Wedndah Asong

Camera Calibration for Tracking Mobile Robots with Surveillance Cameras

Bachelor Thesis based on the examination and study regulations for
the Bachelor of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner: Prof. Dr. Thomas Holzhüter
Second examiner: Prof. Dr. Dieter Müller-Wichards

Date of delivery: June 6th 2007

Wedndah Asong

Title of the Bachelor Thesis

Camera Calibration for Tracking Mobile Robots with Surveillance Cameras

Keywords

Camera calibration, radial and tangential distortions, projection transformation, MATLAB, operation field, camera frame, thresholding, boundary tracing

Abstract

This report presents a step-by-step camera calibration procedure by investigation of the possible distortions. The three main distortions minimised are radial, tangential and projection distortions. Two cameras were calibrated in this project and used to track robots. The camera model used and the calibration parameters estimated are given and discussed. The extraction of calibration and control points from the images taken by the camera is accomplished by some digital image processing techniques. These techniques together with some data processing techniques used are also discussed in detail.

End of text

Wedndah Asong

Thema der Bachelorarbeit

Kalibrierung von Kameras für die Bahnverfolgung mobiler Roboter mit Überwachungskameras

Stichworte

Kamera-Kalibrierung, radiale und tangentiale Verzerrungen, Projektions-Transfomation, MATLAB, Operationsfeld, Sehfeld, Schwellenwert, Randverfolgung

Kurzzusammenfassung

Diese Arbeit beschreibt ein schrittweises Verfahren für die Kalibrierung einer Kamera durch die Untersuchung möglicher Verzerrungen. Drei Verzerrungen werden hier minimiert: radiale, tangentiale und Projektions-Verzerrungen. In dieser Arbeit wurden zwei Kameras kalibriert und benutzt, um Roboter zu verfolgen. Das benutzte Kameramodell und die geschätzten Kameraparameter werden vorgestellt und diskutiert. Das Extrahieren von Punkten aus den Bildern der Kameras geschieht durch digitale Bildverarbeitung. Dies wird zusammen mit der benötigten Datenverarbeitung ebenfalls diskutiert.

Ende des Textes

Contents

Chapter 1: Introduction

Cameras are widely used in many fields of application today, in mobile phones, webcams, for video conferencing, just to name a few. They may be used simply for visual purposes or for monitoring or tracking. Remote monitoring is now the state of the art. Here, equipments, devices, some work area or even a location is remotely monitored over a local network or the internet. Some monitoring systems also provide remote control of the camera. Cameras are also used for surveillance or to supplement alarm systems. Some surveillance cameras are used to automatically read the licence plate of moving vehicles. In an alarm system, they may be triggered by infrared light used to detect moving objects in a given area. They may also be used to track moving objects, in which case, there is need to calibrate the camera. This is exactly the case in this project.

The aim of calibrating a camera is to determine and estimate a set of camera parameters that describe the mapping of 3-D reference coordinates in the camera view to 2-D image coordinates. There are various camera calibration procedures presented in the past. A camera calibration procedure involves the extraction of control points from images, model fitting, correction of the lens distortion of the camera and analysis of possible errors originating from the previous stages. The most challenging task in camera calibration is to determine the camera parameters which compensate for the lens distortions. This is due to the non-linear nature of the distortions. The distortions of the control points in an image increase from the centre of the image to its borders as can be seen in the following figure. The image in this figure was taken using one of the cameras used for this project in the lab. The small circles in this image actually form a rectangular pattern but the pattern does not appear rectangular in the image. This is as a result of distortion.

