ISSN 0280-5316
ISRN LUTFD2/TFRT--5670--SE

Computer Vision and Kinematic Sensing in Robotics

Luis Manuel Conde Bento
Duarte Miguel Horta Mendonca

Department of Automatic Control
Lund Institute of Technology
June 2001

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

Document name
MASTER THESIS

Date of issue
June 2001

Document Number
ISRN LUTFD2/TFRT—5670--SE

Author(s)
Luis Manuel Conde Bento
Duarte Miguel Horta Mendonca

Supervisor
Mattias Haage, LTH
Johan Bengtsson, LTH
Rolf Johansson, LTH

Sponsoring organization

Title and subtitle
Computer Vision and Kinematic Sensing in Robotics (Visuell återkoppling i robotsystem)

Abstract
To use vision in a robotic setting it is important to achieve realtime performance. Real-time vision may be used to directly steer robots using for instance visual servoing techniques. In this thesis, an experimental vision setup using a stereo rig mounted on an industrial robot (ABB Irb-6) was built from ground up and then used to perform two experiments; visual servoing and collection of data for calibration of stereo rig and positioning of second robot (ABB Irb-2000) using visual feedback. The system is currently capable of achieving a ~15Hz visual feedback rate which could be easily extended into the 20Hz domain.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
74

Recipient's notes

Security classification

ABSTRACT

To use vision in a robotic setting it is important to achieve realtime performance. Real-time vision may be used to directly steer robots using for instance visual servoing techniques. In this thesis, an experimental vision setup using a stereo rig mounted on an industrial robot (ABB Irb-6) was built from ground up and then used to perform two experiments; visual servoing and collection of data for calibration of stereo rig and positioning of second robot (ABB Irb-2000) using visual feedback. The system is currently capable of achieving a ~15Hz visual feedback rate which could be easily extended into the 20Hz domain.

CONTENTS

Abstract **1**
Contents **2**
1 Introduction **4**
1.1 Robotics 4
1.2 Vision 4
1.3 Short explanation of the investigation 5
1.4 Thesis Outline 6
2 Vision **7**
2.1 Introduction 7
2.2 Camera Model 7
2.2.1 Pinhole Camera Model 7
2.2.2 Light and Lens 8
2.3 Projective Geometry 10
2.4 Feature Extraction 12
2.5 Correspondence Problem 14
2.6 3D Reconstruction 15
2.7 Real-Time Problem 16
3 Kinematic Estimation and Control using Visual Feedback **17**
3.1 Introduction 17
3.2 Experimental Setup 17
3.3 Experiments 21
3.3.1 Calibration Movement 21
3.3.2 Positioning Movement 22
3.3.3 Virtual Robot (Java Robot) 22
3.4 Real Time Stereo Vision Pipeline 23
3.4.1 Image Processing 23
3.4.2 Closest Neighbor Vector Field 25
3.4.3 Line Segments Extraction 27
3.4.4 Lines Extraction and Center Cross Estimation 29
3.4.5 3D Lattice and Feature Points Correspondence 31
3.4.6 Interpolation and Extrapolation 34
3.4.7 3D Kinematic Estimation using Lattice 35
3.5 Concerns using Virtual Robot 37
3.6 3D Kinematic Estimation 38
3.7 Robot Control using Visual Feedback 41
3.7.1 Control of Robot 43
3.7.2 Control of Virtual Robot 45
4 Prototype **47**
4.1 System Architecture with Dataflow Diagram 47
4.2 External API (robot, matcomm, camera) 48
4.3 Kinematics Control (Simulink/Matlab) 53
4.3.1 Calibration Movement 53
4.3.2 Position Movement 57
4.4 Vision Processing (Visual C++) 59
5 Results and Conclusion **66**
5.1 Performance Estimation and Code Profiling 66

5.2 Robustness 66
5.3 Errors 67
5.4 Conclusion 67
5.5 Future Research 67
References 69
A-Simulink Models 70

1 INTRODUCTION

1.1 Robotics

Today automation is used frequently in industry for different applications. The robot industry grew very fast primarily due to large investments done by the automotive industry. Modern industrial robots have increased in capability and performance through controller and language development, improved mechanisms, sensing, and drive systems.
The present level of robotics technology, in such areas as machine vision, tactile sensing and artificial intelligence is still primitive compared to the adaptability and dexterity of humans. This thesis investigates the vision applied in a robotics setting.

Robots with restricted sensor feedback are limited in the kinds of behavior they can exhibit. Yet, this is how robots currently used in industrial applications perform their tasks.
The needs for motion descriptions and operator interactions clearly show that robot control requires its own control techniques.

In this investigation the robot control is done with visual feedback that give the relative position compared to a reference position

Traditional robot control uses world and joint coordinate system representations to describe goals, plan and execute moves. In a static industrial environment, where the environment, the robot model and the task are known, this works well. Some models are used to transform the task into a sequence of robot motions. However, most natural settings are not structured or easy to model analytically.

1.2 Vision

In robotics there are many tasks in which inspection, manipulation, or measure of three-dimensional objects are involved. To be able to use feedback control in these tasks it is necessary to use techniques that return three-dimensional information about the objects. For this purpose there are several types of sensors. If we look in the robotics domain these types of sensors are denominated as external. External sensors can be classified from the way that they acquire the measure in two classes; the ones that require direct physical contact, such as contact switch, force on tact sensors, and the ones that don't require direct contact, such as ultra-sound sensors, infra-red sensors, laser and video cameras. Computer vision systems enables the use of robots in non-structured environments, i.e. the work area isn't limited to a special room or environment.
Computer vision is commonly denominated as "Image understanding" but understanding is far from being easy. The processing of visual

information presents problems that are hard to manage. For instance many surfaces composed by different materials and with different geometric properties have the same image making it hard for the vision system to differ between them. Therefore it is difficult to recover a good interpretation of images using surface models. These problems are especially important for three-dimensional objects being represented in two-dimensional images.

Human vision is a controlled hallucination. This means that what we infer from images is more than we can explain using physics of light or image formation models, as illustrated in Figure 1.1.
An example of how human's process images is called "pictorial depth cues". A "cue" can be the most familiar size, interposing or occlusion (both are represented in Figure 1.1), shades or shaded areas, size of the object related to horizon line, motion and motion parallax and binocular perception (i.e. stereoscopy).

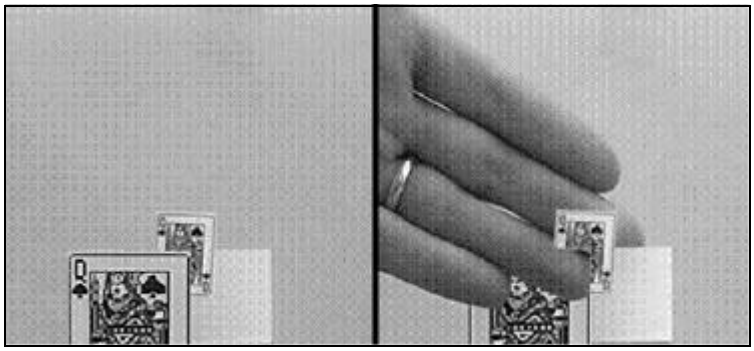


Figure 1.1

Stereoscopy means the study of corresponding images to recreate three-dimensional coordinates. It is the disparity between the two retinal images that enables stereoscopic perception of depth. It is based on projective geometry.

1.3 Short explanation of the investigation

The goal of this thesis is to implement a self-calibrating system which after calibrated is able to track moving objects and accurately determine their position in 3D coordinates. The technique used provides a control system based on visual information which doesn't depend on the cameras parameters, this is a major gain since achieving a good calibration accuracy on the cameras parameters is very hard to get. Since the cameras parameters aren't important to this kind of implementation, it is avoided one of bottleneck of Computer Vision.
To implement the system it is used a robot and two cameras, the robot is used as positioning system and the two cameras compose a stereovision system mounted in the end-effector of the robot. The

system performs a calibrating routine by a well-known trajectory, the information acquired is used as pattern in future real time tracking and positioning.
In this investigation it is presented a method to obtain some three-dimension information of a specific object in the surrounding area of a robot, by the use of images. The estimation of that information basically consists of the determination of the distance to the object using disparity cues. It was developed a system to determine the relative position between objects, that use a stereo rig composed by two cameras with no geometry pre-defined for their referential, mounted on the end-effector of a IRB-6 robot. The IRB-6 position the stereo rig right above a plate, with reflective surface (calibration plate) and an array of holes where feature points are extracted. The stereo rig is calibrated by moving the IRB-6 end-effector in a linear movement sampling image feature points from a calibration plate.

The lattice is made from several layers corresponding each layer to one z relative distance to the calibration board. A layer is composed by the displacement of each feature point visualised by both cameras, so in this way using interpolation, it is possible to build an x and y characteristic displacement for the image area covered by both cameras. This is done for a set of z relative distances to the calibration board. After all the layers been saved it is possible to estimate the relative distance of any point in image area covered by both cameras, using either interpolation for points with a depth between the layers, for depths higher or smaller than the set of relative distances saved in the pattern/database the depth is extrapolated.

Although, the method is not wide spread it is foreseen as a method for the future with lot of potential.

1.4 Thesis Outline

The thesis is organized as follows:

The second chapter introduces problems in vision, particularly in stereo vision.

The third chapter gives an overview of the system architecture and its modules. It describes experiments, control problems and their causes. Also describes the algorithms implemented at the vision level.

Chapter four describes the software prototype.

In Chapter five presents results and conclusions.

2 VISION

2.1 Introduction

In this chapter gives a short theoretical orientation of computer vision problems. It shows image acquisition and processing methods necessary to produce data estimation of 3D kinematics data.

2.2 Camera Model

In a work that involves image analysis to determine three-dimensional structure, it is necessary to establish a model that describes projection to 2D image planes.
There are two considerations to take into account: the projective geometry that determines where a point of the scene would be projected onto the image plane and the physics of light that determines how brightness changes as a function of scene illumination and surfaces properties.

2.2.1 Pinhole Camera Model

A camera model for the generation of an image is the pinhole camera. It is a box that has an infinitesimal small hole through which light enters and forms an inverted image on the camera back plane. To simplify things, we usually model a pinhole camera by placing the image plane between the focal point of the camera and the object, so that the image is not inverted. This mapping of three dimensions onto two is called perspective projection. Several alternative projection models exist, like paraperspective or orthographic projection.
Projective geometry is fundamental to the understanding of image analysis.

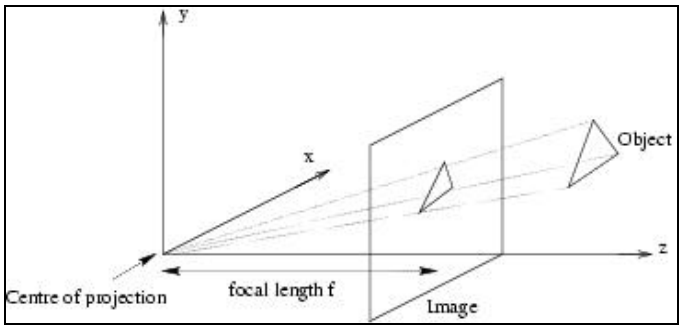


Figure 2.1

In Figure 2.1 3D feature points are projected onto an image plane with perspective rays originating at the center of projection (COP). The origin of the coordinate system is traditionally taken to be the COP and the focal length. Where f is the distance from the COP to the

image plane along the principal axis (or optical axis). The optical axis is traditionally aligned with the $\vec{z}$ axis.

Geometry shows that if we denote the distance of the image plane to the centre of projection by f, then the image coordinates (x_i,y_i) are related to the object coordinates (x_o,y_o,z_o) by

$$x_i = \frac{f}{z_0} x_0 \text{ and } y_i = \frac{f}{z_0} y_0$$

These equations can be easily expressed by introducing homogeneous transformations, which is a matter of placing euclidean geometry into the projective geometry space. In homogeneous coordinates, the perspective projection onto the plane is given by:

$$w_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

Under perspective projection, parallel lines (train tracks) converge to a point on the horizon.

2.2.2 Light and Lens

Real world light is affected by error introducing effects such as lens effect, sampling, quantification, etc before reaching the final state as a digital image.

The camera lens mimics the pinhole camera, without using small apertures. Ideally a lens receives all the light radiated by object-point and focus all the radiation in only one image-point.
The limitation of lenses is that they can only bring into focus those objects that lie on a particular plane parallel to the image plane. Assuming the lens is relatively thin and that its optical axis is perpendicular to the image plane, it operates according to the following law:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}$$

Where u is the distance of an object point from the plane of the lens, v is the distance of the focused image from this plane, and f is the focal length of the lens Figure 2.2.

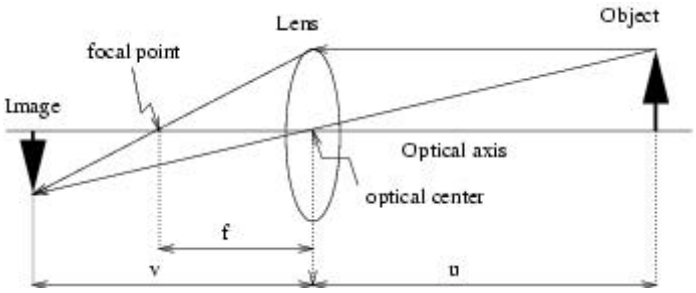


Figure 2.2

Points that don't belong to the focal plane have their representation as circles instead of points; the circle is called "blur circle".

The diameter of the circle is proportional to the aperture diameter. If the aperture diameter decreases the range of world approximately focused increases but there is a reduction on light intensity. Longer exposure times can solve the reduction of light intensity but is accompanied by a loss of time resolution; the trade off is between loss of spatial resolution or time resolution.

The lenses also impose several aberrations:

- Chromatic aberration in a lens will not focus different colors in exactly the same place because the focal length depends on refraction. The index of refraction for blue light (short wavelengths) is larger than that of red light (long wavelengths) exhibits.
- Spherical aberration exists for lenses made out of spherical surfaces. Rays which are parallel to the optic axis but at different distances fail to converge at the same point.
- Coma aberration causes rays from an off-axis point of light in the object plane to create a trailing "comet-like" blur directed away from the optic axis.
- Oblique astigmatism is a result of different lens curvatures in different planes.
- Curvature of field causes a planar object to project to a curved (nonplanar) image.
- Distortion occurs due to the geometry of the lens, this is the reason for a practical limitation in the magnification.

After crossing the lens the light, in digital cameras, reaches the Charged-Coupled Device (CCD), and here another problem arise due to the fact that cameras in most cases uses a three color model technique to capture the light (RGB). For each pixel there are three detectors in the sensor. This cause a effect called "lateral displacement", i.e. there is a spatial displacement from the video signal and the point in the world.

Another loss of information occurs when sampling and quantization is applied to digitalise the image.

2.3 Projective Geometry

Intrinsic Conditions- Although focal length is the most emphasized internal camera geometry parameter, there exist more complex parameterizations. In fact, real cameras have many other internal geometry variables. For an ideal pinhole camera to deliver a perspective image a mapping must be done. This mapping can be characterized completely by using six parameters, called the intrinsic parameters of the camera:

- focal length, in pixels (f);
- x-coordinate of the center of projection (u_0)
- y-coordinate of the center of projection (v_0);
- scaling of the image plane along the x (s_x)
- scaling of the image plane along the y (s_y) axes;
- skew between the optical axes ($s\theta$).

The matrix K includes all six intrinsic parameters, which is effectively reduced to five parameters because s_x and s_y are dependent on each other.

$$K = \begin{bmatrix} fs_x & fs_q & u_0 \\ 0 & fs_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The image coordinates (x_i,y_i) are related to the object coordinates (x_o,y_o,z_o) by the following relation:

$$w_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = K \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

Extrinsic Conditions- There are six extrinsic camera parameters: three are for the position of the center of projection, and three are for the orientation of the image plane coordinate frame:

$$P = \left[R_{(3x3)} T_{(3x1)} \right]$$

The final relation between image coordinates (x_i,y_i) and object coordinates (x_o,y_o,z_o) is given by:

$$w_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = PK \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

The result of the matrix multiplication PK is the calibration matrix this transformation matrix gives the relation between points in the image plan and the pixels in the sampled image.

The fundamental geometric relationship between two perspective cameras is represented in Figure 2.3.

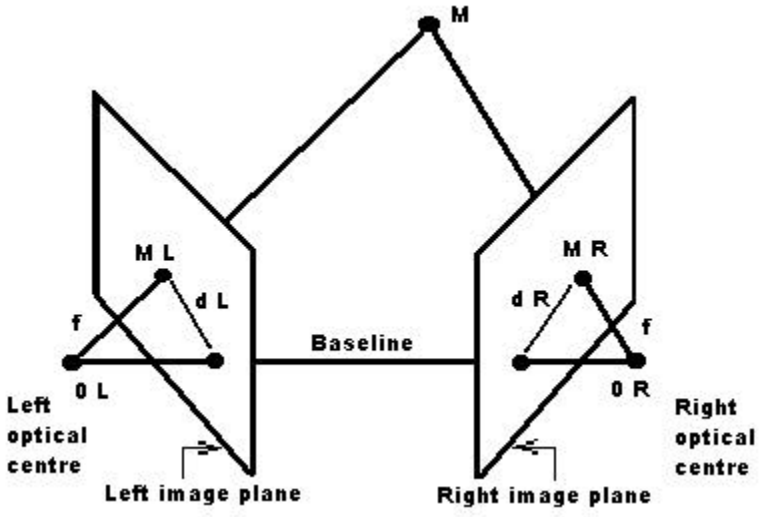


Figure 2.3

The epipole is the point of intersection of the line joining the optical centers (the baseline) with the image plane. The epipole is the image in one camera of the optical centre of the other camera. The epipolar plane is the plane defined by a 3D point and the optical centres or equivalently, by an image point and the optical centres.
The epipolar line is the straight line of intersection of the epipolar plane with the image plane. It is the image in one camera of a ray through the optical centre and image point in the other camera. All epipolar lines intersect at the epipole.
Stereo vision determines the position in space by using triangulation. That is, by intersecting the rays defined by the centers of projection and the image. Triangulation depends crucially on the solution of the correspondence problem.
The disparity measures the difference in retinal position between corresponding points in two images. Depth is inversely proportional to disparity. It is possible to verify by looking at moving objects that distant objects seem to move more slowly than closer ones.

2.4 Feature Extraction

The sequence of operations of most computer vision system begins by the detection, location and representation of special parts of the image, called image features, usually corresponding to interesting elements of the scene.
In computer vision the term image feature refers to two possible entities:
A global property of an image or part thereof, for instance the average grey level the area in pixel (global feature).
A part of the image with some special properties, for instance a circle, a line, or a textures region in an intensity image, planar surface in a range image (local feature).

How to detect special parts of intensity and range images like points, curves, particular structures of gray levels or surfaces patches represents the second definition and the focused one.

Image features are local, meaningful, detectable parts of the image.
Meaningful means that the features are associated to interesting scene elements via the image formation process, such as sharp intensity variations created by the contours of the objects in the scene, or image regions with uniform gray levels, for instance image planar surfaces.
Detectable means the location algorithms must exist, otherwise a particular feature is no use. Different features are, of course, associated to different detection algorithms, these algorithms output collections of feature descriptors, which specify the position and other essential properties found in the image.
Image features can be edges, points, corners, surfaces, lines, curves, etc.

