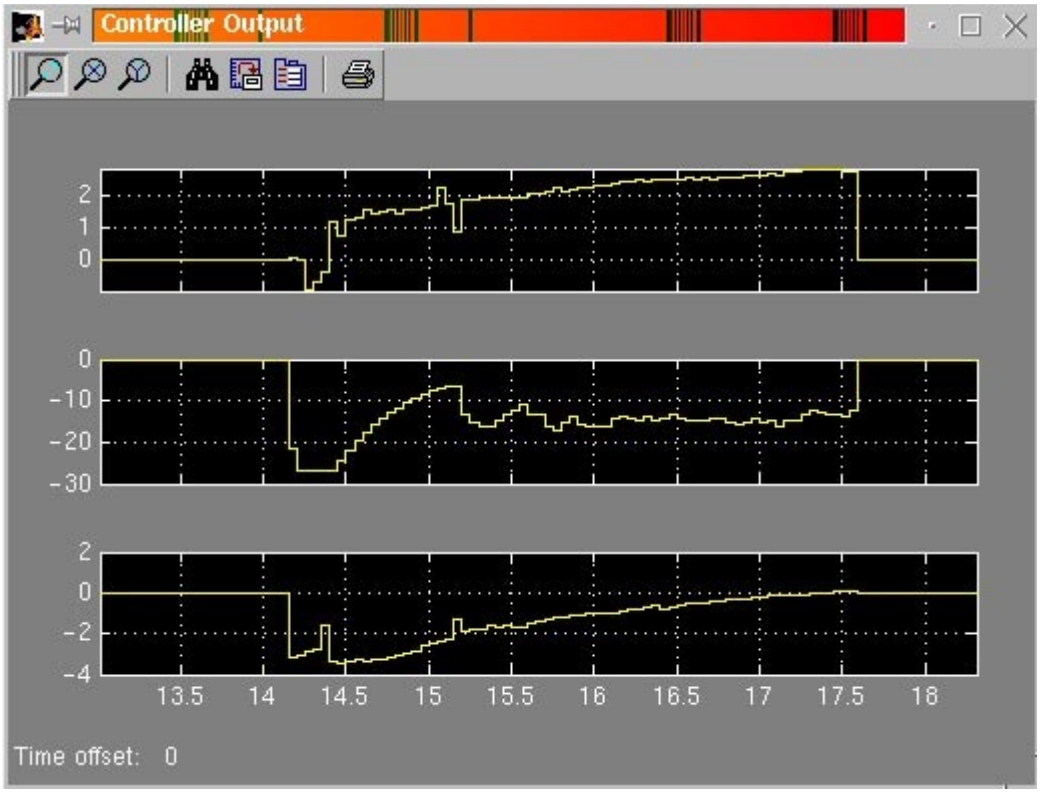


Figure 4.8 : Outputs in [mm] of the P controllers in X, Y and Z Cartesian coordinates

Figure 4.9 : Outputs in [mm] of the PI controllers in X, Y and Z Cartesian coordinates

Safety Precautions

The nature of the experiment led us to take some necessary safety precautions. More specifically bounds were set for the allowed movement of the robotic manipulator, using as references extreme positions in the Cartesian space.

In Y direction we could only allow the robot to be at a certain distance from the cameras, in order to avoid accident involving the stereo rig. On the other end the robot should not cross a specific point so we can avoid reaching the limits of joints 2 and 3. This meant that the tracking of the ball is not allowed through the whole course of the ball movement, but only until few centimetres prior to it's end.

In Z direction we should find a way to avoid collision with the bar, on which the ball rolls. In order to do that we measured two points on the bar and another point in Cartesian space so we can identify a plane below which positioning would not be allowed. The measurements showed that the two points on the bar had a difference of around 5mm and so instead of using the plane we originally planned to, we used a unique Z bound for the robot. The robot TCP is not allowed to position itself below 1070mm from the ground level.

Finally the last safety measure we decided to take was the saturation of the error input to the controller for the Y (depth) direction. This was done in order to prevent rapid movements due to wrong measurements. The wrong measurements could sometimes just be noise but we have had certain occasions where this wasn't the case. We have noticed that depending on the light conditions it was possible to lose track of either the ball or the TCP and start tracking some other object in the visible space. In some other cases the marker tracking the ball would jump to track the TCP. We should keep in mind that the tracking of the ball is done through motion and so an abrupt movement of the TCP right above

the ball could trick the image processing software, when lighting conditions are bad.

All the above cases of disturbance in the image processing could lead us to some accident since wrong coordinates could mean large errors and eventually large movements. Saturating the movement along Y seemed enough to prevent us from such an accident.