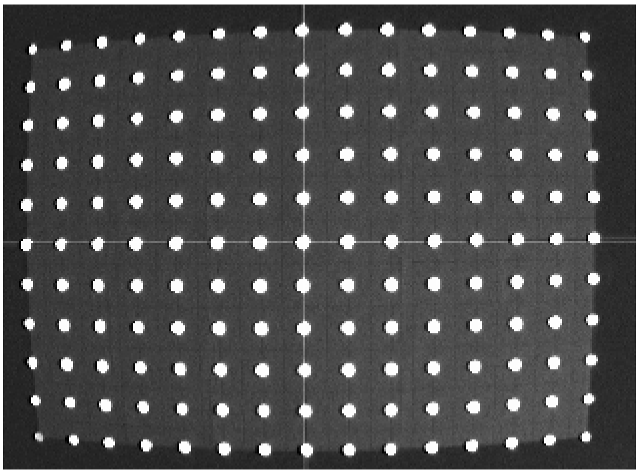


Figure 1.1: An illustration of camera distortion

The most outstanding distortion caused by a camera lens is radial distortion. It can be of pin-cushion type, barrel type or even a combination of both as described in [14]. In the pin-cushion distortion, the distance of the object from the centre of the image over estimates the real distance in the world coordinate system while that of the barrel type is underestimated. Both types of distortions are nonlinear, i.e. the image of a rectangle in the world coordinates does not appear as a rectangle in the image. Straight lines are transformed into curved lines when radial distortion is present.

Most simple lenses in use today are spherical. Spherical lenses are easy to grind and have been in use for centuries. An image is properly transformed (i.e. without distortion) only when its maximum distance from the optical centre of the lens is small (this implies small angles of incidence of light rays) compared to the focal length of the lens. However, when the distance is large the mapping object-to-image is not perfect anymore. The radial distortion of a checkerboard pattern is illustrated in *figure 1.2*. A comparison of *figure 1.1* and *figure 1.2* clearly shows that the camera lens used in this project causes barrel distortions.

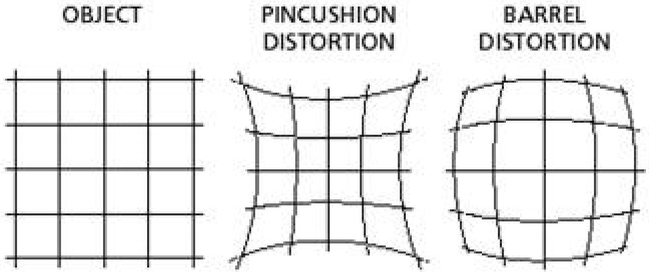


Figure 1.2: Illustration of radial distortions

The extraction of the control points from the image (*figure 1.1*) involves digital image and data processing. Generally, solutions to problems in the field of digital image processing require a lot of experimental work and testing with large sets of sample images. It should be noted here that it is required to keep the image processing time as low as possible because the position and orientation of the robots must be updated fast enough to have better control of it. The problem that arises here is that the pixel intensities of the digital images obtained from the camera vary with luminance. This makes it difficult to develop a general algorithm for thresholding the image as discussed in chapter 4.

The task in this project is to determine the position and orientation of mobile robots with the aid of surveillance cameras. Of course the draw back here is that the position of a robot can not be determined if the robot is out of the cameras' view. To accomplish this task, the robots are equipped with two circular reflex marks at their top. The position and orientation of a robot can then be obtained by determining the centres of these marks. Robot control is not part of this project. There are previous projects carried out on these robots by Wang Q. & Ye G. [8] and Chung J. [9]. Robot control was part of these projects.

1.1 Project Overview

The work station of this project consists of two over head cameras mounted on the ceiling, one or several robots in the operation field and a PC control station with Windows XP as operating system. The cameras are connected to the PC by firewire cables and the robots by wireless connections. An overview of the work station is presented in the following figure.