Edges points or simple edges, are pixels at or which values undergo a sharp variation. Image edges are commonly presented as discontinuities in the underlying irradiance function, but it seems more accurate to speak of sharp image variations than discontinuities, the reason being that the scene radiance is low pass filtered by optics and the resulting image brightness cannot have real 0-order discontinuities.
There are various reasons for the interest in edges. The contours of potentially interesting scene elements like objects, marks in surfaces and shadows, all generate intensity edges. Moreover, image lines, curves, and contours are often basic elements for stereopsis, calibration, motion analysis and recognition, are detected from chains of edges points.
The edges detection bottleneck is to locate the edges generated by the scene elements and not by the noise. The trade off is to suppress the noise as much as possible, without destroying the true edges.

Corners can be characterized more easily than edges in mathematical terms, but do not correspond necessarily to any geometry entities of the observed scene. These features can be interpreted as corners, but not only in the sense of intersections of the image lines. They capture corners in patterns of intensities. Such features are stable across sequences of images, and are therefore interesting to track objects across sequences.
A corner is identified by two strong edges, feature points include high contrast image corners and junctions generated by the intersection of objects contours, but also corners of the local intensity pattern not corresponding to obvious scene features.
In general terms, at corners points the intensity surface has two well-pronounced and distinctive directions.

Many objects, especially man-made, can be conveniently described in terms of shape and position, of the surfaces they are made of. Surface based descriptions are used for classification, pose estimation and reverse engineering, and are omnipresent in computer graphics.
The solution for several computer problems involving 3D models is simpler when using 3D features than 2-D features, as image formation must be taken into account for the latter.
To solve the problem its needed two tools: a dictionary of shape classes and a algorithm determining which shape class approximates best the surface at each pixel.

Lines and curves are important features because they define the contours of objects in the image.
Lines extraction is difficult because of pixelization and errors introduced by image acquisition and edge detection, there is no line going exactly through all the points, it has to be found the best compromise for line.
In Lines and Curves extraction there are problems to overcame. Which image points in the image compose each instance of the Line or Curve, and given a set of image points probably belonging to a single instance of the target Line or Curve, find the best Line or Curve interpolating the points.
Another problem that arises is due to accidental positioning two 3D Lines or Curves far apart from each other project onto close image Lines or Curves.

In 3D computer vision feature extraction is an intermediate step, not the goal of the system. We do not extract features, just to obtain features representations, we extract deatures to navigate robots, to decide whether an image contain a certain object, to calibrate cameras, etc.

2.5 Correspondence Problem

Correspondence consists in determining which item in the left camera corresponds to the item in the right camera. A rather subtle difficult is that some arts of the scene are visible by one camera only. Therefore, a stereo system must also be able to determine the image parts that should not be matched.
The correspondence problem involves two decisions: which image element to match and which similarity measure to adopt.

The correspondence algorithms can be classified in two classes, correlation-based and feature-based methods. The correlation-based methods apply to the totally image points, while feature-based methods attempt to establish a correspondence between sparse sets of image features.

In correspondence correlation-based methods, the elements to match are image windows of fixed size, and the similarity criterion is a measure of the correlation between windows in the to images. The corresponding element is given by the window that maximizes the similarity within a search region.

In correspondence feature-based method, there is correspondence search restricted for a sparse set of features. Instead of windows, they use numerical and symbolic properties of features available from feature descriptors. Instead of correlation like measures, they use a measure of the distance between feature descriptors.

Unfortunately there is no correspondence method giving optimal results under all possible circumstances. Choosing the method depends on factors like the application, the available hardware, or software requirements.

Correlation-based method is easier to implement and provide dense disparity maps for the purpose of reconstructing surfaces. They need textured images to work well. However, due to foreshortening effects and change in illumination direction, they are inadequate for matching image pairs taken from very different viewpoints. Also, the interpolation necessary to refine correspondences from pixel to subpixel precision can make correlation-based matching quite expensive.

Feature-based methods are suitable when a priori information is available about the scene, so that optimal feature can be used. A typical example is the case of indoor scenes, which usually contain many straight lines but rather untextured surfaces. Feature-based algorithms can also prove faster than correlation-based ones, but any comparison of specific algorithms must take into account the cost of producing the feature descriptors. The sparse disparity maps

generated by these methods may look inferior to the dense maps of correlation-based matching, but in some applications (e.g., visual navigation) they may well be all you need in order to perform the required tasks successfully. Another advantage of feature-based techniques is that they are relatively insensitive to illumination changes and highlights.

2.6 3D Reconstruction

If the geometry of the stereo system is known, the disparity map can be converted to a 3D map of the viewed scene that is a 3D reconstruction. Our 3D perception of the world is due to the interpretation that the brain gives of the computed difference in the retinal position, named disparity, between items. The disparities of all the image points form the disparity map, which can be displayed as an image.
So given a number of corresponding parts of the left and right image, and possibly information on the geometry of the stereo system, it is possible to obtain the 3D location and structure of the observed objects.
The 3D reconstruction that can be obtained depends on the parameters amount of priori knowledge available on the parameters of the stereo system. There are tree cases possible to identify:

- If both intrinsic and extrinsic parameters are known, in this case it is possible the reconstruction problem unambiguously by triangulation.
- If only the intrinsic parameters are known, in this case it is possible to solve the problem by estimate the extrinsic parameters of the system, but only up to an unknown scaling factor.
- If the pixels correspondences are the only information available and neither the intrinsic nor the extrinsic parameters are known, it is possible to obtain 3D reconstruction of the environment, but only up an unknown, global projective transformation.

Reconstruction by triangulation is the simplest case. If you know both the intrinsic and extrinsic parameters of your stereo system, reconstruction is straightforward.

Reconstruction up to a scale factor is when only intrinsic parameters of both cameras are known, extrinsic parameters as well as the 3D structure of the scene are derived. Unlike triangulation, in which the geometry of the stereo system was fully known, the solution cannot rely on sufficient information to locate 3D points unambiguously.
Since the baseline of the system is unknown it isn't possible to recover the true scale of the viewed scene. Consequently, the reconstruction is unique only up to an unknown scaling factor.This factor can be determined if we know the distance between to points in the observed scene.

Reconstruction up to a projective transformation is a 3D reconstruction even in the absence of any information on the intrinsic and extrinsic parameters. This reconstruction is unique only up to unknown projective transformation of the world. It is worth noticing that, if no estimates of intrinsic and extrinsic parameters are available and non-linear deformations can be neglected the accuracy of the reconstruction is only affected by the algorithms computing the disparities, not by calibration.

2.7 Real-Time Problem

A real-time application is one that can respond in a predictable, timely way to external events. Real-time system requirements are typically classified as hard or soft real-time. For a hard real-time system, events must be handled predictably in all cases; a late response can cause a catastrophic failure. For a soft real-time system, not all events must be handled predictably; some late responses are tolerated. For many real-time computer vision applications, a soft real-time system is sufficient. For example, in a real-time gesture recognition system, it may be tolerable to occasionally or systematically drop video frames, as long as the system is designed to robustly handle frame drops.
For fast tracking systems a drop video frame is not tolerable.
Real time vision is interested the visual information that can be extracted from spatial and temporal changes occurring in an image sequence.
The temporal dimension in visual processing is very important, the apparent motion of objects onto the image is a strong visual cue for understanding structure and 3D motion.

In order to acquire fast image sequences it is necessary a frame grabber capable of storing frames at fast rate. If the grabbing rate is allowed to be chosen, then it should be fast enough to guarantee that the discrete sequence is a representative sampling of the continuous image evolving over time. In order to process the images inside the time rate fast algorithms must be implemented. In stereopsis it should be taken into account the image grabbing time, to synchronize the cameras.

3 KINEMATIC ESTIMATION AND CONTROL USING VISUAL FEEDBACK

3.1 Introduction

This chapter is divided in four main subjects.

The first subject is related to the system architecture, where the architecture of the vision system, control system and global system, which is composed by the two previous, is described.

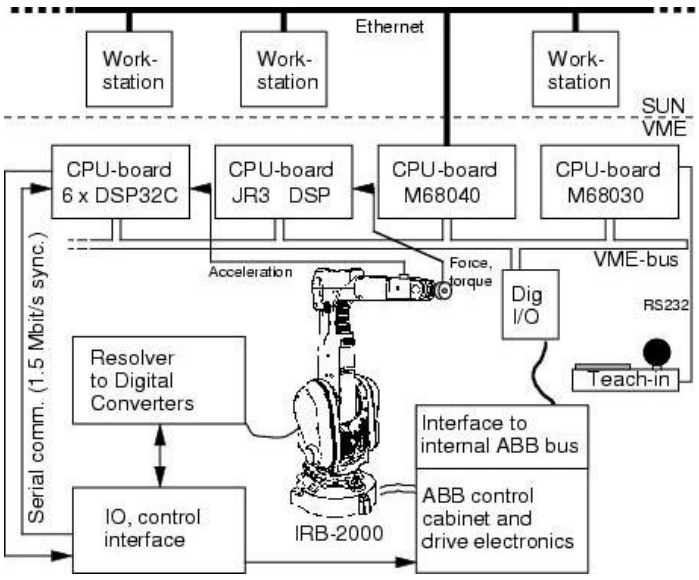
The second main subject explains the experiments carried out in the investigation.

The third subject describes the vision processing in every stage.
The description is organized in four elements: the virtual Java robot image processing, the processing of the raw image, the lattice and the object recognition.

3.2 Experimental Setup

The architecture of the robot software is an open architecture.
An open robot architecture allows the system to be connected easily to devices and programs made by researchers and manufacturers. A system with a closed architecture, on the other hand, is one whose design is proprietary, making it difficult to connect the system to other systems.

Sun workstations are used for software development, control engineering, as well as for robot operator interaction. The robots IRB-6 and IRB-2000 are controlled from VME-based embedded computers. Signals from the internal sensors of the robot to the VME system go via the sensor interface to the DSP board connected to the VME bus.



Hardware configuration of the open architecture robot controller

The **robot** characteristics of the IRB-2000: it has 6 DOF (Degrees of freedom) and a precision of 0.1 mm. Joint 1, 4, 6 are cylindrical joints and the others three joints 2, 3, 5 are revolute joints. This robot can reach all the points in the possible work area with arbitrary orientation.

The IRB-6 is a robot with five degrees of freedom and a precision of 0.2 mm. The joints are connected with six links, the joints 1 and 5 are cylindrical and the others (2,3,4) are revolute joints.

The **control system** used in RobotLab is composed in three modules: IgripServer, Trajec and Regul. This control system is for both robots and only it is different in Cartesian coordinate system because they have different degree of freedom, Figure 3.1.

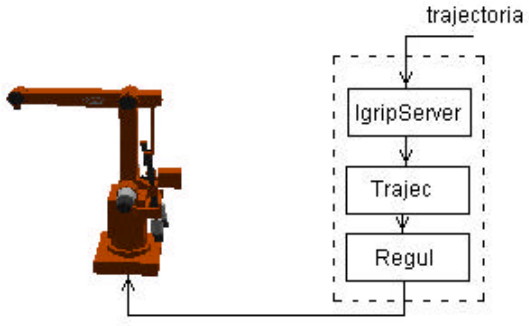


Figure 3.1

Control System

IgripServer module communicates by a socket to an application, e.g. Matlab, which generate the trajectory. Sockets sets up a two ways network communication and usually do not use TCP. This makes it

possible to communicate between computers, so the application only has to be run on a computer, which is connected to the network. The IgripServer receive pre-calculated trajectories and send sends to the Trajec module.

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

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

The **stereo vision system** is built with this robot and the two cameras see 3.2. The robot has mounted in the end-effector an uncalibrated stereo rig. This stereo rig has two cameras with a unknown distance and angle between them. The distance and the angle between cameras have not been measured because the goal is to overcome the intrinsic and extrinsic conditions without any calculation.

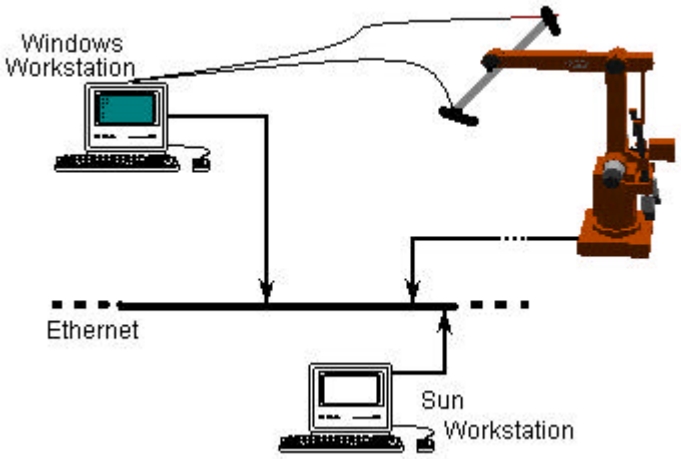


Figure 3.2

Vision System

The cameras communication is made by Fire-i, that is a standard IEEE 1394-1995 for High Performance Serial Bus.
The cameras are connected with the board FireBoard 400 that works in Windows stations with a CPU 400 MHz. This machine is connected to the same Ethernet that the Sun workstations use to communicate with the robot IRB-6 and IRB-2000.

The **cables** used between the cameras and the board work for all frequencies between 100MHz and 400MHz.

The **cameras** used are Sony DFW V-300 that utilizes the IEEE 1394 high performance serial bus to provide, non-compressed YUV digital

data for image capturing, compression and transmission systems. With the DFW V-300 are possible control functions such as color tone, brightness, picture quality, white balance and AGC (Automatic Gain Control). The camera signals are transmitted with data rates of 200 megabits and 30 images per second. The picture format of 640 x 480 pixels (4:1:1) can be modified to 320x240 pixels (4:2:2) that is the resolution used.
A video picture is an array of pixels. The position of each pixel is defined by 2 numbers: horizontal & vertical co-ordinates inside the array.
With this camera is possible changing several parameters but only the shutter and the gain is adjusted in our software.
The shutter is used to reduce camera sensitivity in high light conditions, to reduce picture blur for fast moving subject, and to match camera integration time to pulsed light sources in order to avoid flickering.
The gain works combined with iris control. Increasing exposure by gain adds noise to the picture and increasing exposure by iris reduces the depth of field of the picture.

The **Fire Wire** is a high-speed, non-proprietary, scaleable digital serial bus that can transport data at rates of 100, 200 and 400 Mbits per second. The bus enables true plug and play which allows the user to connect new devices with the system switched on and the bus active.

The features are:
- dates rates of 100, 200 and 400 Mb/s
- real time transmission of data is ideal for video & audio
- universal I/O interconnect
- peer-to-peer communication structure
- based on a "memory-mapped-like" architecture
- backwardly compatible speeds
- bus is dynamically configurable and active termination is not needed

Matcomm is software that was developed in Department of Automatic Control. This software works with protocol TCP/IP to transmit data between computers in network and with different systems. The reason to use Matcomm is because it is a easy and fast way to connect with another machine without necessity of the setting all the parameters. Matcomm can be use in Unix and Windows environment. Data are sent using sockets in array format.
The vision system, shown in Figure 3.2, is a closed loop that starts in the cameras, which interact with the world environment. Then the images of both cameras are processed in a PC running Windows platform. The visual feedback is sent by Matcomm to the Sun workstations that communicates and control the IRB-6 and IRB-2000.

3.3 Experiments

3.3.1 Calibration Movement

This experiment was the most important one in the investigation, and the one in which more time resources were spent.
It consists in the acquisition of disparities maps called layers for a set of well-known relative depths.
It was used a calibration plate to acquire the feature points and build the disparity map.
The disparity map for a certain depth is called layer and the set of all the layers is called lattice.
The calibration plate is composed by a reflecting surface with structured non-reflective holes on it.
The disposition of the holes is done in a way that a cross can be found in the centre of the plate. The holes have a linear distribution in x and y coordinates (Figure 3.3).
An IRB-6 robot performs the movement of the stereo rig between depths; this represents very accurate depth position. Using IRB-6 for depth position an add feature is obtained, the depth positioning is actually a z positioning, and the x and y positioning are the add feature.

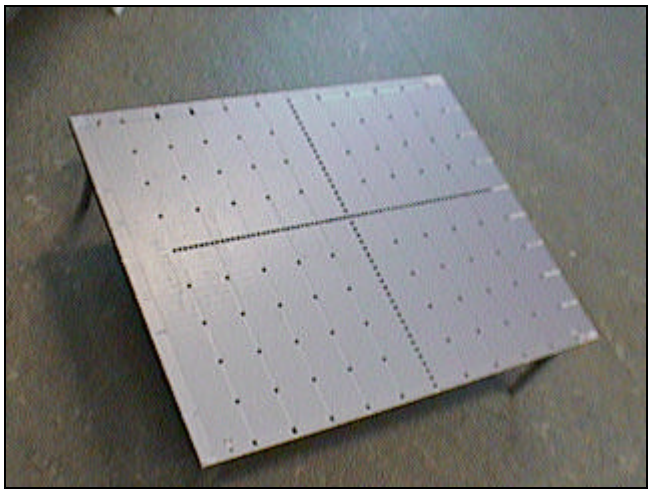


Figure 3.3

The x and y positioning is used in order to position the robot right above the plate, which means more feature points and more reliable lattice.
The lattice is made from several layers corresponding each layer to one z relative distance to the calibration plate. A layer is composed by the displacement of each feature point visualized by both cameras, so in this way using interpolation, it is possible to build an x and y characteristic displacement for the image area covered by both cameras. This is done for a set of z relative distances to the calibration plate. After all the layers been saved it is possible to estimate the

relative distance of any point in image area covered by both cameras, using either interpolation for points with a depth between the layers, for depths higher or smaller than the set of relative distances saved in the lattice the depth is extrapolated.

3.3.2 Positioning Movement

In this experiment, we address the problem of visually guiding and controlling a robot in projective 3D-space using stereo vision. The experimental setup is composed by one stereo rig mounted on the end-effector of IRB-6 robot, and two cross drawings. One in the object and the other in the end-effector of the IRB-2000 robot.
The method is entirely formulated using comparison of the disparity between the cameras and applying it to the lattice built in the calibration movement.
The IRB-6 was used to position the cameras in a position where the two crosses were in the image area covered by both cameras.
The IRB-2000 is used to position the cross centre, mounted in the end-effector, right above the cross centre located in the object.
This was performed in real time, and the control feedback minimizes the error computed by the depth, x and y difference between the crosses centre.
The experiment is done with the object stopped and with the object in motion.

3.3.3 Virtual Robot (Java Robot)

Virtual reality can be used to test some applications implemented in the real world.
The virtual robot was used to test algorithms of control, image processing and the communication between the different modules.
Using virtual robot is possible to avoid collisions and bad trajectories before using in the real robot and it is a fast and safe test bench for the system.

The graphical appearance of the virtual environment is shown in Figure 3.4.