Smoothing The Movement

When we first tried to run the experiment we noticed than in some cases the robot TCP managed to get at a desired position above the ball or in some cases, depending on the controller gain and the noise in our data, a little bit ahead. The behaviour of the system at that point was to stop the movement of the gripper or even reverse it. In the same time the ball kept rolling and so after a few samples the ball was way ahead of the gripper. This caused a rapid acceleration and possibly a reverse movement, combined with a jump in velocity.

The output of this procedure could be seen in the movement of the TCP as shakings and stoppings of movement. Since we desired a smooth movement with no stoppings, we decided to act against this phenomenon. The way we compensated for this situation was by preventing the error on Y, sent to the controller, to be less than the 75% of its previous value. We should keep in mind that when running in 20Hz if the TCP is far ahead from the ball then we will still keep moving ahead with smaller velocities only for a couple more samples, which means around 100–150 msecs. The ball will be allowed to catch up rather fast and then we can jump back to our usual calculations of the error. This way we don't allow big jumps in velocities and our movement is smoother.

When running the experiment we checked how many times this procedure was followed. It turns out that we use this compensation only

for about 10 samples from which almost half are during the first second of the movement, where a rapid movement is allowed. This means that the saturation of error measurements does not take over from the control algorithm or even the calculation of errors algorithm that we have implemented.

5 Results

5.1 Results For The Kalman Predictors

The predictions provided by the Kalman filters we have implemented have quite satisfactory results. The predictors seem to have rather small errors and achieve the goals for which they were inserted in our system.

More specifically compensation for the time delays was achieved, even though for such small delays. The time delays in our system are now around 40 msecs and so this compensation can not be seen in the movement in the actual experiment. However we can see improvement in the measurements we receive from running the experiment. In figure 5.1 we can see how the predictions used are much closer to the real output than the formerly used timed delayed information. The predictions and the real data correspond to the X and Y dimensions in the image plane.

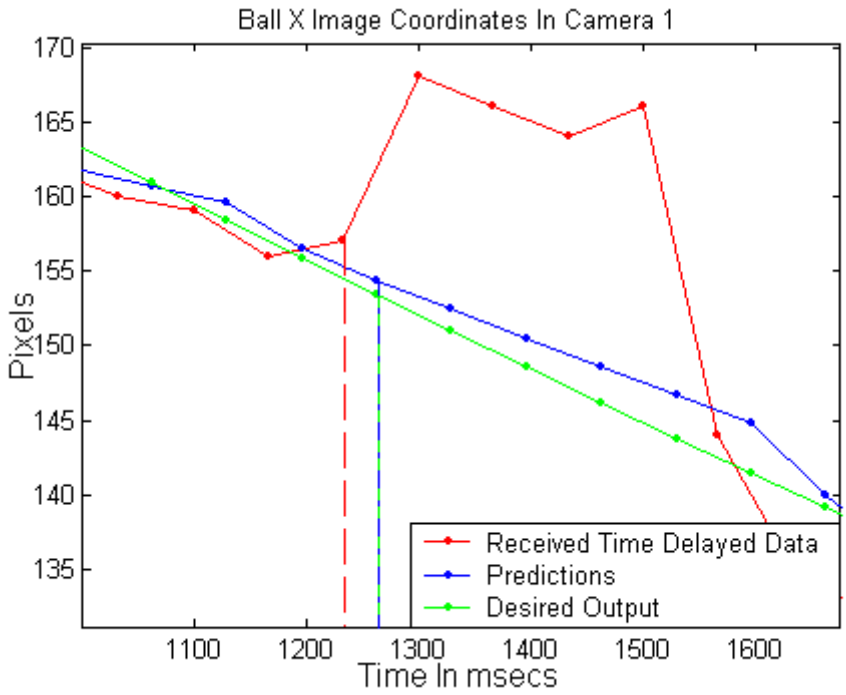


Figure 5.1 : Compensation for time delays

The second goal for which the Kalman predictors were implemented was reached with much more success. Filtering of noise was achieved in such a degree that we could notice the difference in the motion of the robotic manipulator by just looking. The graphs showing the improvement in the performance of the system are presented below.