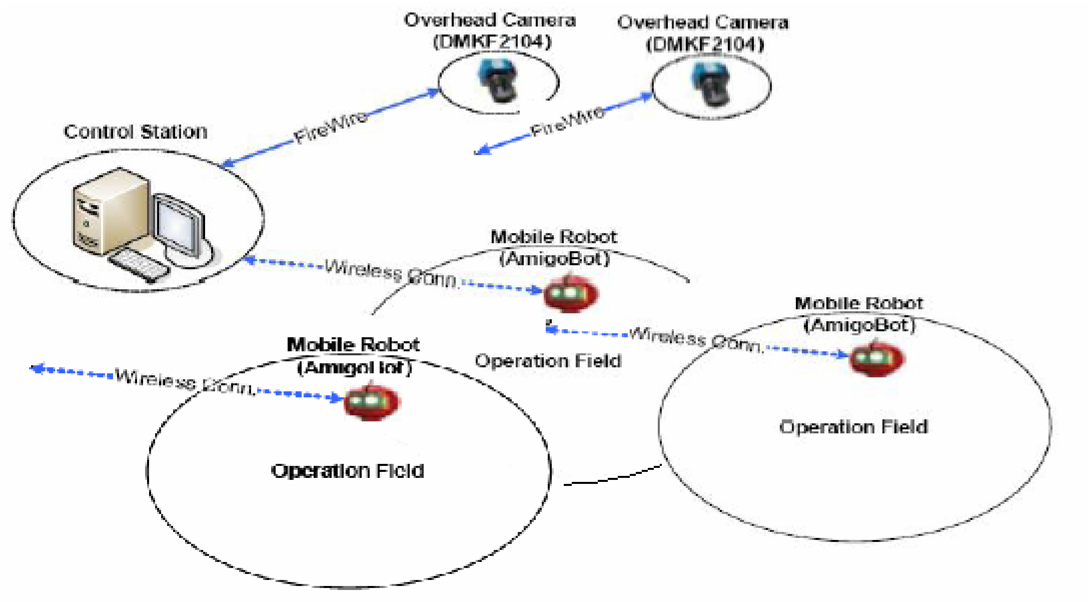


Figure 1.3: Overview of project workstation

Two circular reflex marks are attached to the top of the robot, one larger than the other. The larger mark has a radius of 70 mm and is placed just above the tyres and the smaller mark has a radius of 50 mm and is placed in front of the larger mark. So the position of a robot corresponds to the centre of the larger mark and the orientation can be obtained by computing the angle between the x-axis and a line drawn from the centre of the larger reflex mark through that of the smaller one as shown in the following figure.

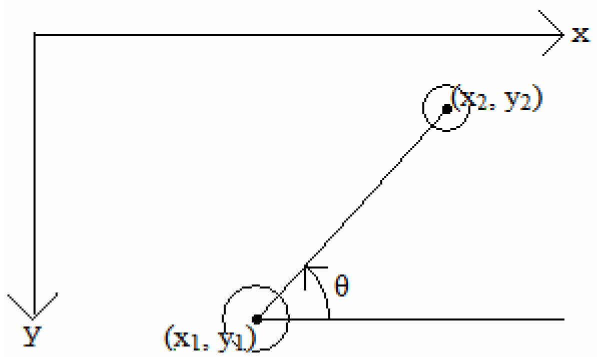


Figure 1.4: Parameters used in determining the state of a robot

Consider that the actual coordinates of the centres of the reflex marks of a robot have been calculated as depicted in the figure above. The coordinate axes show the positive x- and y-direction. The coordinates of the centre of the larger reflex mark is given by (x_1, y_1) and that of the smaller by (x_2, y_2). Then the position of the robot is given by the coordinates (x_1, y_1) and its orientation by the angle θ measured anti-clockwise from the positive x-axis. The angle θ is computed as follows;

$$\theta = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

The task of this project involves the following stages:

- Camera calibration using appropriate calibration images.
- Determination of robot position which involves:
    - Acquiring image from camera.
    - Segmenting the image with a suitable threshold to obtain a binary image in which the reflex marks (objects) are white and the rest of the image (background) is black.
    - Determining the boundaries of the white objects in the binary image.
    - Computing the coordinates of the centre of these objects.
    - Calculating the actual reference coordinates on the operation field from the image coordinates using camera calibration.

Robot control is not discussed in this project.