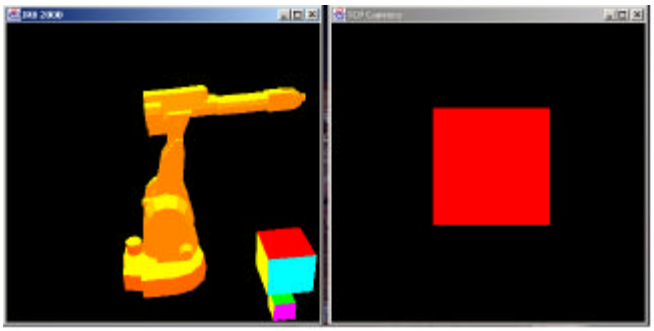


Figure 3.4

Virtual Robot

3.4 Real Time Stereo Vision Pipeline

3.4.1 Image Processing

The image constitution follows a three-color model and its respective RGB components can be retrieved from the image buffer.
The plate is made of reflective material, see Figure 3.5.
By adjusting the shutter and gain a natural contrast between the color of the plate surface and their holes is achieved.

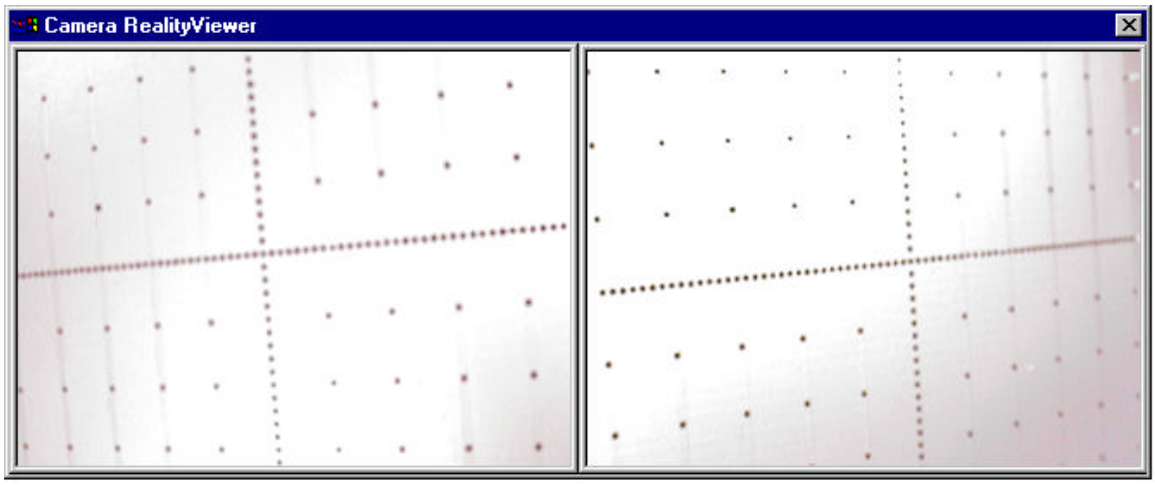


Figure 3.5

The image is stored in a buffer and then displayed in order to provide visual feedback on the effect of changing the shutter and gain.

In the experimental setup the light conditions are controlled and the image contain enough information to recognize the feature points, in this case the holes on the plate. The colored image is passed through a threshold filter to produce a binary image.

A Line Scan algorithm was implemented to perform the threshold and acquire the x, y positions of the feature points.

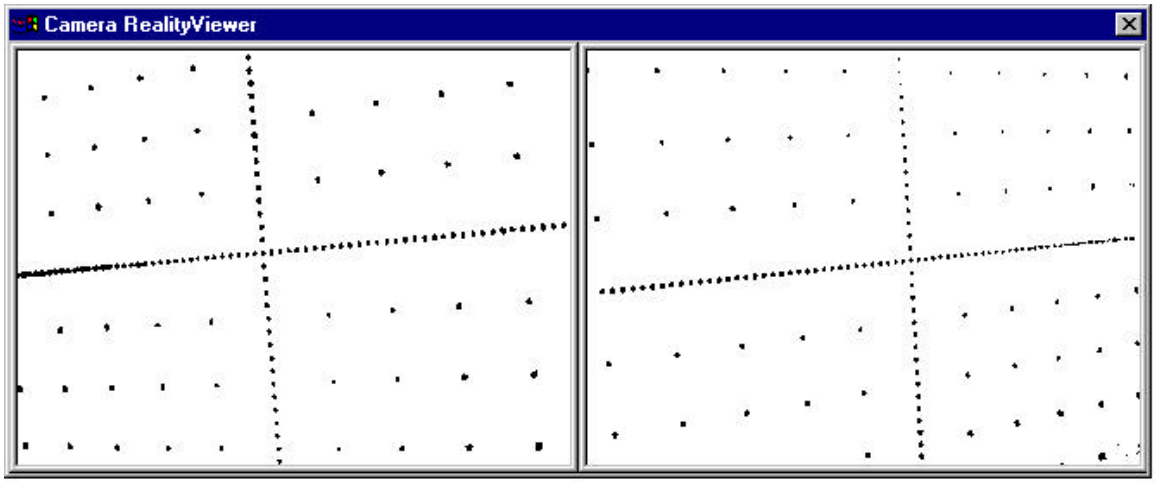


Figure 3.6

The threshold limits for the plate application were:

$$B(x,y)=\begin{cases}1, if & (R(x,y)<200)and(G(x,y)<200)and(B(x,y)<200)\\ 0, if & otherwise\end{cases}$$

the resulted image is shown on Figure 3.6.
Due to the difference between the reflective surface and the holes on the calibration plate a high contrast is achieved.
This threshold is sufficient for finding feature points.

The color mass is calculated for each feature point. It is then used in a filter to exclude feature points from noise.

The noise can be differentiated from feature points by excluding small and large color regions.
The limiting threshold parameters can be changed online
On start-up they are: minimum color mass equals 5 and maximum color mass equals 100.

$$FP(x,y)=\begin{cases}1, if & (B(x,y,m)>5)\,\&\,(B(x,y,m)<100)\\ 0, if & otherwise\end{cases}$$

Where m is the color mass and FP is feature point.
After the filtering the color mass center is computed and saved in Feature Point (feature point) x and y array.

Figure 3.7 displays the feature points after the applying the filter.

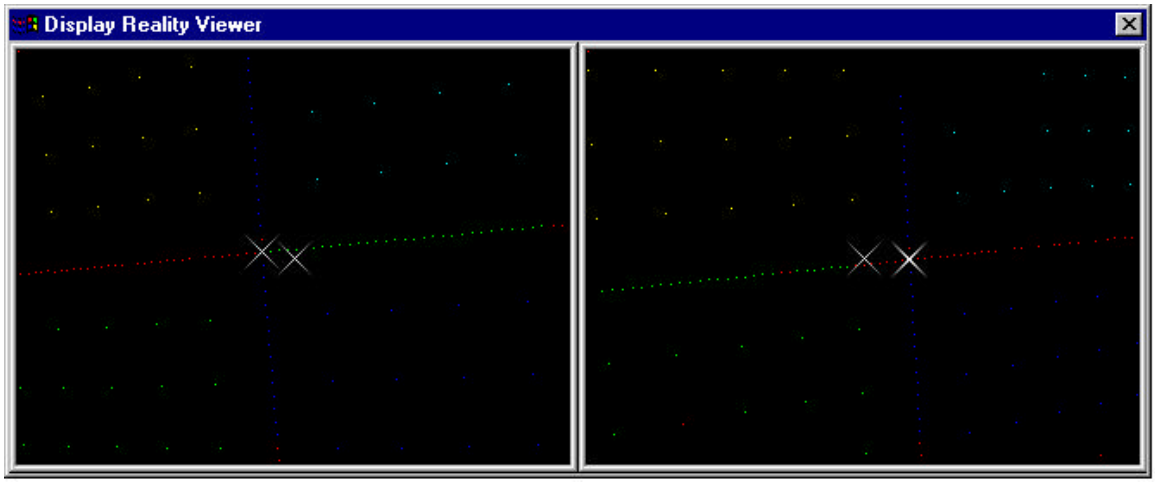


Figure 3.7

By applying a threshold followed by filtering the amount of information to be processed is reduced.

3.4.2 Closest Neighbor Vector Field

In this stage the goal is to find the center of the cross. It is important to exclude feature points that aren't included in the cross.
The x and y position stored for each feature point is now used to build a relation between every point and its closest neighbor, this is done by calculating the euclidean distance to a every feature point with no closest neighbor already associated.
The feature point with the minimum distance associated is store to an array of closest neighbors.
In Figure 3.8 it is shown the vectors from each point its closest neighbor.

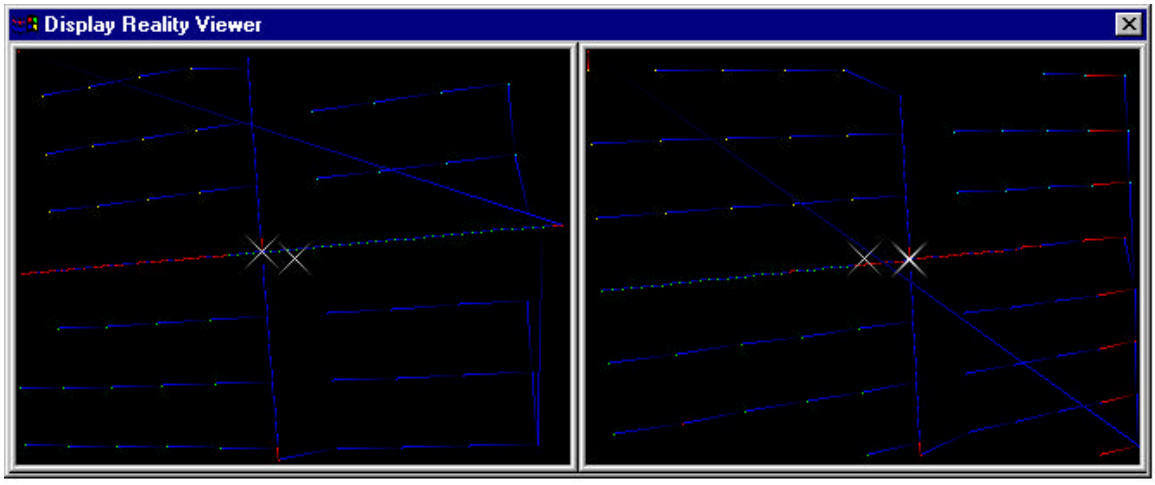


Figure 3.8

It is important to notice that once a feature point has a closest neighbor associated he cant no longer be a closest neighbor for others

feature points. Each feature point has a unique closest neighbor. This algorithm was chosen to prevent two feature points from having the other as closest neighbor.
Other way could be the Hough transform but we didn't use this method.

The reason why almost all vectors have their direction to the right or to the bottom is because the feature points were stored from left to the right, top to the bottom when the LineScan was made.

Now we have all points with a vector associated, but we still don't know which vector belongs or not to the cross.
In order to get the cross, several filters were implemented, the first one was based on the average length of the vectors.

As it is possible to see from the Figure 3.8 the vectors that belong to the cross have a shorter length than the others do. Since the vectors belonging to the cross have a bigger rate of occurrences than the others do, if an average is computed, the value of the average will be close to the size of vectors belonging to the cross. This is used as a term of comparison to exclude more points.

$$FP(x,y)=\begin{cases}1, & if \quad \left(\left|\vec{V}\right| > 0.03*\left(\dfrac{\sum_{i=1}^{n}\left|\vec{V}\right|}{n}\right)\right) and \left(\left|\vec{V}\right|) < 2.5*\left(\dfrac{\sum_{i=1}^{n}\left|\vec{V}\right|}{n}\right)\right)\\ 0, & if \quad otherwise\end{cases}$$

Where $\left|\vec{V}\right|$ is Euclidian length and $\left(\dfrac{\sum_{i=1}^{n}\left|\vec{V}\right|}{n}\right)$ is the Euclidian average length of vectors.

It must be taken into account that several aberration vectors will ruin the average, because they are thousands of times bigger than the vectors that belong to the cross. In order to avoid the bad effect of the aberration vectors a previous filter is superimposed, it takes the previous average and excludes large vectors. This filter introduces a problem by using old averages, but still it is very useful if the first measure could be rejected.

$$FP(x,y) = \begin{cases} 1, if & \left(|\vec{V}| < 10 * \left(\frac{\sum_{i=1}^{n} |\vec{V}|}{n} \right)_{t-1} \right) \\ 0, & \\ if & otherwise \end{cases}$$

Where $|\vec{V}|$ is Euclidian length and $\left(\frac{\sum_{i=1}^{n} |\vec{V}|}{n} \right)_{t-1}$ is the Euclidian average length of vectors in the previous image.

After this filter almost every vector that doesn't belong to the cross was eliminated. It is possible to change the parameters of the filters online.

3.4.3 Line Segments Extraction

Once almost all the undesired points have been excluded, the points that have an similar vector length are left.

In order to exclude more points that could be noise or points that aren't noise but don't belong to the cross, a direction filter is required. The direction filtering takes into account that all the vectors have an angle associated, which can vary from 0 to 90 degrees. This is very important to state, because other ranges must have different approaches. The angle of a vector is taken by the x and y size of the vector and then clustered to the first quadrant.

The direction filtering is done together with a new vector length filtering. Lets take the example on Figure 3.9 with the vectors from 0 to 7, vector i (Vi), when i ∈ {1.. 7}.

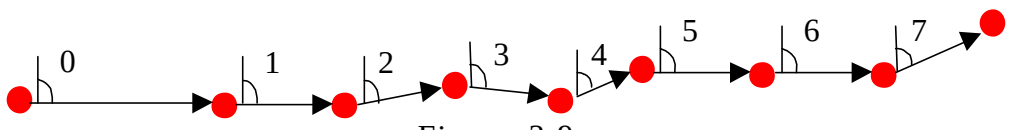


Figure 3.9

Lets start on vector 0 (V0) and take it as a reference of analysis. When compared with V1 the angle is the same but the size is larger so the chain is broken here since there was no similarities among the to vectors. Now lets take the V1 as reference. When compared with V2 the size is almost the same and the angle as well, so the two vectors are consider belonging to the chain, and it proceeds to the next vector. The vector V3 is compared with the average size and angle of the

chain, their values are inside the limits, so V3 is also included in the chain.
The next vector to be evaluated is V4 and their values are quite different from the average size and angle of the chain, which makes a break on the chain.
Since the vectors included in the chain are in sufficient number to be consider as a connected vector, (i.e. it has more then 2 vectors included in the chain) then the first point and the last point on the chain are saved to an database of connected vectors called CV.

The last chain was finished and a new chain starts on V4 but it has no continuation due to the disparity of the sizes and angles
The process starts again on V5. The analysis of V6 includes it on the chain but excludes V7. Now there are only two vectors on the chain, so these vectors are discarded.

In Figure 3.10 the several chains that fulfilled all the requirements are shown. They are represented as straight lines.

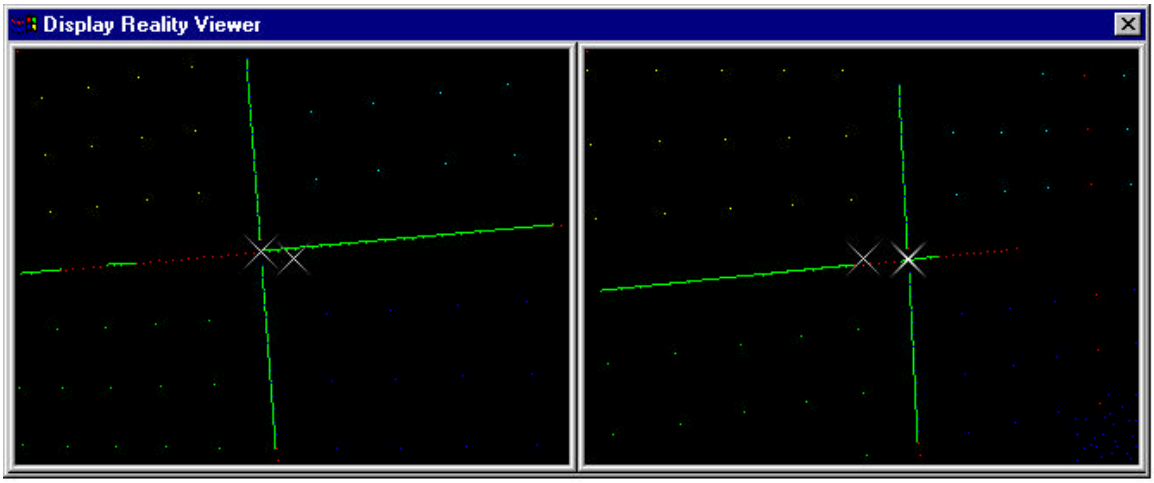


Figure 3.10

All the points included on the chain, (i.e. all the points associated with vectors included on the chain) were stored to an array, where they were clustered to the respective chain.
It is important to notice that all relevant information related to each chain is saved to a database of connected vectors, such as feature points included in the chain, the total number of points included in the chain, the start/end position of the chain.

The size limits is set up to minimum 0.05 and maximum 2.0.
The allowed angle variation is set to ±10°.
The minimum number of vectors in the chain is set to 3.
All the values mentioned above could be changed online for test results evaluation.

$$CV\begin{pmatrix} x_{stzrt}, \\ y_{start}, \\ x_{finish}, \\ y_{finish}, \\ FP_{chain}, \\ NFP_{chain} \end{pmatrix} = \begin{cases} 1, & if \quad \left(0.05 * \left(\frac{\sum_{i=1}^{n} |\vec{V}|}{n}\right) < |\vec{V}| < 2 * \left(\frac{\sum_{i=1}^{n} |\vec{V}|}{n}\right)\right) and \\ & \left(\left|\angle\vec{V} - \left(\frac{\sum_{i=1}^{n} \angle\vec{V}}{n}\right)\right| < 10^{\circ}\right) and (n > 2) \\ 0, & if \quad otherwise \end{cases}$$

Where $|\vec{V}|$ is Euclidian length and $\left(\frac{\sum_{i=1}^{n} |\vec{V}|}{n}\right)$ is the Euclidian average length of vectors, $\angle\vec{V}$ is the vector angle and $\left(\frac{\sum_{i=1}^{n} \angle\vec{V}}{n}\right)$ average angle of vectors and n is the n of connected vectors included.

The above structure represents the algorithm implemented.

3.4.4 Lines Extraction and Center Cross Estimation

At this stage the information stored in the database is enough to estimate the center of the cross, but still before finding the cross center the connected vectors have to be classified as Horizontal Line (HL) or Vertical Line (VL). Finding the cross center is then a question of solving the intersection of two lines.

The group of CV that belong to HL are defined as the ones that have a longer x than y length. The inclusion on the HL is not straightforward, there is a filtering process in between. For instance the x versus y length can satisfy the requirements but there are situations in which it shouldn't be included in that set.
Lets take a look on the example stated in Figure 3.11.