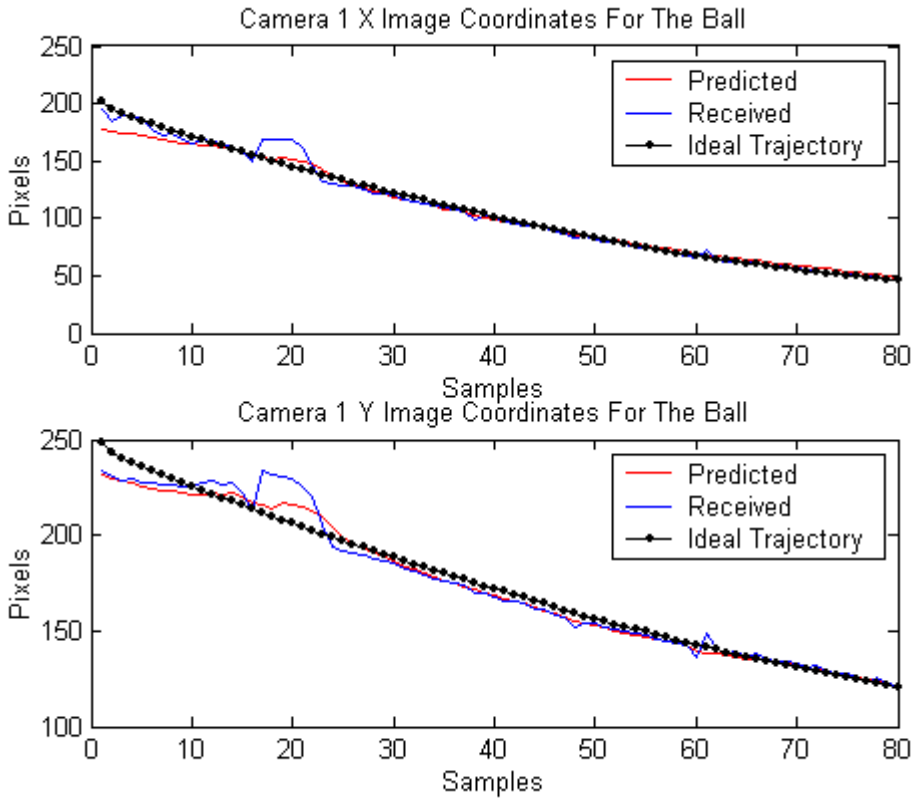


Figure 5.2 : Predicted trajectory, ideal trajectory and received data for camera 1

We can see how smoother and closer to the ideal trajectory the predicted movement of the ball in the image space is, contrary to the movement indicated by the noisy vision measurements.

The estimators used for the TCP positioning in the image space were satisfactory and can be seen in the figures 5.3 and 5.4.

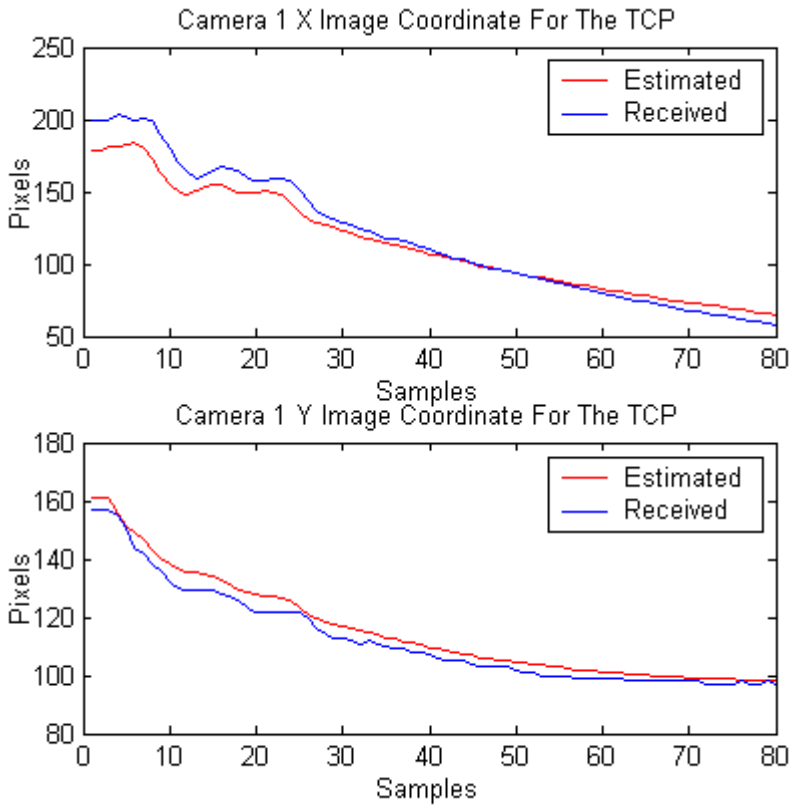


Figure 5.3 : Estimation for the TCP in camera 1

We should point out that these estimations are used only when we lose track of the robot TCP and for their calculation camera calibration, see [8], is required. In our experiments, even if we have implemented this feature for safety reasons, we don't really use it in order to examine a complete system which uses uncalibrated cameras. The estimations shown in the figures have slight errors, since we have had some slight movements in the cameras and so our calibration has limited accuracy.