1.2. Thesis Outline

Chapter 2 gives an overview of the system used in accomplishing the task of this project. The specification and requirements of the hard- and software are discussed in this chapter. A brief description of the robots and cameras and a short discussion of the MATLAB environment are given. The MATLAB toolboxes used in this project are also introduced.

Chapter 3 is made up of two parts – camera configuration and image acquisition. Here a description is given on how the camera is configured to obtain suitable images to enhance their processing and how these images are loaded into the MATLAB workspace from the cameras.

In chapter 4, the digital image processing techniques used in this project are discussed. Here a brief discussion on digital image representation and image types is given. Some digital image processing techniques like thresholding and boundary tracing techniques are also discussed in this chapter.

Chapter 5 gives a complete discussion of the algorithm used in obtaining the coordinates of the centres of the circular reflex marks. The algorithm chosen here was used to obtain sub-pixel accuracy. Further mathematical details like the least square fit are also discussed here; specifically circle and ellipse fit are discussed. Finally the projection correction of the robot height is discussed.

Chapter 6 discusses the camera calibration procedure used in the MATLAB calibration toolbox. Here a first attempt to calibrate the camera with this toolbox and an analysis of the results are given.

Chapter 7 discusses the step-by-step camera calibration procedure adapted for this project. Here a step-by-step camera calibration procedure is introduced. A discussion is first given on the possible sources of distortions and their effects are demonstrated. In each calibration step, a distortion is chosen and minimised. The outcome of the minimisation at each step is also demonstrated and the overall error minimisation is also given for each step.

Chapter 8 gives a conclusion about the project and general observations and difficulties encountered while carrying out the task of this project.

1.3. Development Environment

Each camera has a view of about 3400 mm x 2400 mm on the plane of the operation field but the operation field of the robots is just 2800 mm x 2000 mm in each camera view and somewhat centralised in the camera view. This is because only this area of each camera view is calibrated. Also, when the robot is too close to the edge of the camera view, the reflex marks reflect too little light to make it possible to differentiate it from the background. The following figure shows the dimensions of the operation field of the robots and that of the camera view for each one camera.