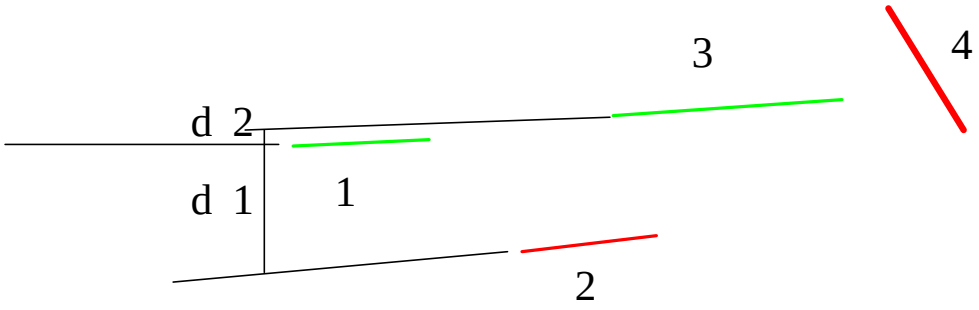


Figure 3.11

The first Connected Vector CV1 has a similar angle to CV2 but is obvious that it shouldn't belong to the same set as CV1. The two CV are parallel and the way to exclude from a set and include in another set, no longer can be their angle. From CV1 a normal vector is traced until it intersects the line that includes CV2, the length d1 is the resulted prolongation of the normal vector.
The length of the normal vector is a relevant variable, due to its importance the normal vector length is used in grouping the CV in different sets.
In Figure 3.11 the previous fact can be stated, CV1 and CV3 are included in the same set, but CV2 is not included, CV4 is obviously not included in none of the sets for the HL.
The CV4 is a vector in which the x length component is smaller than the y so this one will be clustered in one of VL sets.
After fitting every CV in the correct group and line (HL or VL), it is time to assign the correct set as function of the line. This is done in a very simple way, the set with greater amount of feature points associated to the CVs belonging to that set; is the chosen one.

The feature points assigned to each line don't actually represent the line, first it is necessary to use a least square line method and fit the line in there.

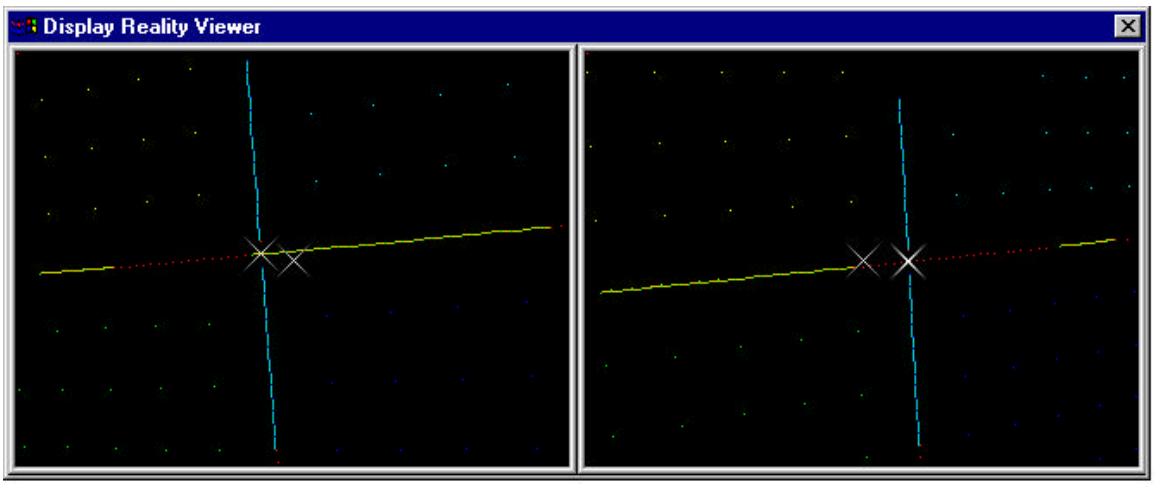


Figure 3.12

The least-squares line uses a straight line $y = a + bx$ to approximate the given set of data, feature point (x_1, y1), feature point (x_2, y2),..., feature point(x_i,y_i). The values a and b are unknown coefficients while all feature point (x_i,y_i) are given. To obtain the least squares error, the unknown coefficients a and b must yield zero first derivatives.
The unknown coefficients a and b can therefore be obtained.
This is applied to both lines HL and VL. The centre of the cross is the intersection given by the to lines equation, see Figure 3.12.

3.4.5 3D Lattice and Feature Points Correspondence

To fill the lattice it is necessary to know which feature point in the left camera corresponds to the feature point in the right camera
A better accuracy of the lattice means as many feature point as possible in the image area covered by both cameras.
The centre of the cross as already been found and it is possible to use it as feedback in order to center robot above the centre of the plate, which means more feature point as desired. It is important to state that the centre of the cross is also the center of the plate.
Before finding the correspondence of feature point, it is important to group them for further processing.
The grouping of the feature point was performed by override all feature point and fit them into a parallel lines to the VL, the HL parallel lines weren't suitable to cluster feature point. It was used VL parallel lines to cluster feature point because the real lines made by the holes in plate, were still parallel in the projective image plane, but the same didn't happen with the horizontal lines.
The horizontal lines when projected to the image plane suffer a tilting effect, and that makes them harder to use as a method for clustering feature point.
In Figure 3.13 this effect is shown, by analyzing the tree VL it is possible to observe that they have almost the same slope, rather the HL lines which have a very different slope due to the tilting effect.

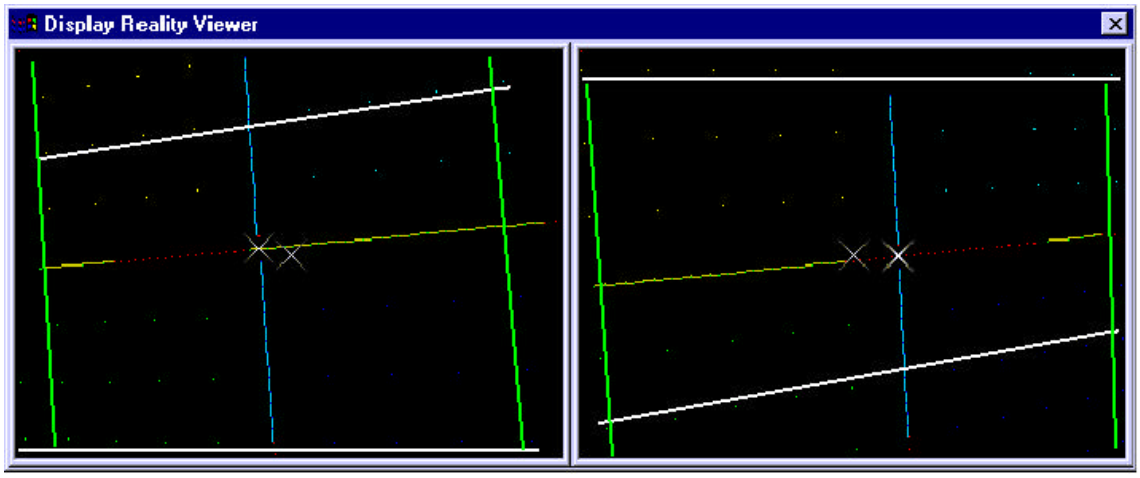


Figure 3.13

As described above the VL lines can be used for grouping feature point, the Figure 3.14 exhibits how feature points were grouped.

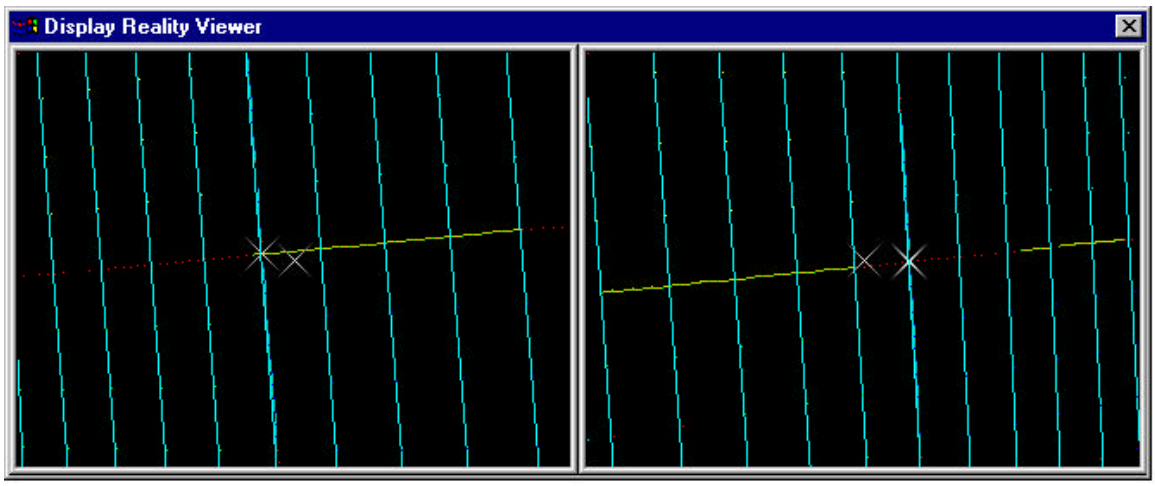


Figure 3.14

The image was override by a parallel line to the VL belonging to the cross and the feature point were inserted into groups, first line groups and further into left line groups and right line groups, this is VL parallel (VLP) that were situated to the left or right side of the VL belonging to the cross (VLC).
The VPL to each side were numbered increasing from their distance to the VLC, this is first VLP to the left side was the immediate line to the left side and so on to left border of the image. The same numbering process was used for the right side VLP.
In order to have a easier correspondence, the feature point were again clustered into two new set inside the VPL left and right. The feature point above the HLC were grouped into a new set, HCL up and the feature point below the HLC were grouped into a new set, HCL down, and they numbering followed the assumption that more far from the HL more high would be their index.
In Figure 3.15 the numbering of the VLP follows the horizontal arrows direction, and the numbering of the feature point inside the four sets follows the vertical arrows direction,

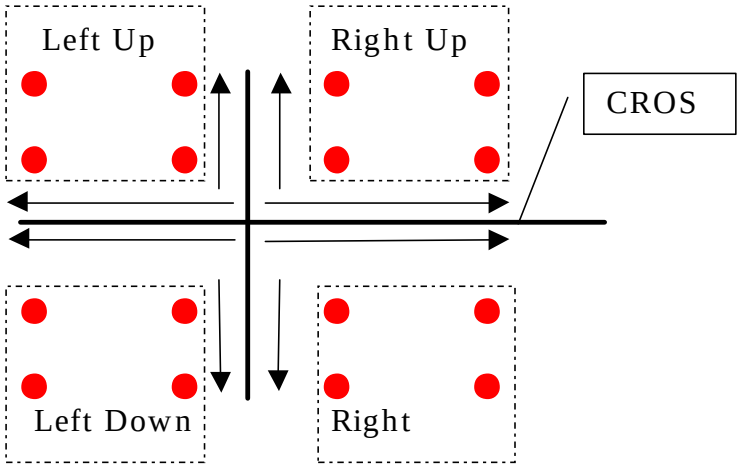


Figure 3.15

This enumeration method was chosen because the centre of the cross is well known, as the VL and HL, so this information is used for correspondence proposes.
After this complex clustering the correspondence is straightforward problem, the points always find their correspondence in the same database position for the other camera, e.g. lets analyze the first the left top point in the Figure 3.15, it is included in the second left VLP, then he is grouped to the top left values and is position is the second, so its final position is indexed to second VLP and second feature point in the top group.
To check the correspondent point in the other camera it is just a question of retrieving the point on the same position in the database of the other camera.
A rather subtle difficult in correspondence is that some feature point visible for one camera but not for the other, this is overcame by the previous method, if the x and y values in one database are null it means that the point isn't seen by the camera.

The information contained in the database is displayed in the next two Figures.

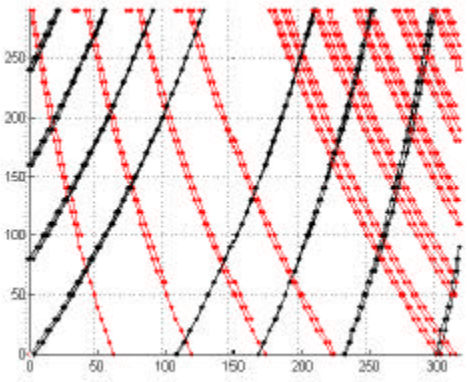


Figure 3.16

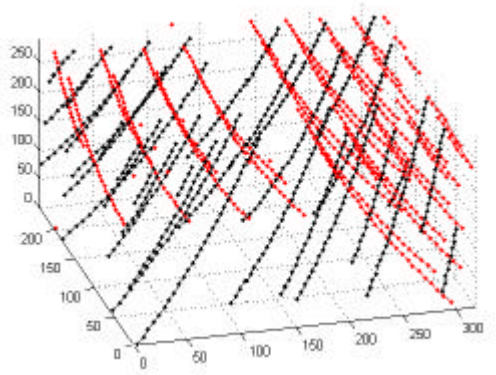


Figure 3.17

In Figure 3.16 the three dimensional feature point are displayed in two dimensions x (0,320) versus z (0,290), each line corresponds to the trajectory of the one feature point in the image plane while the stereo rig is moved away from the plate, the black and lines represent the feature point trajectory in the left camera and right camera respectively.
It is observed that when the stereo rig moves away from the plate the feature point's tend to move to the opposite way of the tilt camera angle, and more feature point are included in the image area plane.
This is better stated in Figure 3.17 where a 3D view of the feature point's trajectory is displayed.
This means that more far from the plate better is the accuracy of the database since more points are included, and smaller interpolation space is obtained.
It is also possible to visualize the several z layers constituents of the database.
This information combined gives the x and y displacement for several z layers, which builds a characteristic displacement map for all the feature point with different depths.

3.4.6 Interpolation and Extrapolation

The lattice is a discrete function of the feature point, in order to solve characteristic displacements maps with depths not included inside the range of depths with data acquired it is necessary to extrapolate the x and y displacements.
It is also necessary to interpolate the x and y displacements for depths in between the depths with data acquired.The interpolation/extrapolation is done via a second order polynomial approximation:

$$z = \frac{1}{a_1 * x + a_2 * x} \quad (\text{where } a_1 \text{ and } a_2 \text{ are the polynomial coefficients})$$

$$z = \frac{1}{b_1 * x + b_2 * x} \quad (\text{where } b_1 \text{ and } b_2 \text{ are the polynomial coefficients})$$

The Figure 3.18 (z,x) show the approximation made to the x position of one feature point through the z position, i.e. the depth versus x estimation.
The line represents the approximation when only the 20 first values of the chain were take in to account, i.e. only the values until z=190 where used for the coefficients to obtain the polynomial coefficients.
The points in the chain with z>190 were acquired during the calibration movement but weren't used in the polynomial coefficients in order to evaluate the accuracy of extrapolation.

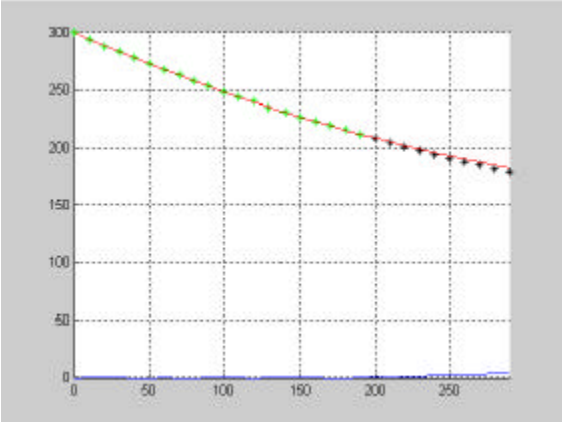


Figure 3.1 8

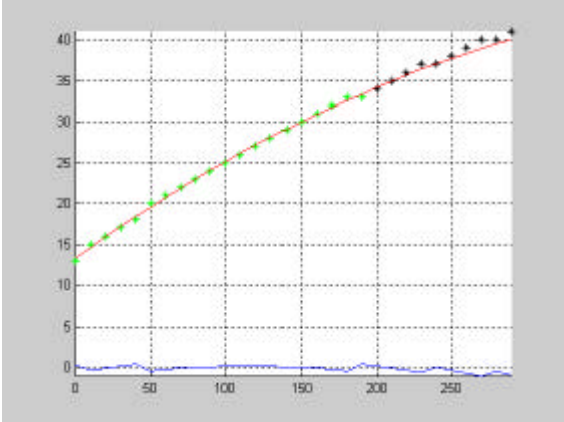


Figure 3.19

In the Figure 3.19 it is shown the approximation of the y position through the depth. The method and the amount of data used were the same as for the x position estimation through the depth.
The results where quite satisfactory, with a maximum error of 3 pixels for the several chains of feature point analyzed.
The error is represented by the bottom line in both figures.
The Figure 3.20 represents the approximation in x, y and z using the method previously described.

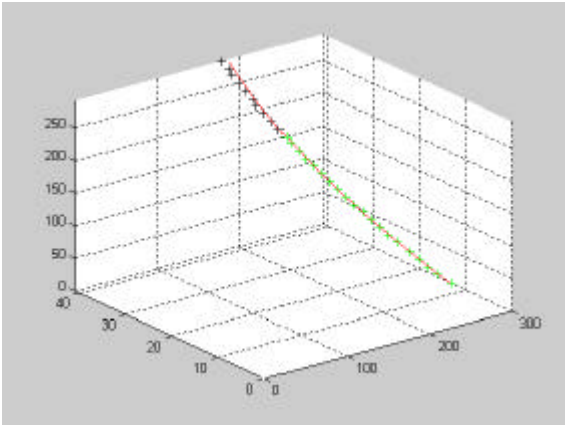


Figure 3.20

Until now it was only approached the problem of extrapolation and interpolation between depths, but also a interpolation and extrapolation must be performed for the same layer. Locations that don't belong to feature points in the same layer have to be interpolated or extrapolated.

3.4.7 3D Kinematic Estimation using Lattice

In Figure 3.21 and 3.22 it is shown in 3D one layer acquired. This layer corresponds to the 400mm plus a small offset distance not

known to the calibration board. Where the left camera was taken has reference and the z axis represents the x disparity.

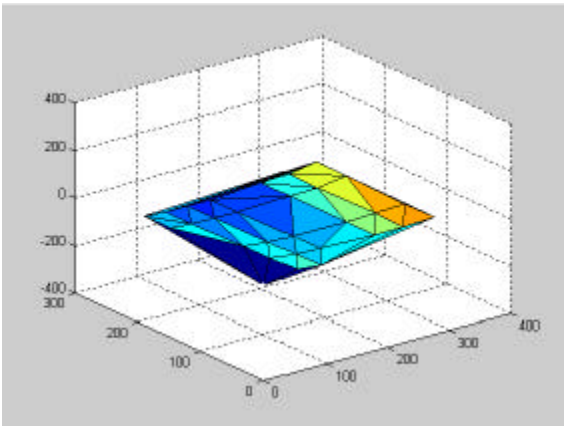


Figure 3.21

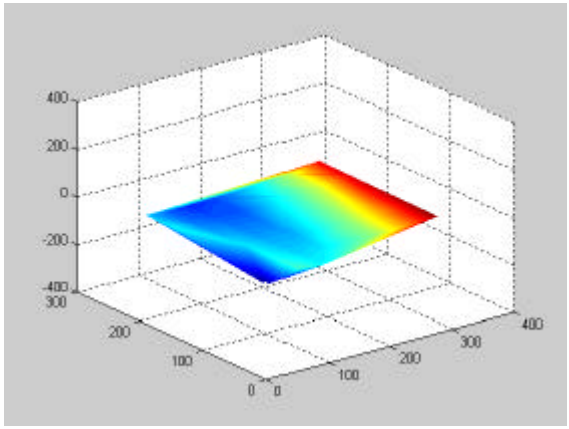


Figure 3.2 2

The Figure 3.23 shows the layer but now in 2D, the Figure 3.24 represent a lattice built on base of the Intrinsic parameter matrix and the transformation matrix between cameras.