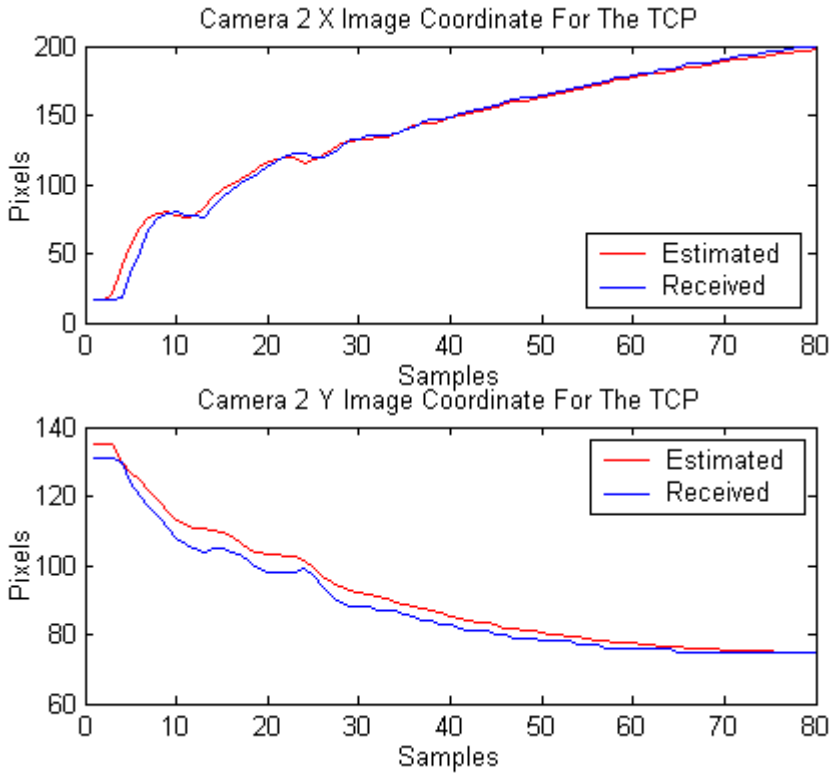


Figure 5.4 : Estimation for the TCP in camera 2

5.2 Performance

The experiment is currently working successfully in 20Hz. We have however tried running the control algorithm at 30Hz. The controller functions normally but the noise injected in our system from vision cannot be easily filtered and therefore we are facing larger errors when tracking the ball, as well as rapid accelerations and stoppings of the robot TCP.

When running in 20Hz the results are much better. The noise is filtered satisfactorily by our predictors and the movement of the TCP is rather smooth.

When coming to the grasping of the ball we have noticed that although it is done in a fixed position with respect to the Y coordinate it seems to work much more frequently than we initially expected. The initial velocity of the ball when entering the image does not affect the positioning of the robots TCP above the ball at any time, so the grasping is most of the times successful.

The cameras work at 30Hz and are software triggered. They are not synchronised and a time difference of 17msecs between them has been recorded.

The image processing software consumes around 25msecs and the communication times are less than 1msec.

5.3 Robustness

The system has been proved to be robust when the light conditions are stable. Since this is not the case in most situations we noticed large differences in the performance of the system when working during day and night.

When working during the day the outside lighting conditions change constantly and affect the lighting conditions inside the lab. In this cases the vision system seemed to be experiencing great difficulties to keep track of both the TCP and the ball. During the night when outside lighting conditions did not effect the inside lighting conditions the vision part seemed extremely robust and so was the complete system.

It is obvious that the robustness of the system is in direct relation to the performance and robustness of the vision algorithms.

5.4 Errors

The errors observed in this experiment had mainly to do with our controlling in Y directions, which is the depth of the movement. The errors on X and Z directions were negligible after a good tuning of the X and Z direction controllers was achieved.

The error in depth is in the range of a few centimetres and is actually noticed when the actual grasping is tried out. The main reasons for this error are :

- Noise introduced by vision
- Errors in our predictions
- Errors in the calculation of movements to be made, due to the fact that we are using non–calibrated cameras.

6 Discussion And Future Work

6.1 Discussion

It was observed that the original idea of compensating for the time delays through timestamping and use of Kalman predictors, was rather succesful in compensating for several other errors occurring in our system, such as temporary loss of tracking and noise from the image processing.