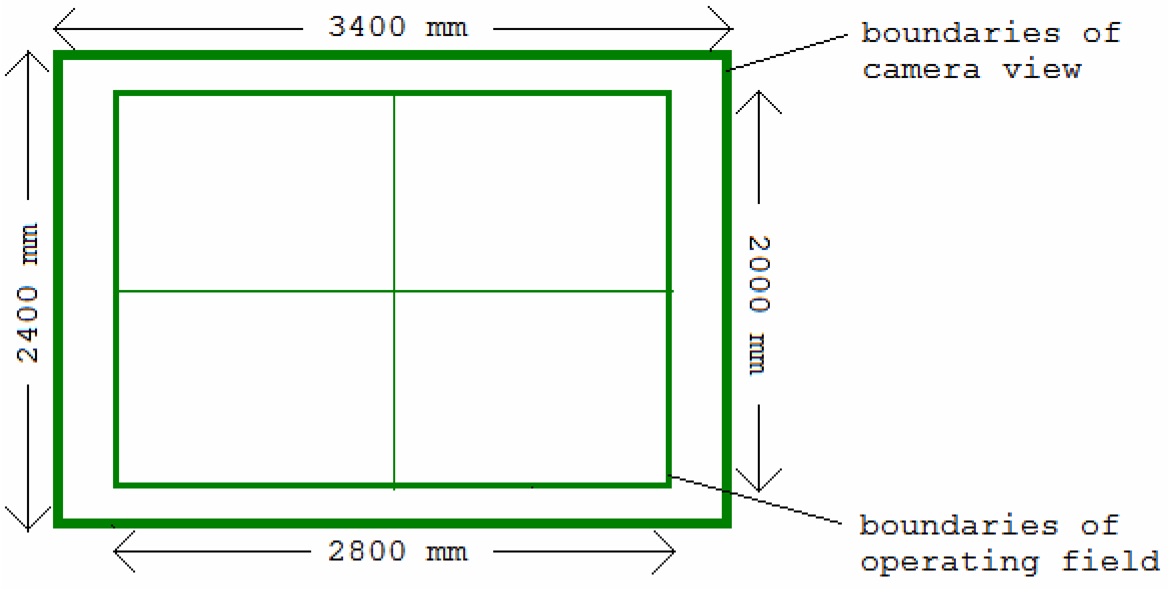


Figure 1.5: Camera view and operation field

The centres of the part of the operation field in the view of both cameras are at a distance of about 2000 mm away from each other. This makes the part of the operation field in the view of both cameras overlap as depicted in the following figure. The coordinate axes of the operation field are also depicted in this figure. To distinguish the two cameras, the one to the right used to track the robots in the red frame will be referred to as camera 1 and the one to the left with the green frame will be referred to as camera 2.

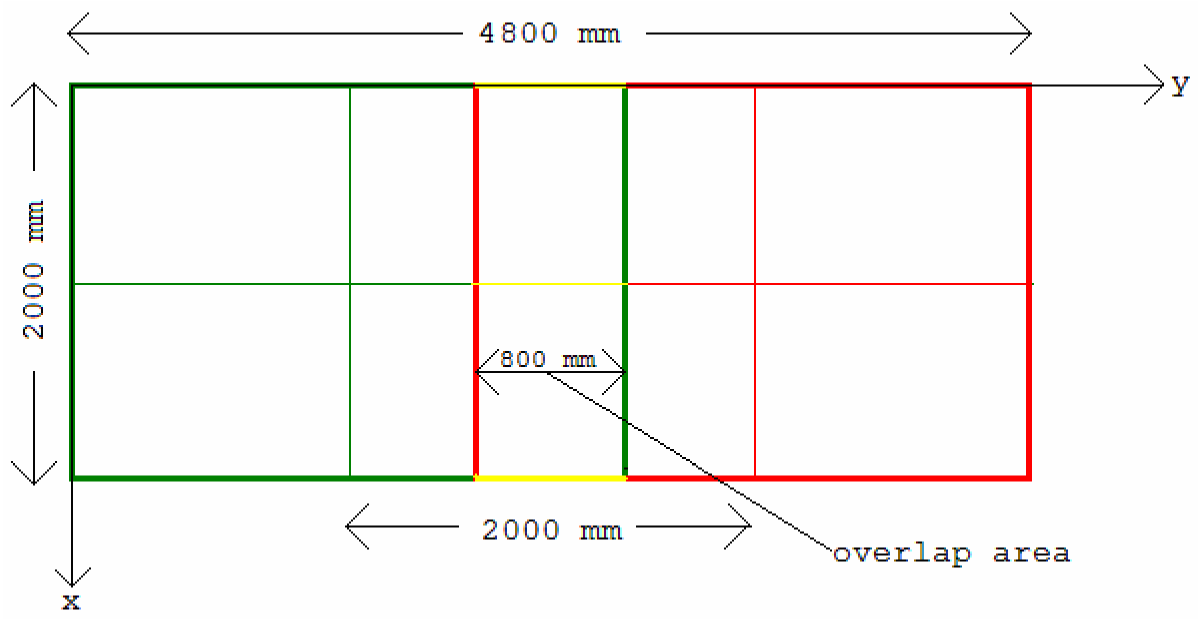


Figure 1.6: Complete view of operation field from both cameras

Chapter 2: System Overview

A description of the soft- and hardware used to accomplish the task in this project is introduced and discussed in this chapter.

2.1 Hardware Specifications:

The hardware mainly used in this project is the mobile robot, reflex marks and the camera. A brief discussion of these hardware components are given below;

2.1.1 Mobile Robot:

The AmigoBot mobile robot from MOBILEROBOTS ActivMedia Robotics is used in this project. This robot is designed specifically for indoor use especially for classroom applications. The following figure shows a picture of this robot viewed from the top.