An observation of the two graphics shows a consistency on the values acquired.

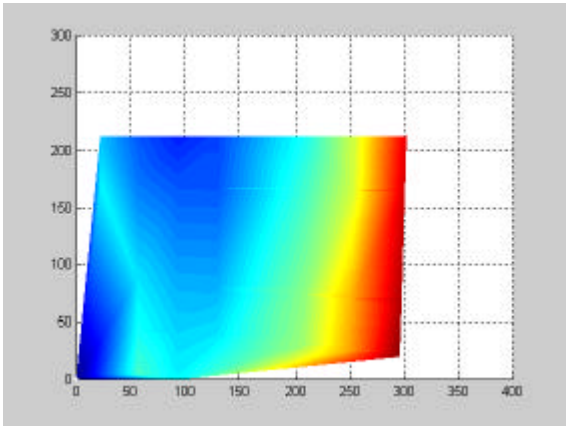


Figure 3.23

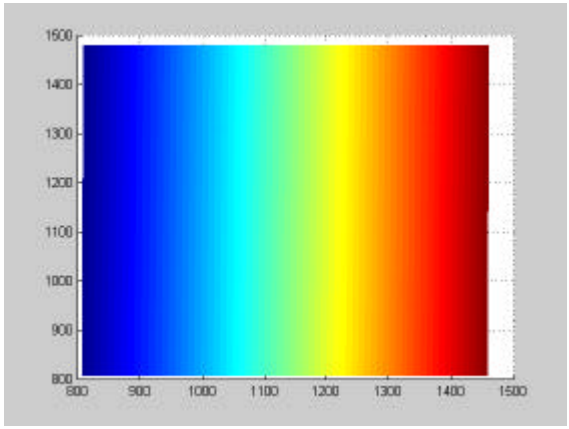


Figure 3.24

In Figure 3.25 and 3.26 it is shown in 3D one layer acquired. This layer corresponds to the 400mm plus a small offset distance not known to the calibration board. Where the left camera was taken has reference and the z axe represent the x disparity.

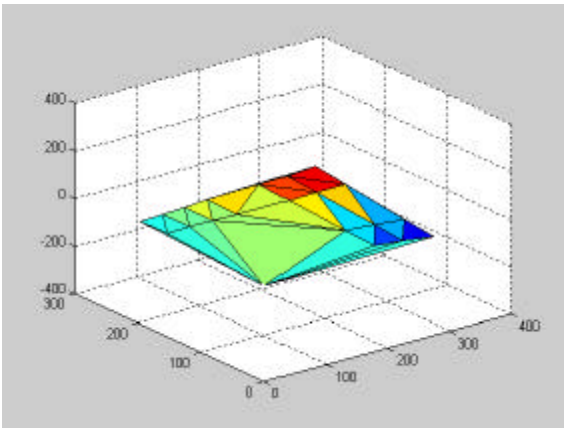


Figure 3.25

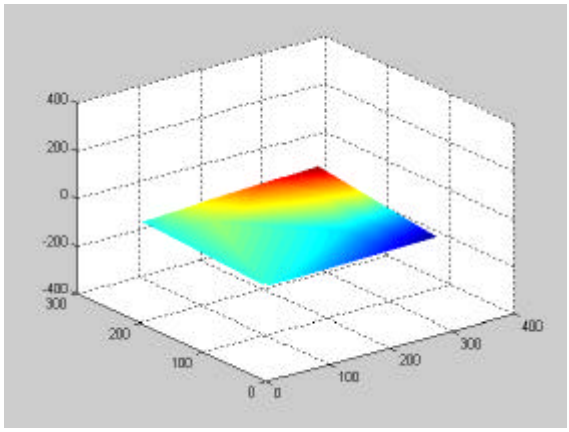


Figure 3.26

The Figure 3.27 shows the layer but now in 2D, the Figure 3.28 represent a lattice built on base of the Intrinsic parameter matrix and the transformation matrix between cameras.

An observation of the two graphics shows a consistency on the values acquired.

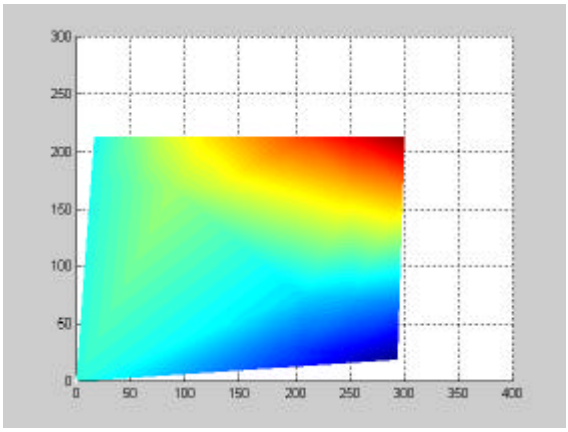


Figure 3.27

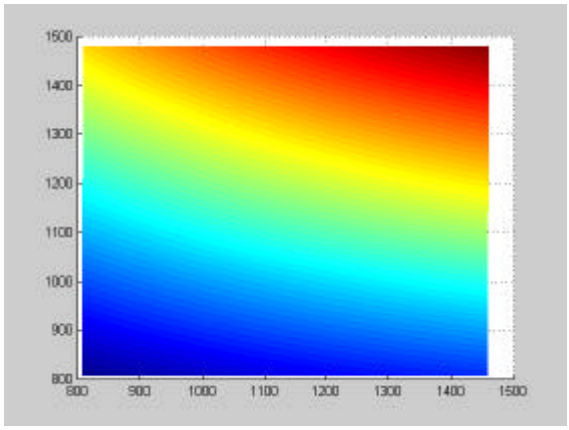


Figure 3.28

3.5 Concerns using Virtual Robot

The vision algorithms in the investigation are tested with a simulation of the robot IRB-6. The main feature of this robot is the image shown from the end-effector, that is a view of the plane perpendicular to the Z-axis of the gripper. The virtual image is a view of the camera mounted in the end-effector, representing the visual sensor that performs the image acquisition to feedback the robot controller.
The task is to process the image to center the object in the window.

The image is received on a PC running the Windows operating system by using socket inter-process communication. The image is then processed in visual C++.
The object is a cube with the colors blue, red and green in each face. The image is received on each face as three matrices(R, G, B), each one having a size of 200x200 elements and being approximately 1Mbytes.

After the image acquisition an image scan is done in order to find the cube centre. The scan starts on the first row and column and scans from the upper left corner to the right bottom corner. The image background is black and the cube colors are sharp. For example to find a blue color a scan is done in blue matrix searching for a blue component higher than 240 and in red and green component with value below 10. These reference values are chosen because it is a 8 bit color range. To find blue, the values should be R=0, G=0 and B=255.

The color mass center

$$mean(x, y) = \sum_{i=1}^{n} \frac{(x_i, y_i)}{n}$$

is calculated on regions with pixels of the same color. (x_i,y_i) are the coordinates of each pixel in the region. The color mass center is sent to the virtual robot.

3.6 3D Kinematic Estimation

Once the lattice is built its possible to use it to calculate relative distance between objects, and that's the goal and reason of all the efforts spent on building the database.
In our case the objects are one test object (TO) and the end-effector of the robot IRB-2000 (RO).
Both TO and RO have a cross draw in one of the faces.
The configuration looks like the one represented in Figure 3.29.

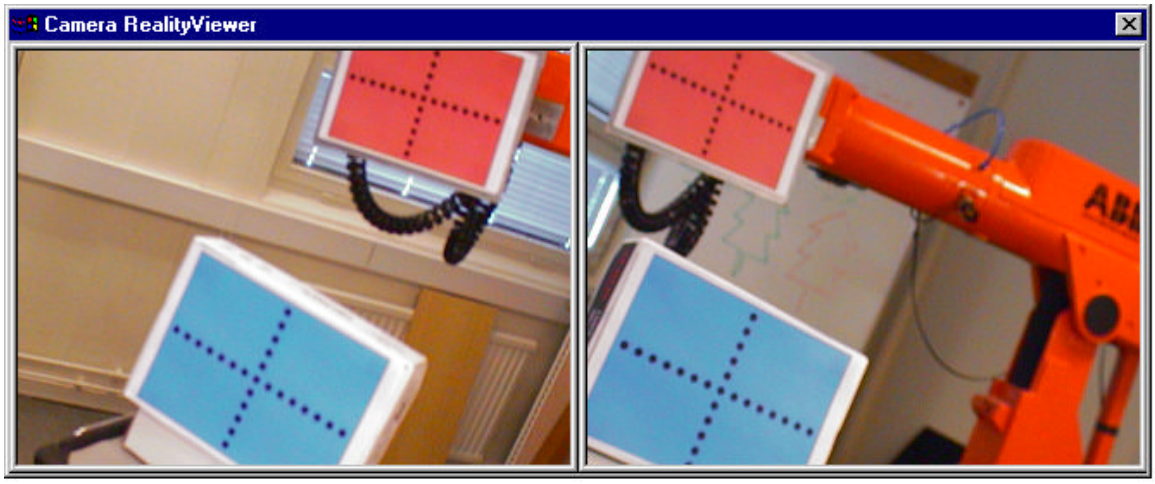


Figure 3.29

To easy recognize the TO and the RO each cross as different background color, in this case the TO as blue background color, and the RO as red background color.
The colors are placed in opposite sides in the chromaticity spectrum in order to better distinguish the TO from the RO.

The Line Scan algorithm was implemented to, at the same time perform the threshold and acquire the x, y position of the feature points, now it is applied two times one for the TO cross and another for the RO cross.

The threshold limits for the TO application were:

$$TOCross(x,y) = \begin{cases} 1, if & (R(x,y) < 200) \,\&\, (G(x,y) < 200) \,\&\, (B(x,y) > 180) \\ 0, if & otherwise \end{cases}$$

The threshold limits for the RO application were:

$$ROCross(x,y) = \begin{cases} 1, if & (R(x,y) > 200) \,\&\, (G(x,y) < 150) \,\&\, (B(x,y) < 150) \\ 0, if & otherwise \end{cases}$$

The resulted image is shown on Figure 3.30.

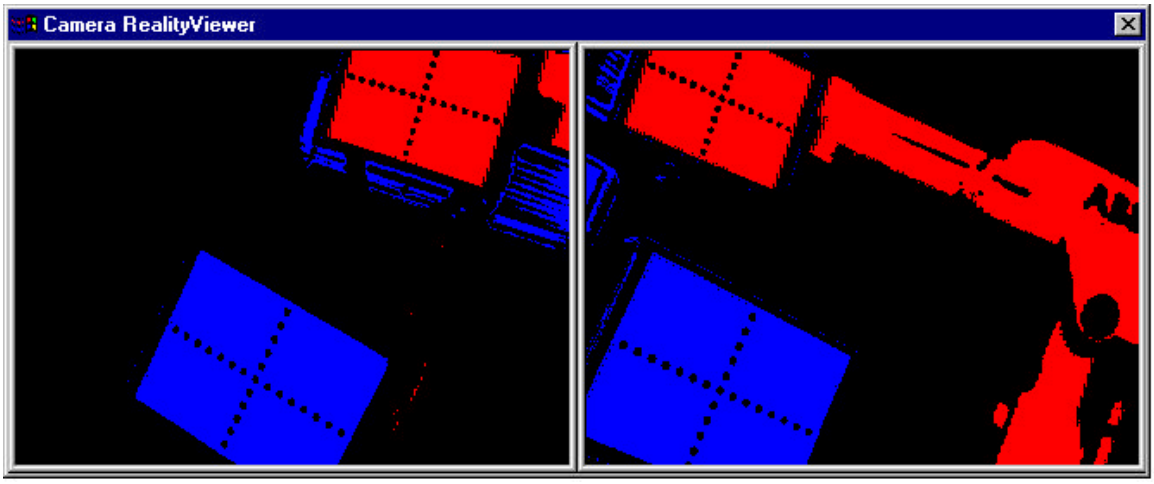


Figure 3.30

At this time there is two databases with the feature point from the TO and the RO, as displayed in Figure 3.31.

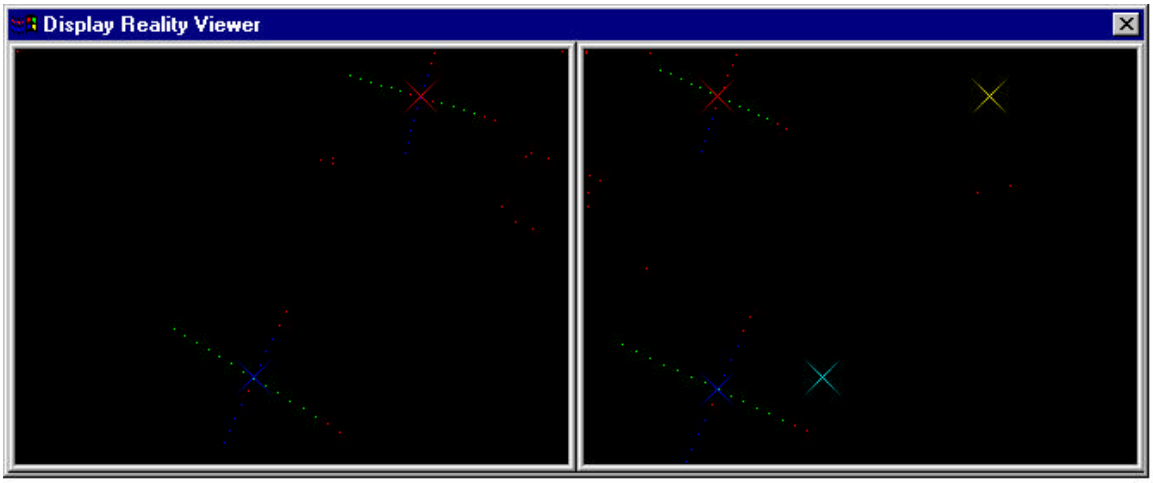


Figure 3.31

Now by applying the algorithm used in the plate, it is possible the get the center of both crosses in both images.
In Figure 3.32 the cross centre achieved is displayed, and on the right window it is draw the cross position of the left camera, this was done to have a better visual idea of the displacement of one cross to the correspondent cross in the other camera.

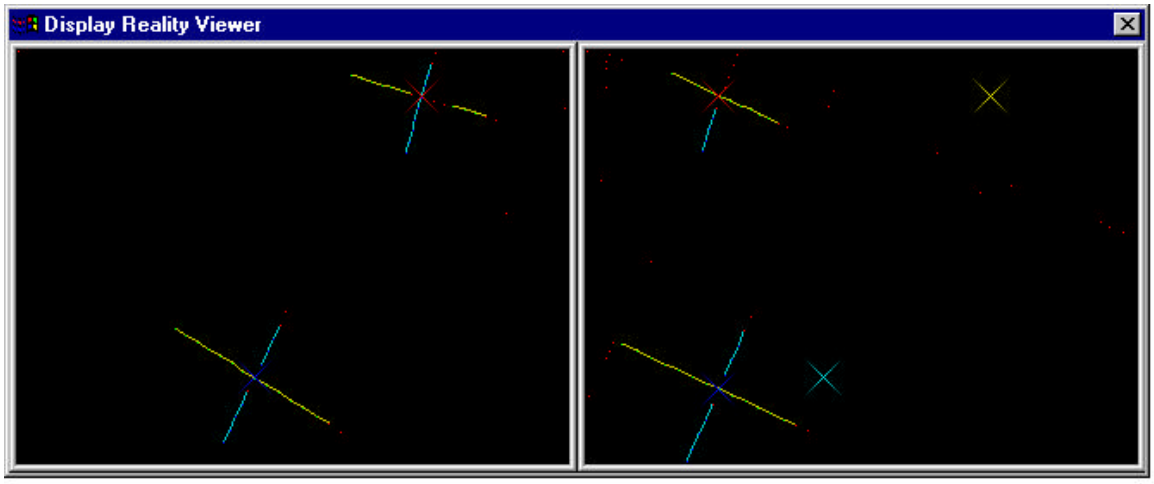


Figure 3.32

The relative distance can now be solved using the lattice information.

3.7 Robot Control using Visual Feedback

The control of the IRB-6 and IRB-2000 was done in two main fields, the kinematics control and the controllers implemented.

The kinematics control was done mainly in inverse kinematics. This means that is given the end point of the structure, in coordinates, to calculate the joint angles to achieve that end point.

There were implemented different solutions for the controllers to find a good relation between fast response, to reach the desired point and a good stability. The controllers tested were proportional and proportional with integral part.

The system architecture of the robot control visual feedback can be seen in Figure 3.33. This system is built in four main modules: robots, robot system, robot control and vision.

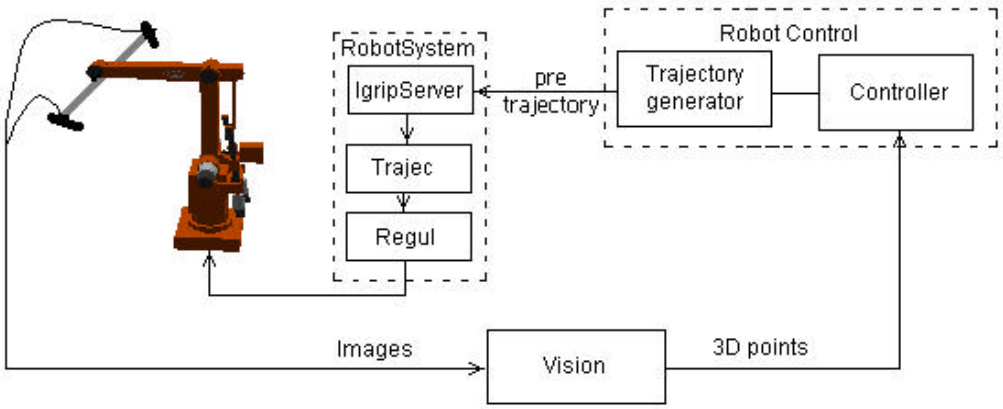


Figure 3.33

Robot control using Visual Feedback

The robot control module and the vision module that we built are connected with the robots and robot system module, forming the close loop that we will describe. These units receive, processes and send the control information that takes some time delays and errors. These are observed and measure to get the sources error.

The transfer information between the different modules takes some time. This time is not constant and depends of the network load, electrical disturbances and data amount.

The information processing takes some time too. In general depends of the amount of image, the way to process it, such as, if it is recursive or how many times is process it, the image area, the points extracted, and the different parameters of the analysis in the same time.
Another delay is added when the image is printed on the screen that means time spent in on plotting to the desktop

The acquiring camera images delays the system because the cameras are not synchronised and then the data acquiring can have different timings.

The three main kinds of time delays observed on the system are:
- Communication delay in network
- Computation delay in vision and control processing
- Acquiring images

The errors observed in the system are the following:
- Soft triggering of cameras
- Assumption of position and images available at the same point in time

3.7.1 Control of Robot

The aim in this control is to make the movement of the robot fast, smooth and stable as possible in a way to realize the desired trajectory.

The feedback is provided from the cameras, so the inputs are x, y in image space coordinates and the outputs are the joint angles.
It is assumed that there is a perfect relation between the image Coordinates and the robot Coordinates. This means that one pixel correspond one 1 mm in the robot coordinates.

P Control
The implemented control in this research is a proportional controller and it works in discrete-time.