However the Kalman predictor from its nature is a linear predictor and we should expect much better results if a non–linear observer was used instead. The current movement of the ball is a slow linear movement. If instead we would like to experiment with a faster and nonlinear movement, the use of a non–linear observer seems absolutely necessary.

The time delays occurring in the system in combination with the 20Hz sampling frequency used made the use of one step ahead predictors possible for this experiment. However if we would like to increase the sampling frequency, one step ahead predictors would not be sufficient. The time delays, at some point, will exceed one or even more samples of delay and then the used of n–step ahead predictors will be necessary.

In the trajectories of the ball, as received from the vision algorithms, we have experienced some strange jumps in specific areas in the image space, see figures 2.5, 3.2 and 5.2. We believe that these jumps are due to shadow of the TCP over the ball. In fact the common room florescent lamps in the lab are situated in such a way that our current setup could not avoid this phenomenon. We could also argue that other lighting phenomena like reflections and highlights are responsible for the high frequency noise injected in our measurements. We should expect in the future an updated image

processing software, able to compensate for lighting conditions. This should be enough to suppress noise injected in the measurements and reduce by far the errors we have observed in our experiment.

The image processing software also introduces some errors in our calculations of the relative position of the TCP with respect to the ball. These error calculations which are fed to the controllers should be more precise and this could be achieved by using calibrated cameras and applying 3–D reconstruction techniques for the feature points tracked.

The win32 platform used for the realisation of the image processing software is not a hard real–time system. Restricting high priority threads to time stamping only proved however sufficient for our experiments needs. Thus it is possible to argue that time stamping has decreased dependency on hard real–time platforms.

Finally we should note that the two controllers used, even though they are simple and common controllers, seemed to operate satisfactorily and no use of more complex control schemes was required. However, in faster and nonlinear movements, where the time response and overshoot properties of the controller are more important, a more complex controller scheme should be considered in order to increase performance.

6.2 Future Work

It is true that there is a lot of space for improvement on the experiment performed. First of all we should mention the possibility of using calibrated cameras for the experiment. A good calibration would allow exact knowledge of the robots TCP and of the ball in Cartesian space, from the corresponding image coordinates, through 3D reconstruction techniques. This should improve the performance of the experiment since the data used for controlling of the robots TCP would be far more accurate than they are now.

Future work should also be done in the vision algorithms. The robustness of the experiment could be improved if the robustness of the vision part is also improved. An effective way to filter out noise should be implemented in order to always have accurate and valid data for the position of both the ball and the TCP.

Much improvement is also possible to be made on the way the grasping is performed. More specifically we believe that a fixed position for grasping is not the ideal solution, but an approach where the gripper grasps the ball when the data from vision indicate that we are in position to grasp, is more suitable.

In the current grasping procedure the gripper is horizontal with respect to the ground. However, the human motion for a similar movement would be having our hand in a certain angle with respect to the ground. This should give thoughts in how the actual grasping of the ball should be performed by the robotic manipulator.

Finally the predictor schemes we have implemented could have space for improvement. The Kalman predictors with no input and constant Kalman gain which are currently used, could be replaced by more effective Kalman filters with variable gains, if the variance of the noise in the system is measured.

Although we have pointed out some features where improvement is necessary, we must emphasise on the fact that the current system is a very good base for future experiments that use visual feedback to control a robotic manipulator.

7 Conclusions

In this thesis project we have presented a practical and simple way to reduce the effects that noise and time delays have on a system. The Kalman predictor has proved to be a powerful tool. It has helped us create a system able to compensate adequately for the above two basic problems.

Other problems we have encountered throughout this project, mainly concerning the robotic manipulator's behaviour, were faced with much success.

The vision software used was very fast and this untied our hands in terms of sampling times and frequencies possible for use.

We have managed to built a system which is rather robust and actually uses uncalibrated cameras for visual feedback. However we should expect much better results when using calibrated cameras, since in that case precision will be increased and errors minimised.

The controllers behaviour was as expected and no problems were faced regarding their tuning. The tuning procedure was simple but most effective.

The results were more than satisfactory as was the robustness of the system under constant lighting conditions.

Finally, I should point out that this thesis assignment has made it possible for me to learn about new fields of research, previously unknown to me, like robotics and vision. The experience was more than fulfilling and the knowledge obtained extremely useful. The fact that for the first time I actually worked in a lab, facing a real process, as was the robot tracking of the ball, and as part of a team, has provided me with invaluable experiences.