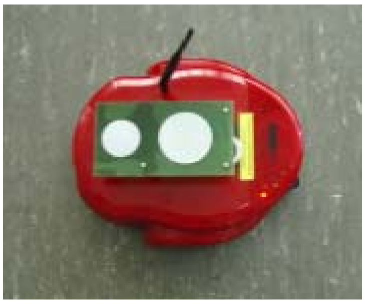


Figure 2.1: AmigoBot

The AmigoBot has dimensions 280mm x 330mm x 165mm and weighs 3.6kg with battery. A green plate with two circular reflex marks is mounted on top of the robot. The reflex marks are described in section 2.1.2. The robot has a high-impact plastic shell which covers a differential drive system with twin motors and two shaft encoders and the drive system is encased with aluminium.

An AmigoBot can drive up to 300 feet away from its control PC. To run an AmigoBot, you require the following;

- PC with 300 MHz or faster processor with one free serial port
- WIN 2000/XP or Linux operating system
- Hard drive with at least 11 Mb free space
- CD ROM drive
- IEEE 80211b Ethernet access point (on your hup) or peer-to-peer station adapter

Since this project does not involve robot control, the internal structure and functionality of the robot will not be discussed here.

2.1.2 Circular Reflex Marks:

The reflex marks were used (as explained in section 1.1) to determine the position and orientation of the robot. These reflex marks are highly reflexive to light. They reflect light back to their source. Their main light source was from the LED rings around the camera. This gave them high light intensities in the images taken by the camera.

The reflex marks are made of Diamond Grade^{TM} VIP (Visual Impact Performance) reflective sheeting from 3M Corporation. There are several colours available for this product: white, yellow, red, blue and green. In this project the white ones were used (product code: Series 3990). The reflexive sheeting has an interlocking diamond seal pattern. It is shown in the following figure. For more information on the reflexive sheeting, visit the web document given in [11].

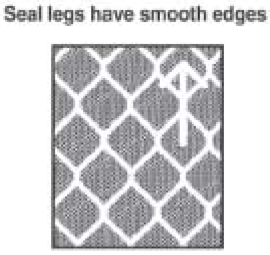


Figure 2.2: Reflexive Sheeting

2.1.3 Camera:

The overhead CCD (Charged Coupled Device) cameras are the DMK 21F04 model from Imaging Source. They take digital images with pixel dimension 640 x 480 at a frame rate of up to 30 fps. The camera has an integrated 10 bit – ADC with an output of 8 bits. Each pixel intensity is stored in 1 byte making the intensities range from 0 to 255.

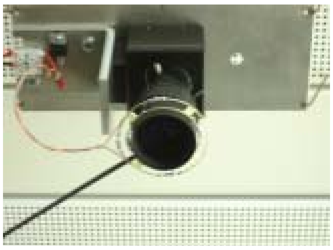


Figure 2.3: CCD camera

The camera is mounted on the ceiling with four parallel connected LEDs and some resistances as shown in the figure above. It is connected to the PC via a FireWire interface 1394. The PC used in the project has a 6-pin FireWire socket which supports FireWire interface 1394. The connection between PC and camera is shown in the following figure;