The proportional controller in continuous-time is described by the equation

$$u(t_k) = K * e(t_k)$$

but the inputs x and y are in discrete time:

$$x_i(kh) = K * \Delta_x(kh)$$
$$y_i(kh) = K * \Delta_y(kh)$$

K is the gain or the proportional gain of the controller.
The proportional controller is a function of the error. That is the difference between the reference position and the actual position. This error is obtained in image coordinates.

$$\Delta_x = (X_{ref} - x)$$
$$\Delta_y = (Y_{ref} - y)$$

PI control
In order to keep $x_i = 0$ and $y_i = 0$ either a proportional (P) or a proportional-integral (PI) controller can be used. The integral part is used to eliminate the stationary error. The PI controller introduces an integrating effect. The integral term is given by

$$I(t) = \frac{K}{Ti}\int_0^t e(s)ds$$

it follows that

$$\frac{dI}{dt} = \frac{K}{Ti} e$$

using the forward differences gives

$$\frac{I(t_{k+1}) - I(t_k)}{h} = \frac{kh}{T_i} e(t_k)$$

This leads to the following recursive equation for the integral term

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

and the PI control will be

$$u(t_k) = K(e(t_k) + \frac{K}{T_i} e(t_k)$$

Parameters T_i has the dimension time called the integral time.

PID control

Using PID control is possible to improve the damping of an oscillatory system. The PID is implemented in the following way. It is add in PI control a D term. In D term is used a backward differences, with the following equation

$$\frac{Td}{N} \frac{D(t_t) - D(t_{k-1})}{h} + D(t_k) = -KTd \frac{y(t_k) - y(t_{k-1})}{h}$$

This can be rewritten as

$$D(t_k) = \frac{Td}{T_d + Nh} D(t_{k-1}) - \frac{KT_d N}{T_d + Nh} y((t_k) - y(t_{k-1}))$$

An **integral windup** [C.C.S.] is an effect that takes place when a PI must compensate for a longer time for an error, so it was implemented a integral anti windup in the PI controller. If the control error is so large that the integral saturates the actuator, the feedback path will be broken because will remain saturated even if the process output changes, Figure 3.34.

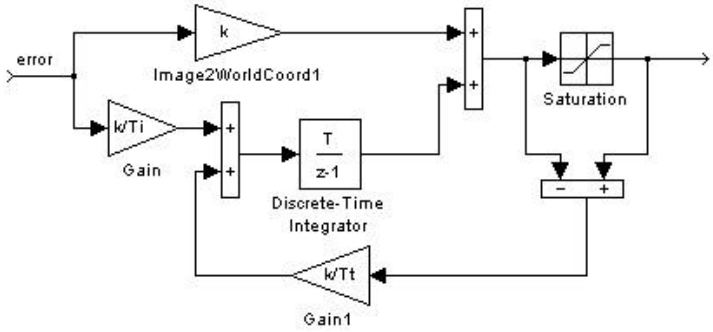


Figure 3.34

Controller PI with anti-windup

Another feature that can improve control is compensating the time delays. They are not constant so, we should predict the error in way to improve to be faster.
It was implemented three different ways to solve the time delays.

- One way to predict the error is set the zero error. So when the time delays is bigger than the sampling time it is set with zero. Figure 3.35(a).
- The second one is keep the error. So the error is constant until new data be received. Figure 3.35(b)
- The last one to predict the error is to set it with the two last errors received. Figure 3.35(c)

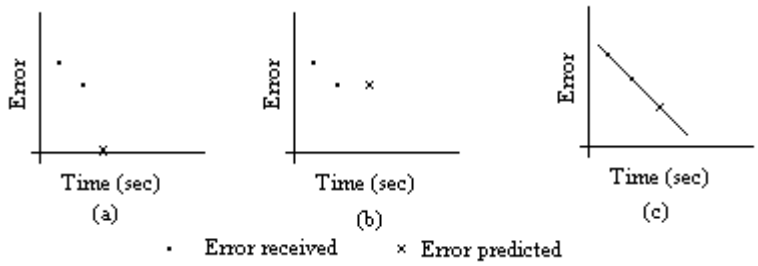


Figure 3.35

Different ways to predict the error

A good way to predict the error is to implement the Kalman filter to estimate the error. This filter was tested but it was difficult to find good parameters for the system. The result was not so good.

3.7.2 Control of Virtual Robot

The Virtual Robot is a useful tool to test implementations.

It runs on Java platform and communicates using matcomm. Matcomm is software that was developed in the Department of Automatic Control and is used as a tool to establish a communication line between processes.

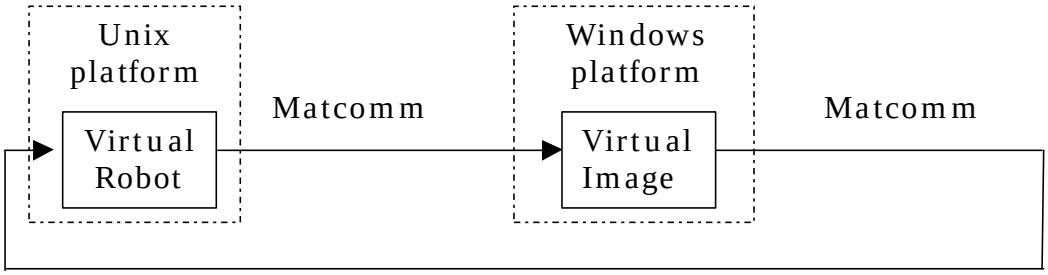


Figure 3.36

Communication and Feedback control of virtual robot

A flow chart for the virtual robot feedback loop is showed in Figure 3.36. The loop starts in the virtual robot and it sends images all the time. This image is a view from the end-effector camera.

The image is processed in Visual C++ in the Windows environment, where it is received and processed. This function is explained in more detail in the Virtual Java Robot Image Processing section. It returns the Coordinates X and Y to give the path that the robot should move for.

The Cartesian-space coordinates have to be converted to joint angles by application of the IRB-6 inverse kinematics giving the position and orientation of the end-effector of the robot. After that the joint values are sent to the robot to close the feedback loop.
The user can interact with the robot by dragging the cube to another position and which activates the control system and its feedback .

4 PROTOTYPE

Two platforms are used in the investigation, a Windows NT platform and a Solaris platform.

The reason for choosing Windows is that the Fire-i API was only available for this platform. Windows has a large number of hardware peripherals available (with supported drivers), particularly for frame grabbers (essential for real-time vision applications).
Windows also provides source software development tools that aren't available on Linux.
Some reasons for choosing Windows NT over the more popular Windows 98 are: Windows NT has a better architecture for real-time applications than Windows 98; Windows NT is more robust than Windows 98.

The PC used was an Intel Pentium III 450MHz with 128Mb RAM memory. The Pentium III includes integer and floating point Single Instruction Multiple Data (SIMD) instructions, which can greatly increase the speed of computer vision algorithms.
The code was developed in **Microsoft Visual C++ 6.0 Enterprise Edition.**
Matlab 6.0 was running in the Sun Solaris 8 platform.

4.1 System Architecture with Dataflow Diagram

The vision system for the two experiments is shown in Figure 4.1. The dataflow shows the different modules implemented. They are the ABB robots, the vision and the robot control for each experiment.

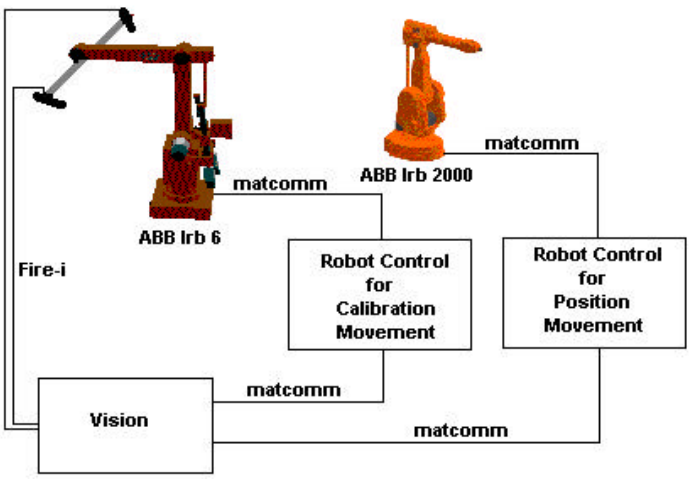


Figure 4.1

Dataflow the system

The **robots** and the **robot system** are described in detail in section 3.2. The robot system executes the pre-calculate trajectory sent by the Robot Control module.

The **vision** module processes the image received from the stereo rig and returns a feature point in image coordinates. This point can be sent to the IRB 6 to make a calibration movement (section 4.5) or can be sent to the IRB 2000 to make the position movement (section 4.6). The camera communication with vision module is made through a Fire-Wire. Network connection communication with Robot control module is made through matcomm. The vision module is further described in section 4.4.

The **robot control** module receives a feature point in image coordinates in order to control and generate the trajectory that is sent in joint space through matcomm to the robot. For each experiment was built a robot control module.

The system forms a close loop which start with the cameras that sends the feature points to the Robot Control module. Then, a trajectory is built to move the robot to the desired position.

4.2 External API (robot, matcomm, camera)

Robot

Functions used for ABB IRB-6

irb6boot
- It is a shell scrip to boot the robot system for IRB-6.

make action
- Open the connection with the robot IRB-6.

Functions used in Matlab for ABB IRB-6

Sync
- Function to send the robot to the home position.

[joints]=CartToJoints([x y z j4 j5])
- Inverse kinematics, that converts Cartesian coordinates to Joints values. The inputs are the Cartesian coordinates and return the joint -values.

[x y z j4 j5]=JointsToCart([j1 j2 j3 j4 j5])
- Forward kinematics, that converts Joints values to Cartesian coordinates. The inputs are the joint-values and return the Cartesian coordinates.

Functions used for ABB IRB-2000:

irb2000boot
- it is a shell scrip to boot the robot system for IRB-2000.

make action
- open the connection with the robot IRB-6.

mjdeg([j1 j2 j3 j4 j5 j6 time socket])
- Used to send joint values to IRB-2000.

Joint=invkin2400(toolToBase, front, noflip, tool_length)
- Inverse kinematics, that converts Joints values to Cartesian coordinates. The inputs are: joint space, front, noflip and tool length. It returns the Cartesian coordinates.

[ToolToBaseFrame]=forward2400(joints, tool_length)
- Forward kinematics, that converts Joints values to Cartesian coordinates. The inputs are the joint-values and tool length. It returns the Cartesian coordinates.

Matcomm

The Matcomm functions used in the Solaris system are the following:

[ID] = matcomm('server', PROCESS)
- To create a server called by process that listens for incoming requests.

[ID] = matcomm('open', TARGET, PROCESS)
- To open a communication line with process. Target is the host to connect.

[b] = matcomm('more', ID)
- To determine if data is available from the communication line in ID process. Return one or zero if receive or not data respectively.

matcomm(id, data)
- Writes a packet to the id system.

[data] = matcomm(id)
- Reads a packet from the id system.

[ID] = matcomm('close', ID)
- To explicitly close communications line with ID.

The Matcomm functions used in the Windows system are the following:

void M atCommWin32Init();
- Initializes the WINSOCK2 library and it must be called before using any Matcomm functions in WIN32.

MatCommLine *MatCommOpen(char *host, char *process, int timeout);
- Opens a communication line to the selected 'process' in 'host'. The call terminates after 'timeout' seconds if 'timeout' greater than 0, even if a connection has not been made.

int MatCommReadDouble(MatCommLine *line, double *data, int rows, int cols);

- If packet contains reals reads them into data and returns true otherwise it skips one datapacket and returns false; rows and cols contains the dimensions of the data matrix.

void MatCommWriteDouble(MatCommLine *line, double *data, int rows, int cols);

- Writes a data packet type double to the target system; rows and cols contains the dimensions of the data matrix.

void MatCommClose(MatCommLine *line);

- Closes an opened 'line'.

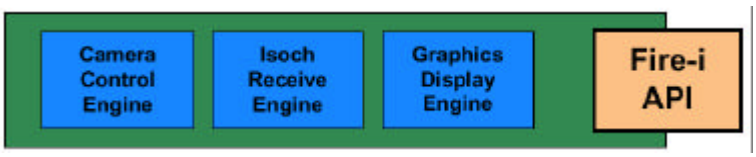
Camera

The Fire-i API can be divided into tree different parts:

- Camera Initialization and Control Engine
- Isochronous Receive Engine
- Display Engine

The three parts operate independently from each other. The isochronous Receive Engine uses a FIREi_CAMERA_STARTUP_INFO structure to perform a isochronous receive. These structures are created and exported by the Camera Control Engine. The same goes for the FIREi_CAMERA_FRAME structure, which is created by the Isochronous Receive Engine and then used by the Display Engine.

The scheme below describes the structure of the Fire-i API.



The Fire-I API functions are need in the following sequence:

Functions used to initialize the vision system.

FIREi_STATUS FiInitialize(void);

- Initializes the Fire-i dll for the caller.

FIREi_STATUS FiBusReset(IN ULONG uAdapterNumber,
IN BOOLEAN bShortReset);

- Issues a software initiated long or short bus reset.

FIREi_STATUS FiLocateCameras(
IN OUT FIREI_CAMERA_GUID *pCameraGuidArray,IN ULONG uFlags,
IN OUT ULONG *uNumberOfCameras);

- Locates Digital Cameras on the local bus.

FIREi_STATUS FiOpenCameraHandle(
OUT PFIREi_CAMERA_HANDLE pCamerahandle,
IN PFIREI_CAMERA_GUID pCameraGuid);

- Opens a handle to one of the cameras on the local bus.

FIREi_STATUS FiInitializeDisplay();
- General initialization of all the display objects.

FIREi_STATUS FiCreateDisplayWindow(IN HWND hWnd,
OUT PFIREi_DISPLAY_HANDLE pDisplayHandle,IN PFIREi_CAMERA_STARTUP_INFO pStartupInfo,IN POINT DisplayPosition,
IN OUT BOOL* bOverlay);
- Creates a FIREi Camera Display Window and attaches it to a user provided GDI window.

FIREi_STATUS FiShowDisplayWindow(
IN FIREi_DISPLAY_HANDLE hDisplay);
- Shows a FIREi Camera Display Window.

FIREi_STATUS FiCreateIsochReceiveEngine(
OUT PFIREi_ISOCH_ENGINE_HANDLE phIsochEngine);
- Creates an Isochronous receive engine and returns a handle to it.

FIREi_STATUS FiStartIsochReceiveEngine(
IN FIREi_ISOCH_ENGINE_HANDLE hIsochEngine,
IN PFIREi_CAMERA_STARTUP_INFO StartupInfo,
IN ULONG uAdapterNumber);
- Starts the provided Isochronous Receive Engine and prepares it to receive frames described by the StartupInfo structure.

FIREi_STATUS FiResetCamera(
IN FIREi_CAMERA_HANDLE CameraHandle);
- Resets the camera.

FIREi_STATUS FiIsCameraRunning(
IN FIREi_CAMERA_HANDLE CameraHandle,
OUT BOOLEAN *pIsRunning);
- Stops the camera.

FIREi_STATUS FiStartCamera(
IN FIREi_CAMERA_HANDLE CameraHandle,
IN PFIREi_CAMERA_STARTUP_INFO pCameraStartupInfo);
- Starts the camera with the specified startup format.

FIREi_STATUS FiQueryCameraHandleEx(
IN FIREi_CAMERA_HANDLE CameraHandle,
IN ULONG Offset,
OUT PVOID Buffer);
- Returns the data contained in the specified offset of the camera.

FIREi_STATUS FiSetCameraRegister(
IN FIREi_CAMERA_HANDLE CameraHandle,
IN CONTROL_OID ControlOID,
IN OUT PVOID Buffer,
IN ULONG uBufferLength);
- Set various features of the camera to specific values.

The following functions are used every time a frame is retrieved and displayed.

FIREi_STATUS FiGetNextCompleteFrame(

OUT PFIREi_CAMERA_FRAME pCameraFrame,
IN FIREi_ISOCH_ENGINE_HANDLE hIsochEngine,
IN DWORD dwTimeOut);
- Retrieves a pointer to the next complete Frame transmitted by the camera.

FIREi_STATUS FiYuv2Rgb(
IN PFIREi_CAMERA_FRAME CameraFrame,
OUT BYTE *pRgbBuffer,
IN OUT DWORD *pdwBufferBytes)
- Convert a image from YUV to RGB representation.

FIREi_STATUS FiDisplayCameraFrame(
IN FIREi_DISPLAY_HANDLE hDisplay,
IN PFIREi_CAMERA_FRAME CameraFrame);
- Displays a camera frame in a specified window.

The followings functions are used to terminate the vision system.

FIREi_STATUS FiCloseDisplayWindow(
IN FIREi_DISPLAY_HANDLE hDisplay);
- Cleanup and destroys all objects created by FiCreateDisplayWindow.

FIREi_STATUS FiStopIsochReceiveEngine(
IN FIREi_ISOCH_ENGINE_HANDLE hIsochEngine);
- Stops the provided Isochcronous Receive Engine.

FIREi_STATUS FiDeleteIsochReceiveEngine(
IN FIREi_ISOCH_ENGINE_HANDLE hIsochEngine);
- Deletes an open IsochReceiveEngine and frees all the structures created by FiCreateIsochReceiveEngine.

FIREi_STATUS FiStopCamera(
IN FIREi_CAMERA_HANDLE CameraHandle);
- Stops the camera.

FIREi_STATUS FiCloseCameraHandle(
IN FIREi_CAMERA_HANDLE CameraHandle);
- Closes the handle of a camera.

void FiTerminate(void);
- Terminates Fire-i support for the specified application.

These functions are used in the vision initialization phase and every time that a camera parameter is changed, in our case shutter and gain.

FIREi_STATUS FiQueryCameraHandleEx(
IN FIREi_CAMERA_HANDLE CameraHandle,
IN ULONG Offset,
OUT PVOID Buffer);
- Returns the data contained in the specified offset of the camera.

FIREi_STATUS FiSetCameraRegister(
IN FIREi_CAMERA_HANDLE CameraHandle,
IN CONTROL_OID ControlOID,
IN OUT PVOID Buffer,
IN ULONG uBufferLength);

- Set various features of the camera to specific values.

4.3 Kinematics Control (Simulink/Matlab)

The kinematics control of the IRB 6 and IRB 2000 robots are done in Matlab/Simulink environment that runs on Solaris platform.

The Simulink model runs with the fixed sampling time of 0.05 seconds.

The model has two subsystems, Figure A.1. One for moving the end-effector to a goal position, Figure A.2 and another to use in the control loop.

The calibration movement starts by positioning the stereo rig into an area so the cameras have a full vision of the board. The positioning movement starts by moving the IRB 2000 end-effector in the vision volume of the stereo rig mounted on IRB 6.

The corresponding Simulink blocks have the same structure for both movements. They only differ in internal functions, such as ReceivingPoints and Traj2Robot.

The **ReceivingPoints** function receives the points from the vision module (section 4.5) by matcomm and is describe in detail for both movements.

The **Traj2Robot2** receives the desired destination in Cartesian coordinates and returns a trajectory for the robot. It has the following steps:

- Receive the error in coordinates x, y and z.
- Update the trajectory with the last position.
- Convert Cartesians to joints.
- Perform the trajectory of the robot with the interpolate function.
- Send the trajectory by Matcomm to the robot.