8 References

BIBLIOGRAPHY

[1] John J. Graig. *'Introduction to Robotics Mechanics & Control'*, Addison–Wesley, 1986

[2] Emanuele Trucco and Alessandro Verri. *'Introductory Techniques for 3–D Computer Vision'*, Prentice Hall, 1998

[3] Karl Johan Äström and Björn Wittenmark. *'Computer Controlled Systems – Theory and Design'*, Prentice Hall, Second Edition, 1990

[4] Rolf Johansson. *'System Modeling And Identification'*, Prentice Hall, 1993

[5] Zoe Doulgeri. *'Textbook For The Coure Robotics'*, Department Of Electrical And Electronic Engineering, Aristotle University of Thessaloniki, 2001

[6] Johan Nilsson. *'Real –Time Control Systems With Delays'*, Doctoral Dissertation, Department of Automatic Control, University of Lund, 1998

[7] Luis Manuel Conde Bento and Duarte Miguel Horta Mendonca, *'Computer Vision and Kinematic Sensing in Robotics'*, Masters Thesis, Department of Automatic Control, University of Lund, 2001

[8] Tomas Olsson, *'Vision Guided Force Control In Robotics'*, Masters Thesis, Department of Automatic Control, University of Lund, 2000

[9] Johan Bengtsson and Anders Ahlstrand. *'A Robot Playing Scrabble Using Visual Feedback'*, Master Thesis, Department of Automatic Control, University of Lund, 1999

[10] Anders Robertsson. *'On Observer – Based Control of Nonlinear Systems'*, Doctoral Dissertation, Department of Automatic Control, University of Lund, 1999

[11] Klas Nilsson. *'Industrial Robot Programming'*, Doctoral Dissertation, Department of Automatic Control, University of Lund, 1996

[12] Lennart Ljung. *'System Identification Theory For The User'*, Prentice–Hall Information And System Sciences Series, 1987

[13] Rolf Johansson, Anders Robertsson, Klas Nilsson and Michael Verhagen. *'State–space Identification of Robot Manipulator Dynamics'*, Mechatronics ,10 : 403–418, 2000

[14] B. Haverkamp and M. Verhagen. *'SMI Toolbox, State Space Model Identification Software for Multivariable Dynamical Systems'*, Version 1, Delft University Of Technology, 1997

RELEVANT WEBSITES

(1) ***http://www.abb.com*** , Asea Brown Boveri (ABB) web site

(2) ***http://www.control.lth.se/~robot*** , Robotics Laboratory, Department Of Automatic Control, University Of Lund

(3) ***http://www.lucas.lth.se*** , LUCAS Center For Applied Software Research, Department of Computer Science, Department Of Communication Systems and Department Of Automatic Control, University Of Lund

(4) ***http://skiron.control.ee.auth.gr*** , Automation And Robotics Lab, Division Of Electronics And Computers, Department Of Electrical And Electronic Engineering, Aristotle University Of Thessaloniki

(5) ***http://www.ieee.org*** , The Institute Of Electrical And Electronic Engineers

(6) ***http://www.iee.org.uk***, The Institue Of Electrical Engineers, London UK

(7) ***http://www.ifr.org*** , Internation Federation Of Robotics

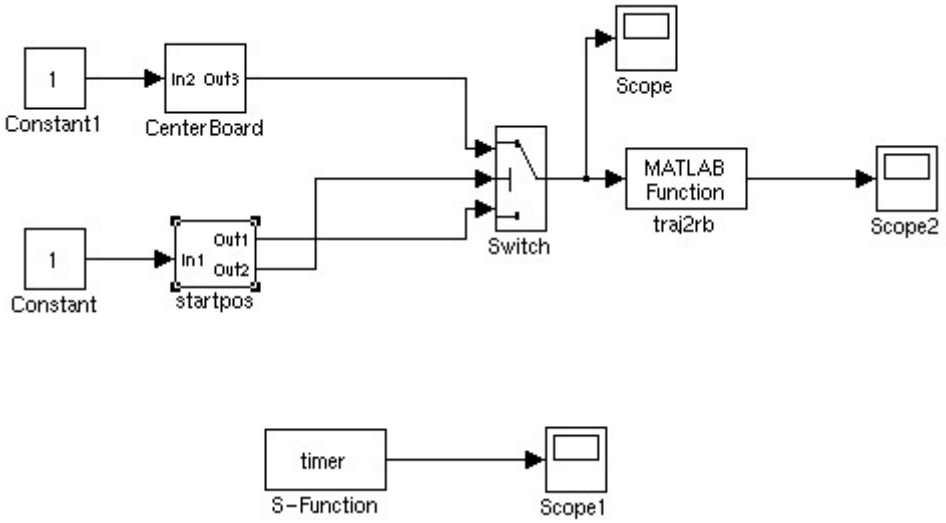
(8) ***http://www.iis.ee.ic.ac.uk*** , Intelligent And Interactive Systems Group, Department Of Electrical And Elecetronic Engineering, Imperial College Of Medicine And Science