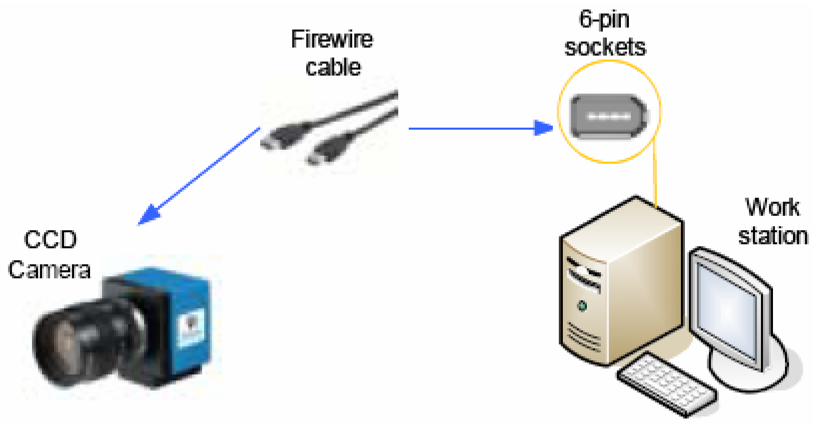


Figure 2.4: Connection between camera and PC

2.2 Software Specifications:

The main software used in this project is MATLAB. In this section, the capabilities of MATLAB is briefly discussed and a few of its toolboxes used in this project are also introduced.

2.2.1 MATLAB:

The name MATLAB stands for MATrix LABoratory. MATLAB [12] is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include the following fields;

- Math and computation.
- Algorithm development.
- Data and image acquisition.
- Modeling, simulation and prototyping.
- Data analysis, exploration and visualization.
- Scientific and engineering graphics.
- Application development, including graphical user interface building.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows users to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or Fortran.

2.2.2 Image Acquisition Toolbox:

The Image Acquisition Toolbox is a collection of functions that extend the capability of the MATLAB numeric computing environment. The toolbox supports a wide range of image acquisition operations, including;

- Acquiring images through many types of image acquisition devices, from professional grade frame grabbers to USB-based Webcams.
- Viewing a preview of the live video stream.
- Triggering acquisitions.
- Configuring callback functions that execute when certain events occur.
- Bringing the image data into the MATLAB workspace.

The toolbox uses components called hardware device adaptors to connect to devices through their drivers. The toolbox includes adaptors that support devices produced by several vendors of image acquisition equipment. In addition, the toolbox includes an adaptor for generic Windows video acquisition devices. The following figure shows these components and their relationship.