The **Interpolate** function creates a trajectory from the last position parameters to the desired position. The trajectory is split into small pieces and the entire trajectory is performed in 0.1s.

4.3.1 Calibration Movement

The stereo rig is mounted on the IRB 6, which is used for the calibration movement.

The calibration consists of sequence of movements along the world space x, y and z-axis. In x and y, the purpose is to keep the cameras

centered on the board for the calibration to have a bigger area of analysis. The movement in z-axis is in intervals of 10 mm.
The calibration model is a board with several holes and a cross in the center, needed to create a reference on the board.
A problem found in this experiment was that of different orientations between the robot coordinate system and the cameras system. A first task was to align the system coordinates of systems, robot and stereo rig x and y coordinates.
This experiment starts to use a proportional controller with different gains in each coordinate, Figure A.3. The step response of the closed loop system is shown in Figure 4.2. The figure shows clearly that there is a steady state error. The error decreases when the controller gain is increased, but the system then becomes oscillatory.

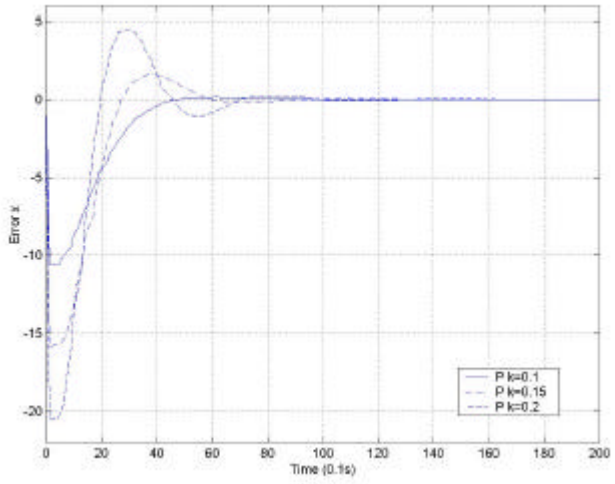


Figure 4.2

Proportional controller applied to x coordinate in Cartesian space. It can be seen how much the gain can influence the response.

To eliminate the stationary error a PI controller was implemented, see Figure A.4. This controller improves a bit the response. The gain is K=0.07 and the integral time is T_i=30 (Figure 4.3)

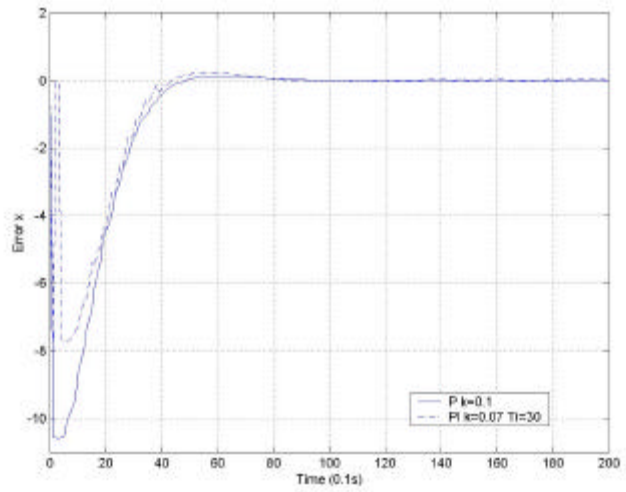


Figure 4.3

PI controller compared with P controller.

It was also tested a PID control, that improve the response (Figure A.5). The parameters for this control are k=0.1, T_I=30 and the derivative time is T_d=0.25. With PID is possible to see that it has a fast response but the trajectory is shaking a bit. The reason of these oscillations during the trajectory is because it is needed to predict some reference points. Figure 4.4.

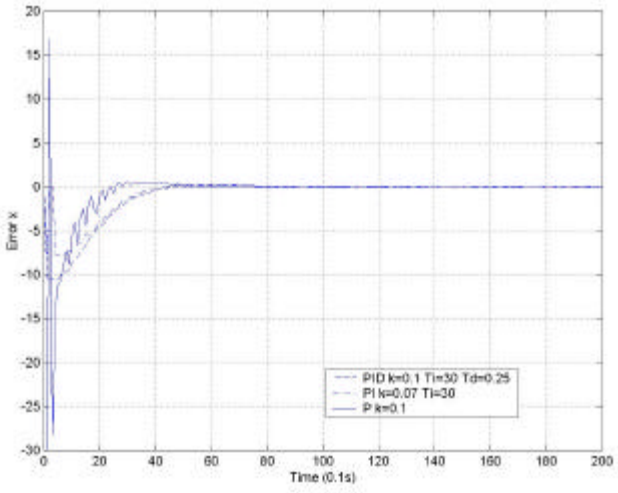


Figure 4.4

PID Controller.It is compared with different controllers such as P and PI

Another test to improve the control was a PI with anti-windup. The behavior of this control is unstable than only PI. Figure 4.5.

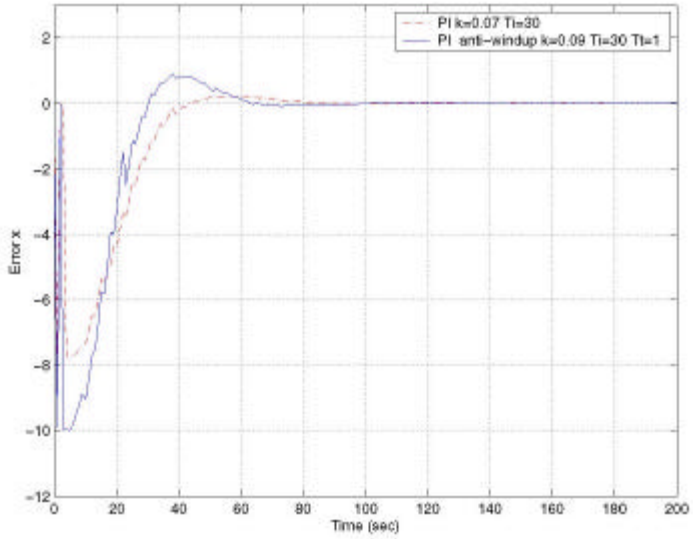


Figure 4.5

PI controller with anti windup

There was implemented also some improvements to control the time delays due to the vision module has a variation in a sampling and the control model has a fix sampling time. This means that some samplings are needed to predict to have a fast control. There were tested three different approaches that can be seen in Figure 4.6.

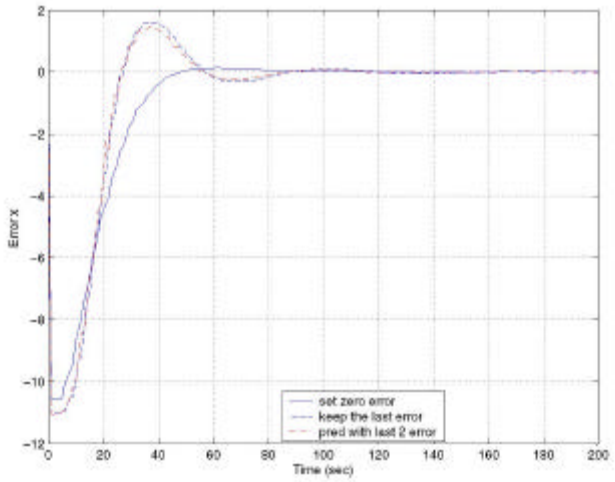


Figure 4.6

Different predictors

A good way to predict the next state is to use the Kalman filter with predictor but it was difficult to find good parameters for the system. The result was not so good.

In the calibration experiment the best controller is a PID to control the position for x and y coordinates.

4.3.2 Position Movement

The positioning movement is made with the two robots, IRB 6 and IRB 2000. The IRB 6 put the stereo rig in a pre-defined position to have a full vision of the two crosses. The IRB 2000 starts the control in start position pre-defined also.
This experiment has the same problem as the previous experiment, which is that orientation of the coordinate system of the IRB 2000 and the cameras. The angle of difference between the coordinate systems was measured $q = 46{,}09$ degrees to align the coordinates. This is made with a rotation matrix about the z-axis

$$R_Z(q) = \begin{bmatrix} \cos(q) & -\sin(q) & 0 & 0 \\ \sin(q) & \cos(q) & 0 & 0 \\ o & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now coordinate axis of the robot and the cameras are aligned. For this experiment is applied the same control as in calibration experiment, but due to the restrictions in time it was not tested a PID to see if it works better than a proportional control. The behavior is shown in Figure 4.7.

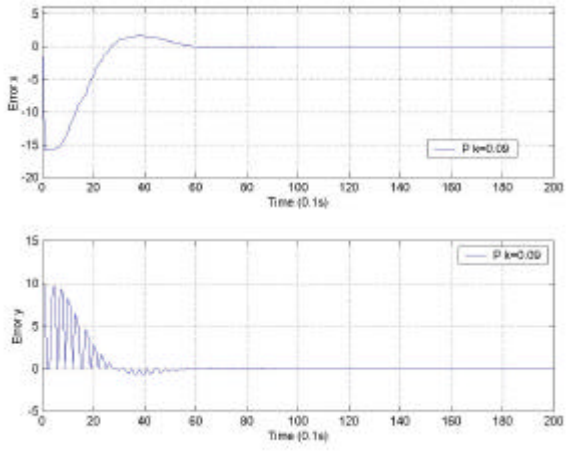


Figure 4.7

Behavior of P controller applied in x and y coordinates.

The y robot coordinate represents the control in depth between the object and the end-effector. The error y is distance between objects.
The distance between the cameras and the depth of the object is calculated using displacement

$$CalcDiffZ = ((X_{irb2000Cam1} - X_{irb2000Cam2}) - (X_{objectCam1} - X_{objectCam2}))$$

These two displacements are subtracted from each other, which equals the difference in depth between the object and the robot.

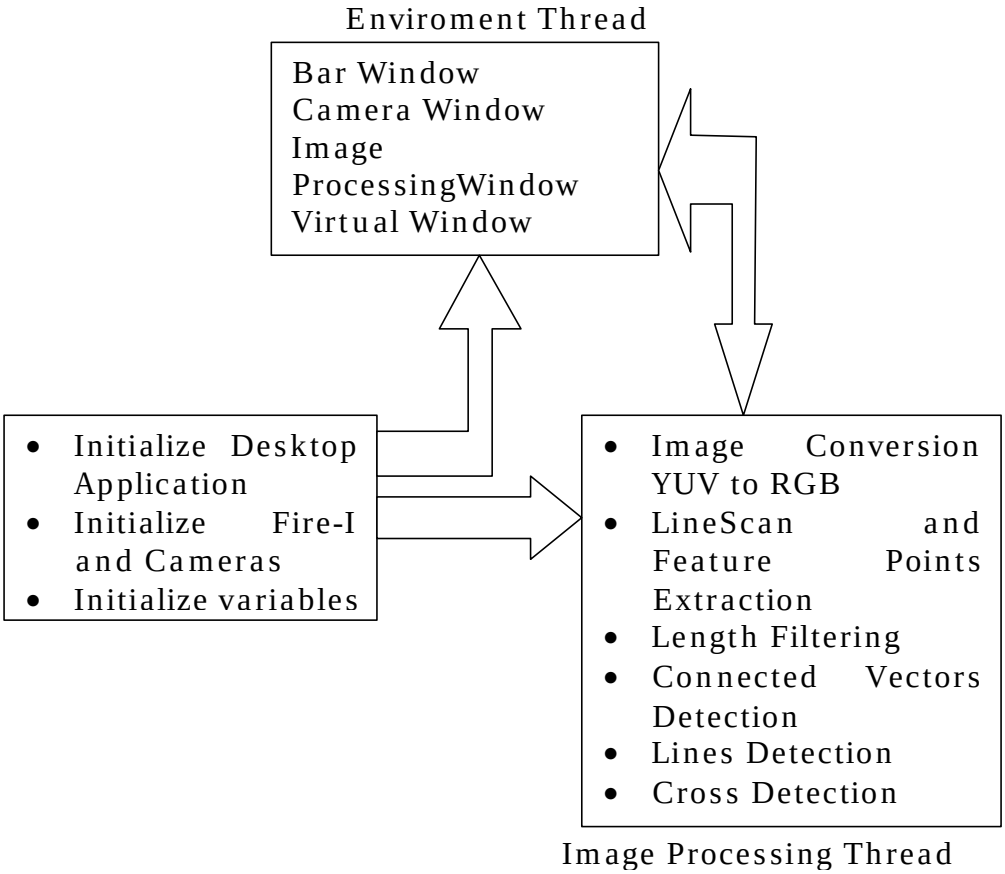
If CalcDiffZ is negative, it means that the object is closer to the stereo rig than the IRB 2000 tool. The CalcDiffZ is applied to the y_{robot} coordinate.
Then, after calculation the error is inserted into a proportional controller and sent to the Traj2robotIRB2000 function, that downloads the trajectory to the IRB 2000.

4.4 Vision Processing (Visual C++)

The Vision Processing prototyping is divided in to three main areas:

- The camera communication and image retrieved.
- The communication through Matcomm with the Sun workstation.
- The interface communication and visualization.

The dataflow sequence starts by initializing all variables. In particular the state variables (VirState=0; CamState=0; DisState=0; CalibState=0; RedBlueState=0). It also allocates memory to perform the image processing algorithm.



- **VirState** is set to 1 if running the Virtual Robot Experiment.
- **CamState** is set to 1 if the cameras are being used.
- **DisState** is set to 1 if running the Image Processing.
- **CalibState** is set to 1 if sending the lattice to the Sun Workstation.
- **RedBlueState** is set to 1 if running the Positioning Experiment.
- **long*malloc_regionscan_buffer( )** and **long* malloc_regionscan_rstat( )** allocate memory to use in the line scan algorithm.

The Firei-i initialization, communications channels and camera structures were filled using **BOOL InitCameraHandleArray()** .

The image format for display in windows is defined using **BOOL CreateDIB().**

In **BOOL InitApplication(HINSTANCE hInstance)** the windows format was specified.

After this setup the Main window is built and a call back procedure waits for user interaction Figure 4.8, by **LRESULT CALLBACK WndDlgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)**.

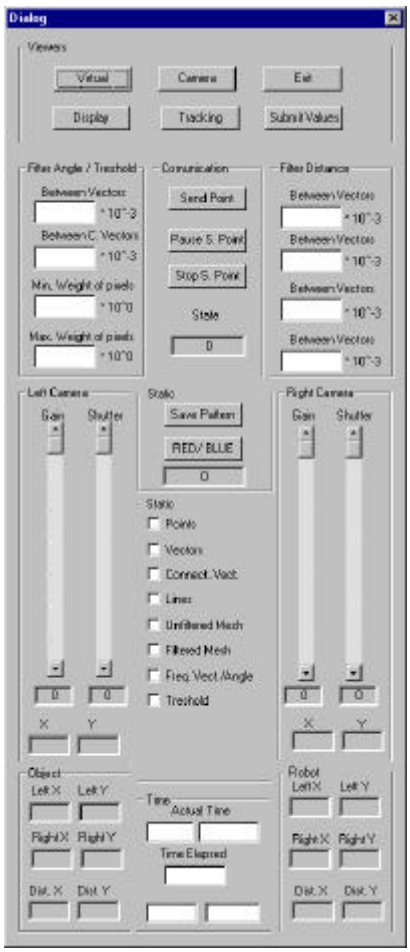


Figure 4.8

User Interface

The button “Virtual” initializes the communication with the Sun machine via matcomm using **void InitControlRobot(HWND m_hwndL,HWND m_hwndR)** and starts a thread for the image processing. The state variable **VirState** is set to 1. The **BOOL CreateWinVir()** function creates a window for the camera image.
The centre cube position is solved and sent back via matcomm to the Sun workstation using **void ControlRobot(void* a_pNoUse)**. When

the Virtual thread is finished the communication is terminated and the window closed using **void FreeRobotHandle()**.
The state variable **VirState** is set to 0.

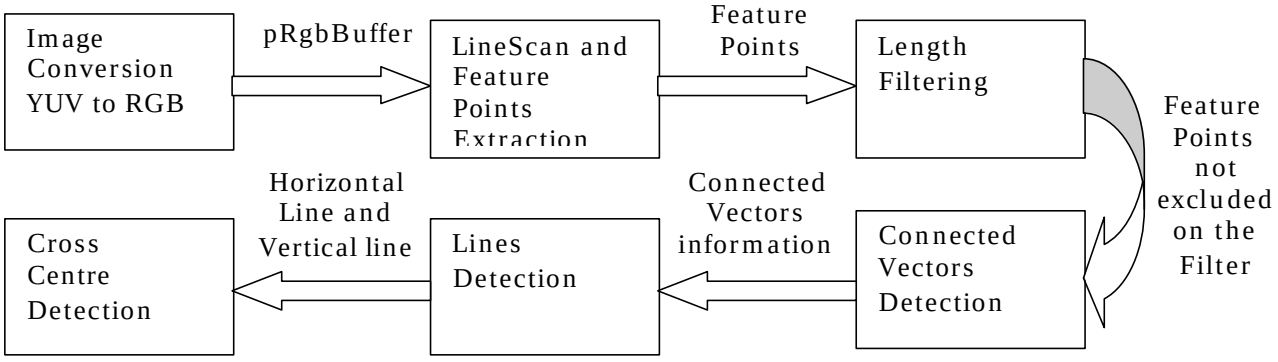
The button "Camera" creates a window for the camera image to be displayed in, using **BOOL CreateWinCam()**. The cameras are started using **BOOL InitDisplayWindowArray(HWND hwndL,HWND hwndR)**.
The state variable **CamState** is set to 1.
The gain and shutter registers for the cameras are set using **BOOL Var()**. The scroll bars are initialized using **BOOL OnDlgInitDialog(HWND a_hDlg)**.

Another thread is created using **void WaitAndDisplayFrame(void* a_pNoUse)** to receive the camera frames in real time.
If the camera window is closed the camera is stopped from transmitting images and the communication line is closed. The state variable **CamState** is then set to 0.

It is possible to change the shutter and gain in real time for both cameras, this is done using **void OnHScroll(HWND a_hDlg,WPARAM wParam, LPARAM lParam)**, which is call by the call back function for the main window.

The button "Display" creates a window for the processed image to be displayed in, using **BOOL CreateWinDis()**. The state variable **DisState** is then set to 1.

The image and vision processing is done according to the next flow chart.



The box "Image Conversion YUV to RGB" calls the function **FiYuv2Rgb(&CameraFrame,pRgbBuffer,&pdwBufferBytes))**.
pRgbBuffer is passed to the next box.

The image is received through a fire-wire card and is stored in memory where it is retrieved by the use of a UBcore function to a buffer. There is an independent buffer for each camera image.

The image is stored in a buffer of continuous bytes. The format of the image is RGB with 24 bit color depth (8 bits Red, 8 bits Green and 8 bits Blue.
The color components for each pixel follow BGR distribution in the buffer, the first byte is the Blue component, the second is the Green component and the third is the Red component.
The image is composed by three bytes for each pixel and followed by another three bytes for the next adjacent pixel, and so on. The image pixels are ordered from left to right and top to bottom. The size of the image is 320 width and 240 high.