(9) ***http://www.ee.ic.ac.uk/CAP*** , The Control And Power Group, Department Of Electrical And Electronic Engineering, Imperial College Of Medicine And Science

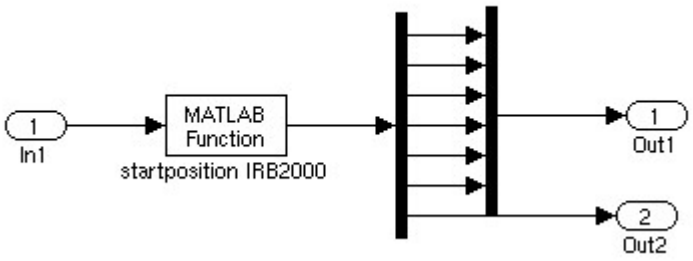
(10) ***http://www.cs.lth.se/~ai*** , Artificial Intelligence At Computer Science (AI@CS), Department Of Computer Science, University Of Lund

9 APPENDIX

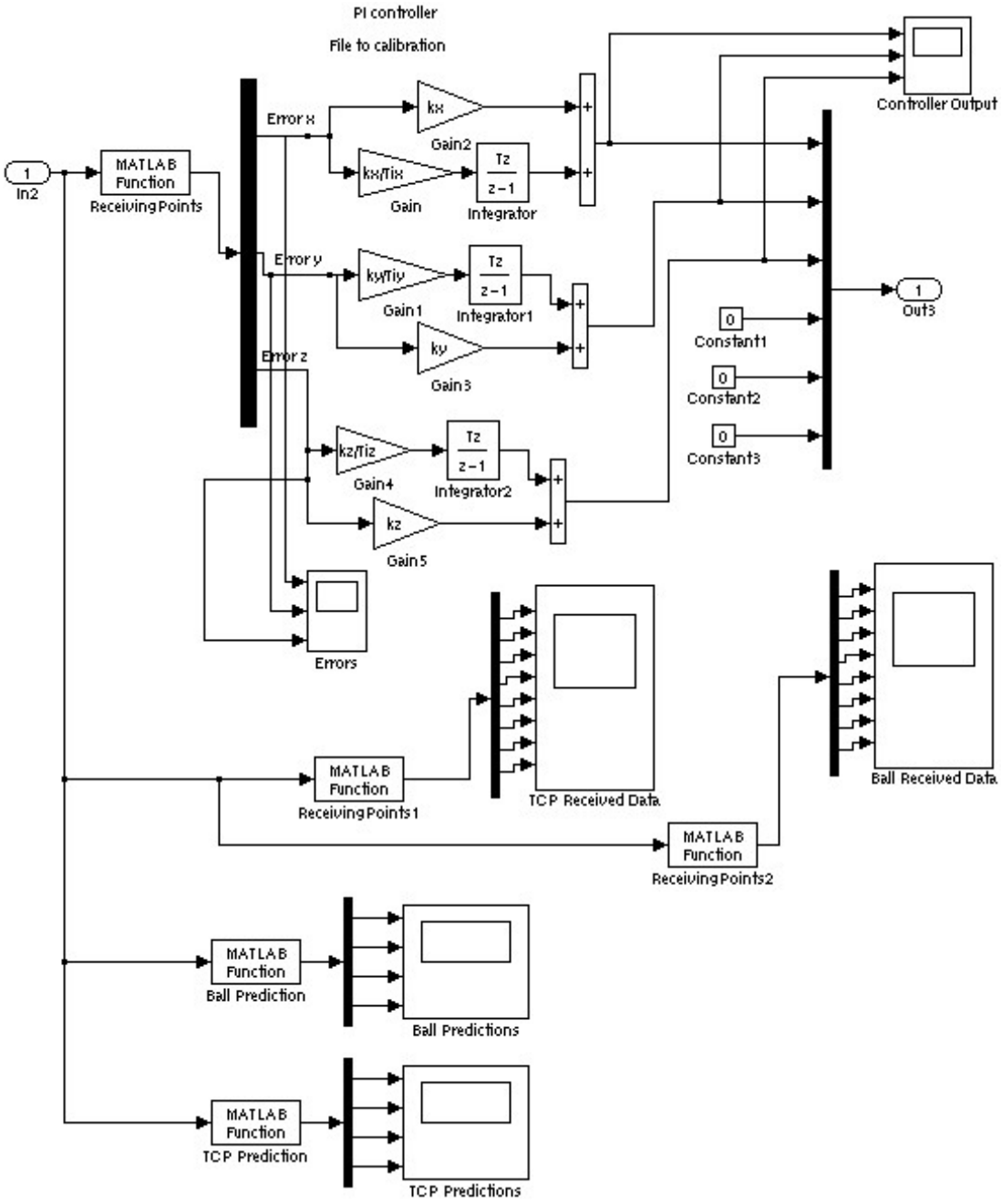
SIMULINK MODELS



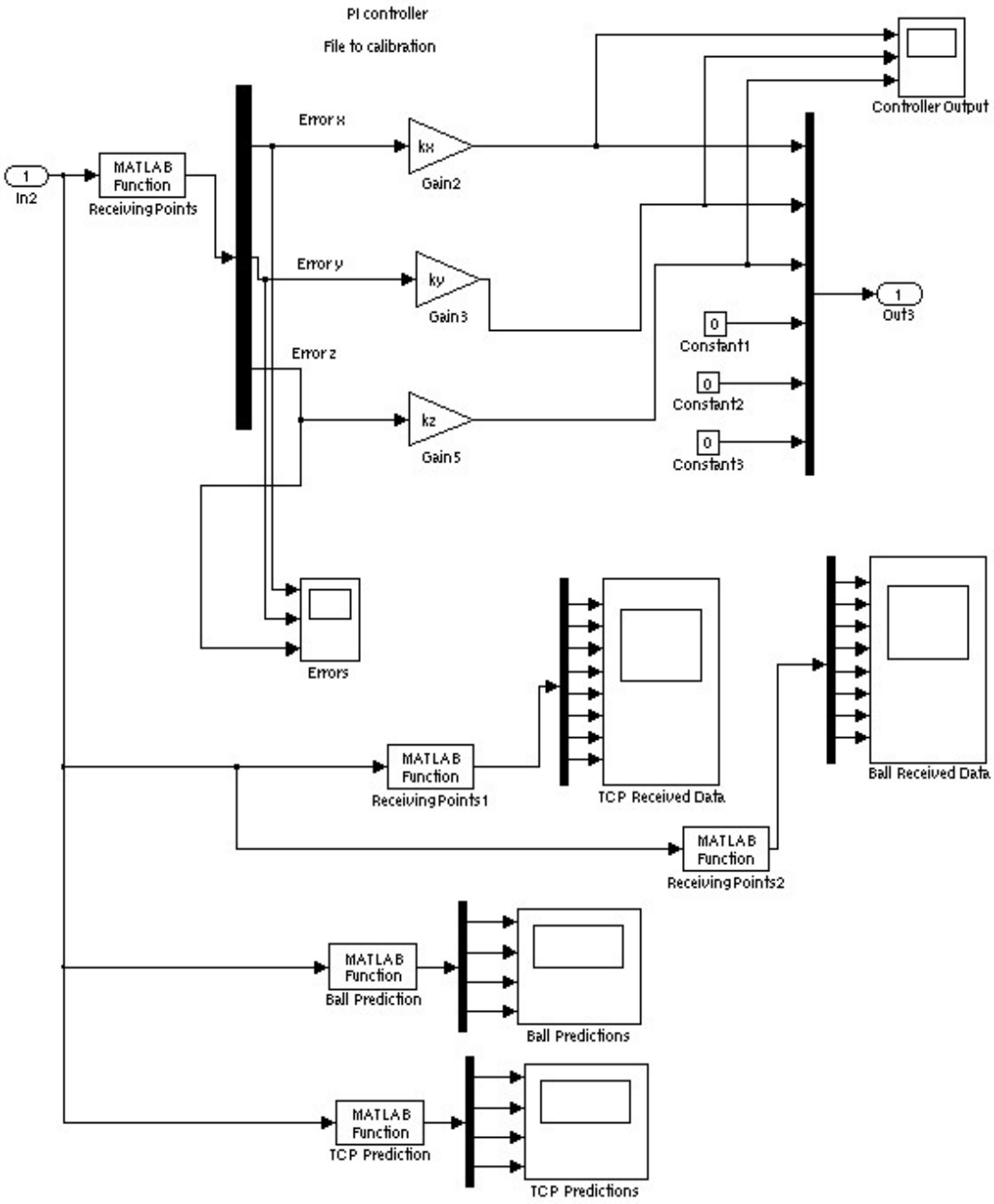
Basic SIMULINK model for both controllers



Block **startpos** for acquisition of start position, for both controllers



PI controller centerboard



P controller centerboard