To get the color component for each pixel the following code was used:

```
for(I=0;I<320;I++)
        for(J=0;J<240;J++)
        {
                count = ( 320*(239-J)+I )*3;
                b=*(pRgbBuffer+count+0);
                g=*(pRgbBuffer+count+1);
                r=*(pRgbBuffer+count+2);
        }
```

The box ”LineScan and Feature Points Extraction” calls the function **void regionscanBlack (unsigned char* image, long* buffer, long* rstat, long* rstatlen)** which is a line scan on black color segment detection. The function is called by **void DrawImage(IN BYTE *pRgbBuffer)** where a filtering on the color mass is done (i.e. filter on number of pixels included ion the color segment). The feature points are sent to the next box.

The box ”Length Filtering” use **void CalculateClosestNeighbour(BYTE *pRgbBuffer)** to find the closest neighbour feature point. The function **void FilterDist(BYTE* pRgbBuffer)** use the closest neighbour feature point to compute the length of vectors between points and perform a filtering on the vector length size. The feature points not excluded are passed to the next box.

The box ”Connected Vectors Detection” computes a filtering on similar vectors via the function **void CalculateVectors(BYTE* pRgbBuffer)**. The information in the connected vectors (length, angle,feature points included, start, end) is sent to the next box.

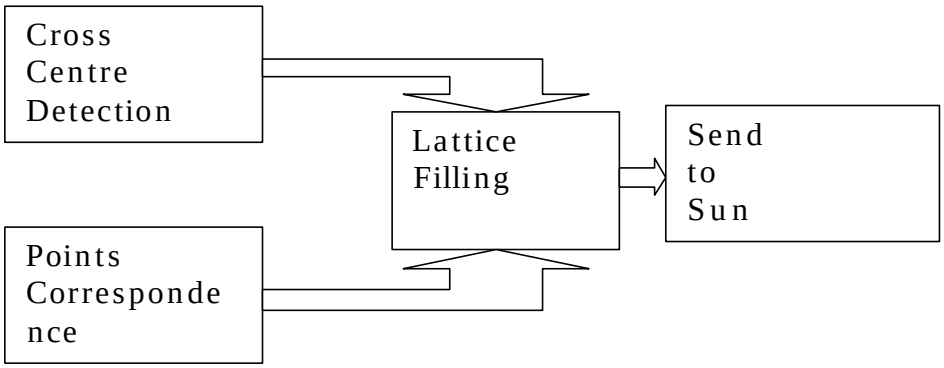
The box “Lines Detection” clusters the connected vectors received from the previous box into a horizontal and vertical line using **void CalculateLinesSegments(BYTE* pRgbBuffer)**. The horizontal line

and vertical line with their associated information, which is the connected vectors is sent to the next box.

The last box "Cross Centre Detection" uses the function **void LineCam(BYTE* pRgbBuffer,double* a,double* b,double* c,double* d,double* Xint,double* Yint)** to compute the cross center. This is done using the least square line method to fit the line in the feature points included in connected vectors clustered to the line.

If the button "Send Point " was pressed the cross centre is sent to the Sun workstation. The function **void SendPointInit()** is called from the main window callback function and the state variable **SendPoint** is set to 1. This enables the points to be sent via Matcomm by the function **void SendPoint()**. The communication is done inside the image processing thread.
The button "Pause S. Point" pauses the communication, the button "Stop S. Point" stops the communication and closes the matcomm line.

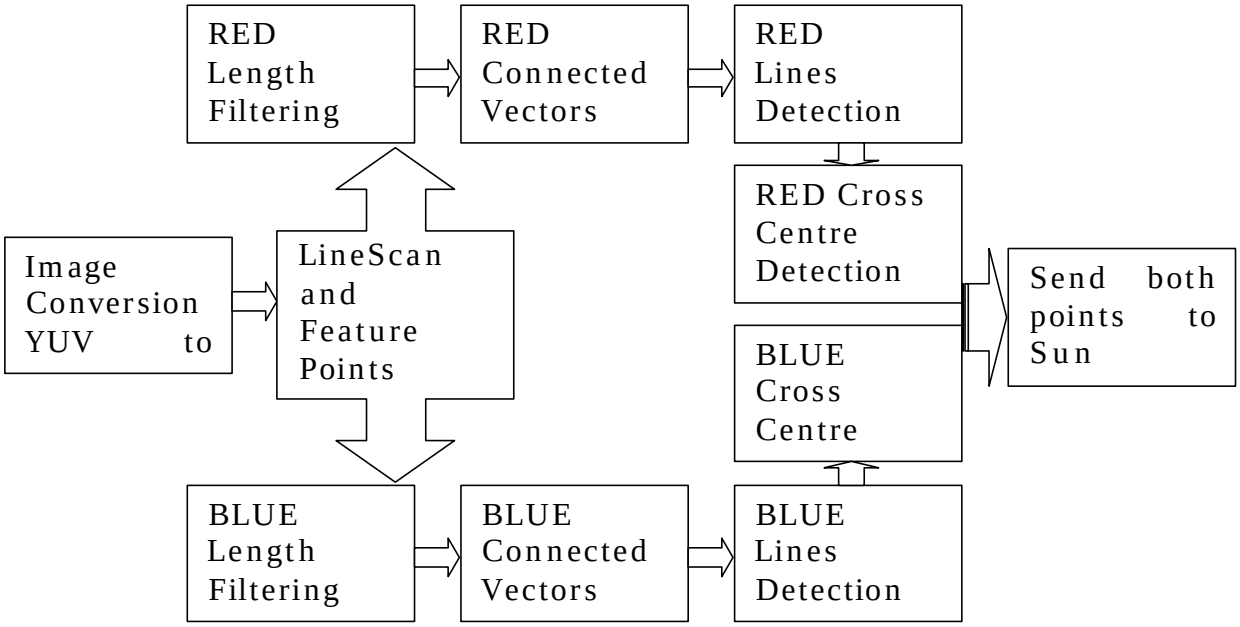
The button "Save Pattern" sets the state variable **PatternState** to 1.
The following data flow then is performed.



The box "Cross Center Detection" corresponds to the last box in the previous figure if at the end there is no cross found the filling of the lattice is not possible.
The box "Points Correspondence Recognition" solves the correspondence problem, and sends the result to the "Lattice Filling" box. This box groups the detected points into vertical lines. The two previous algorithms are performed in **void GetTableLines(BYTE *pRgbBuffer,double a,double b,double c,double d)**.
If the **SendState** and **PatternState** variables are set to 1 then the lattice is sent via Matcomm to the Sun workstation, using the function **void SendPoint()**.

The button "RED/BLUE" is used to alternate from the calibration movement image processing to the positioning movement image

processing. This means that instead of making a black line scan now it is done two line scans, one for Red and other for Blue.
The data flow is the same as for the calibration movement image processing after the line scan, but now in two separated databases.



In the box ” LineScan and Feature Points Extraction RED/BLUE” it was used the function void **regionscanBlue (unsigned char* image, long* buffer, long* rstat, long* rstatlen)** which is a line scan on blue detection. It was used the function **void regionscanRed (unsigned char* image, long* buffer, long* rstat, long* rstatlen)** which is a line scan on red detection. The previous functions are called by **void DrawImage(IN BYTE *pRgbBuffer)** where a filtering on the color mass is done. The feature points are sent to the next boxes, red feature points to the upper box and the blue feature points to the bottom box. Then cross centre is computed and if the button “Send Point ” was pressed the crosses centre is sent to the Sun workstation.

The check boxes are used to display are used to display in the image processing in the image processing window.
The upper edit boxes are used to change the parameters on-line of the filters.
The “Submit Values” is used to apply the parameters change to the filters.
The bottom left edit boxes are used to display the cross centre coordinates in both cameras and the (x,y) displacement of the object crosses.
The bottom right edit boxes are used to display the cross centre coordinates in both cameras and the (x,y) displacement of the Robot crosses.
The centre edit boxes are used to display the time profile.

If the button “Exit” is pressed all the windows are closed, communications stopped and the handles for the cameras released in the function **BOOL FreeCameraHandleArray()**.

5 RESULTS AND CONCLUSION

5.1 Performance Estimation and Code Profiling

The global system is working at 11 Hz when the positioning movement is performed and 15 Hz when the calibration movement is performed. This difference is due to the fact that in the positioning movement the cross recognition algorithm runs twice.

Calibration Movement
The 15Hz frequency means 70 milliseconds of time span. When a profiling is done it shows that the Line Scan algorithm consumes 15 milliseconds, the interface display and communication takes 30 milliseconds and the rest of the algorithm consumes 25 milliseconds.

Positioning Movement
Then 11Hz frequency means 90 milliseconds of time span. When a profiling is done it shows that the Line Scan algorithm consumes 15 milliseconds, which actually means 30 milliseconds because it runs twice, the interface display and communication takes 30 milliseconds and the rest of the algorithm consumes 30 milliseconds.

These timings weren't fixed, they depend on the amount of information to be processed, but the times shown above are the maximum obtained in the experiments.

The cameras work at 30 Hz and are software triggered. This means 33 milliseconds between new images and some time difference between cameras image acquisition.

The time consumed in the communication and in the control loop are negligible.

5.2 Robustness

The system proved to be robust when used in an environment with stable light conditions. The cross recognition algorithm is very robust, i.e. with the cross centre outside the image, it is estimated their coordinates.
There is good robustness to image noise.
Even with a time delay between samples variable, the control system has a good response.
When there is a fast movement of the cross, the image processing algorithm still tracks the cross centre and the control have a sharp and fast response.

When it is performed the z movement in the calibration experiment, even with the camera far away from cross the system proof to be robust.

5.3 Errors

The positing experiment has an error of 2 centimeters in the depth position and the cross centering is precise. In the calibration movement the position above the cross centre was precise.
The main cause of the errors were:

- Cameras Software Triggering, different point in time for each camera.
- Assumption of position and images available at the same point in time.
- Handling the time delays.
- Not using the lattice information (assumption that the disparity at a certain depth is the same for all the image area)

5.4 Conclusion

In this investigation we have presented the current drawbacks of image-based visual servoing and described a new method to cope these drawbacks.
The purpose of this investigation was to build a lattice that could accurately represent characteristic depth disparities and to use it in the positioning movement. This has not completely been achieved. The lattice was obtained but it wasn't used on the positioning movement. We have a system that includes a robot system, a vision system, a cross recognition algorithm and a lattice acquisition algorithm.
The lattice acquisition was achieved with a good accuracy. The positioning movement was made relying only on the disparity difference among the crosses.
The vision system and the image processing algorithms were totally built during this investigation, which proved to be very time consuming. Since Microsoft Visual C++, cameras handle and Fire-i grabber cards were tools for us more time was spent in learning how to use them.
The results where satisfactory and system proved to be robust. The errors due to time delays were the main system bottleneck.

5.5 Future Research

Future work in this investigation should focus the improvement of the time performance in image processing functions.
Also should be improved the control loop, since the scheme directly uses visual feedback as an input of the control law without any

supplementary estimation step. So path planning in the image should be a future work.
The system could be greatly improved by overcame the soft triggering of the cameras.
Although the system isn't totally developed it is a very good base work for future investigations.

REFERENCES

- Klas Nilsson. " Industrial Robot Programming". Department of Automatic control, Lund Institute of Technology, May 1996.
- Emanuele Trucco and Alessandro Verri. "Introductory Techniques for 3-D computer Vision". Prentice Hall, NJ07458.
- Jorge Batista. "Sistemas de Visao Activa: Comportamentos e Calibracao". Doctoral Thesis, Department of Electrical Engineering, Coimbra University, 1999.
- Urbano Nunes. " Controlo de Robos com Realimentacao Sensorial no Espaco Tarefa". Doctoral Thesis, Department of Electrical Engineering, Coimbra University, 1995.
- Johan Nilsson. "Real-Time Control Systems with Delays". Department of Automatic control, Lund Institute of Technology, January1998.
- Gustaf Olsson and Gianguido Piani. "Computer System for Automatic and Control". Department of industrial Electrical Engineering and Automation, Lund Institute of Technology, 1993.
- Karl J. Åström and Björn Wittenmark. "Computer-Controlled Systems". Prentice-Hall, ISBN: 0-13-314899-8.
- Karl-Erik Årzén. "Real-Time Control Systems". Department of Automatic control, Lund Institute of Technology, 2000.
- Johan Bengtsson and Anders Ahlstrand. "A robot Playing Scrabble Using Visual Feedback". ". Department of Automatic control, Lund Institute of Technology, April 1999.
- John J. Craig. "Introduction to Robotics". ISBN 0-201-09528-9.

A-SIMULINK MODELS

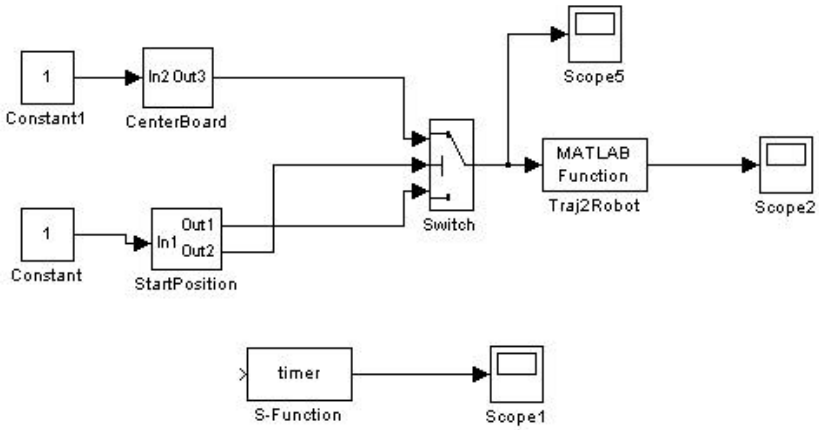


Figure A.1-Main model with sub-systems. This model is general for both experiments.

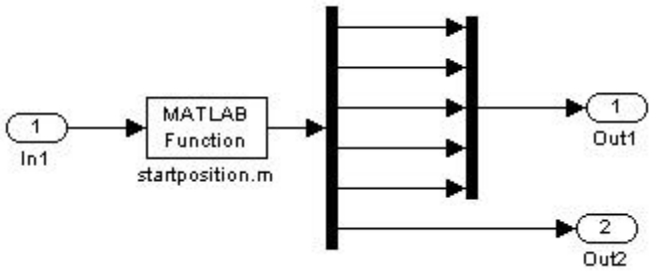


Figure A.2-Sub-system of main module. This model has the function of positioning the robot in a position to start the control.

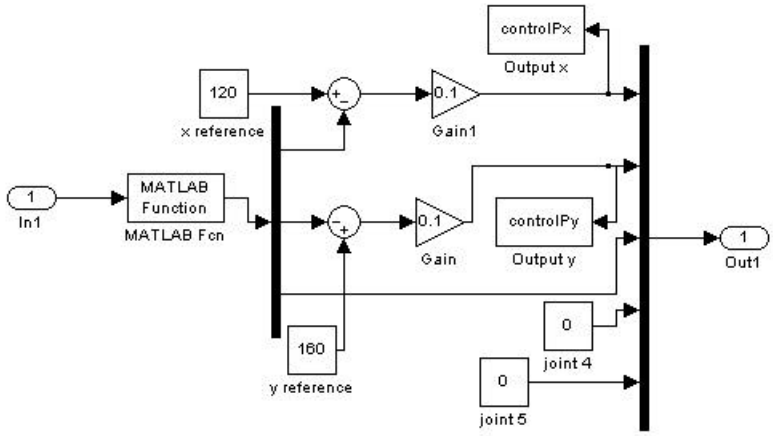


Figure A.3 sub-system in calibration experiment. This module performs the control with a proportional controller.

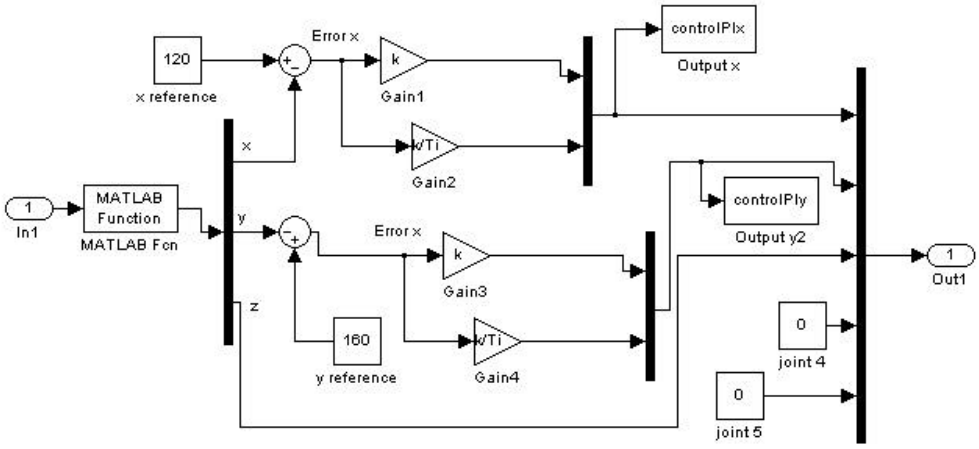


Figure A.4 -Sub-system in calibration experiment. In this module is tested the PI controller.

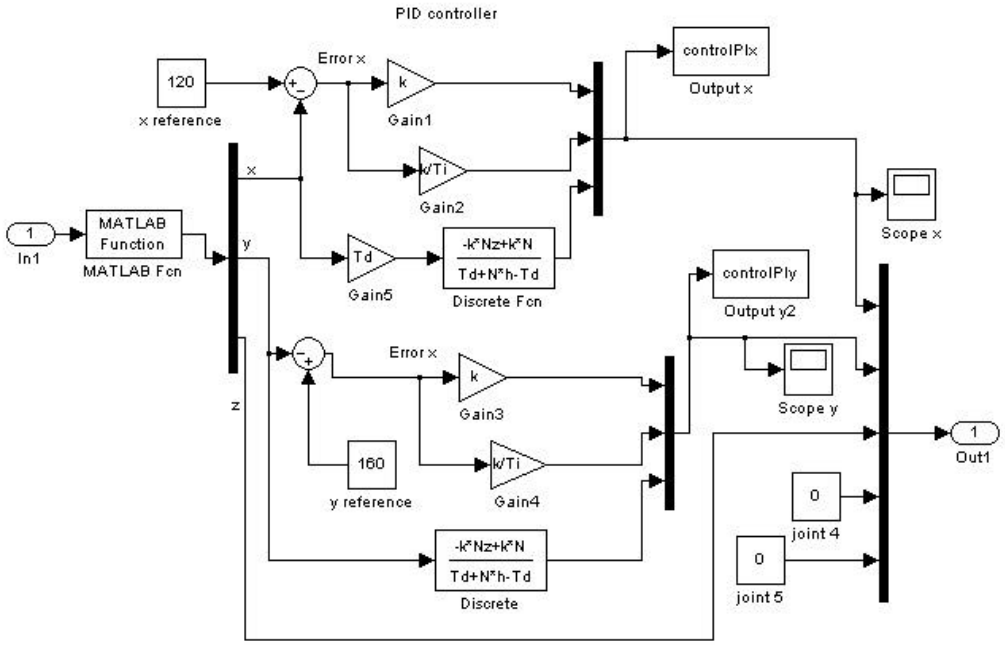


Figure A.5 -Sub-system in calibration experiment. In this module is tested the PID controller.