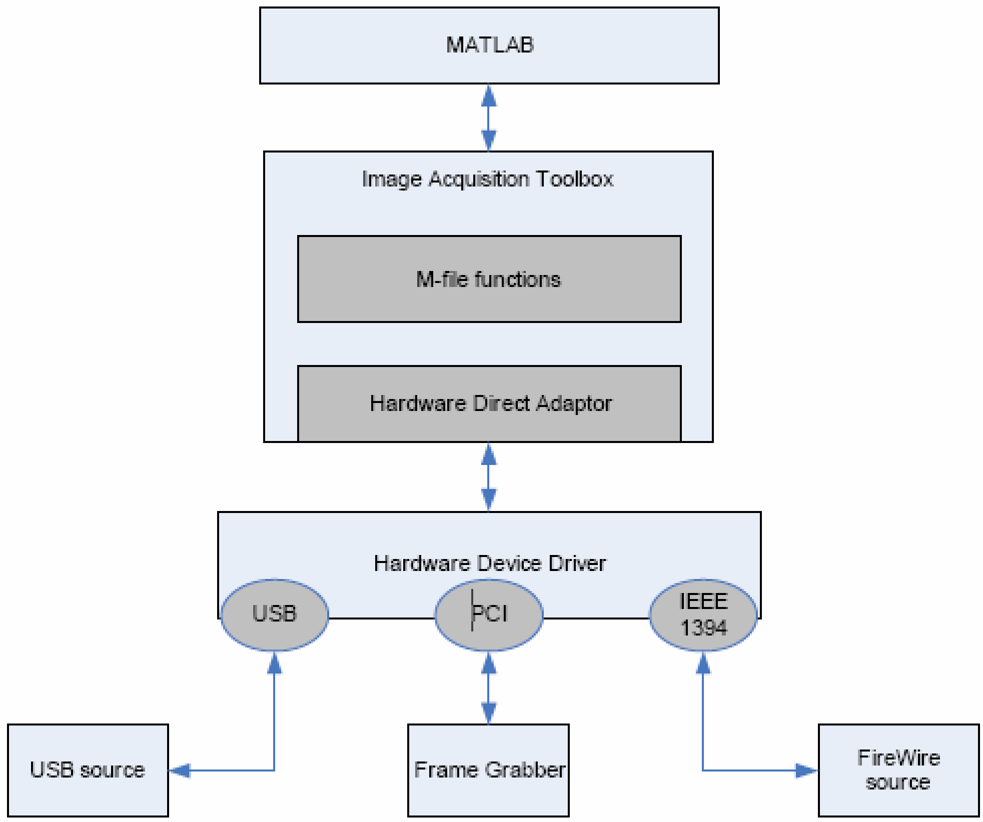


Figure 2.5: Components of Image Acquisition Toolbox

This toolbox was used in this project to get images from the camera into the MATLAB workspace and also to configure the camera. For more information on this toolbox, see [12].

2.2.3 Image Processing Toolbox:

The Image Processing Toolbox is a collection of functions that extend the capability of the MATLAB numeric computing environment. The toolbox supports a wide range of image processing operations, including;

- Spatial image transformations.
- Morphological operations.
- Neighborhood and block operations.
- Linear filtering and filter design.
- Transforms.
- Image analysis and enhancement image registration.
- Deblurring.
- Region of interest operations.

This toolbox is used in this project to process the images acquired from the camera in order to determine the position and orientation of the robot. For more information on this toolbox, see [12].

2.2.4 Camera Calibration Toolbox:

This toolbox is not original part of the toolboxes in MATLAB. It is an open source available on the web and can be downloaded for free. The toolbox provides functions for camera calibration. The camera model used in this toolbox is explained in section 6.6. For more information on downloading and using this toolbox, see the online web documentation at [5].

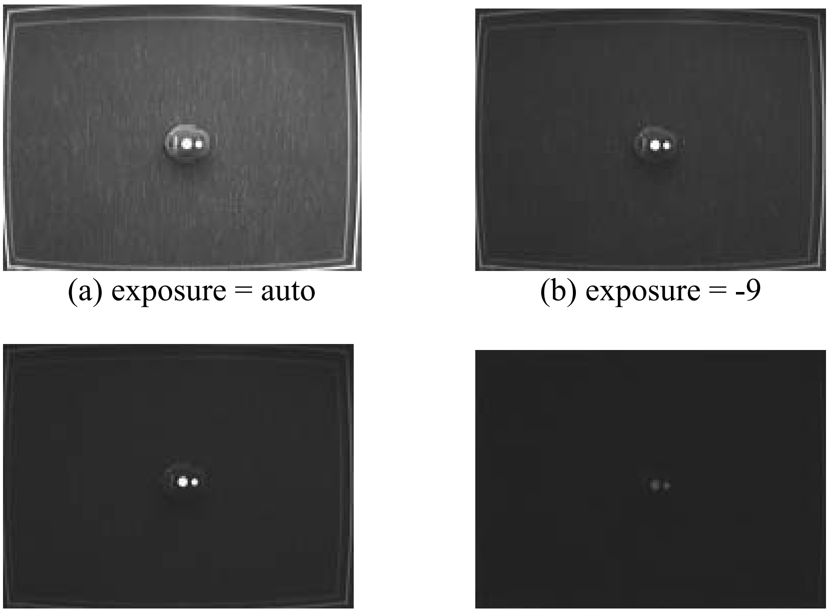
Chapter 3: Image Acquisition

This is the first step done in determining the position and orientation of the robot. In this chapter the image acquisition procedure used in MATLAB will be discussed together with the camera configuration. The camera configuration is particularly important in order to obtain good and easy to process images.

3.1 Camera Configuration:

Before acquiring an image, the camera is first configured. This is done with the help of a camera object. The three main parameters of the camera object to be set are brightness, contrast and exposure. Of most importance here is the exposure parameter which defines the amount of time the camera shutter is kept open to receive light. It takes values in the range from -11 to -5 and these correspond to 2^{-11}s to 2^{-5}s. The exposure time is also known as shutter time. Varying the value of the exposure varies the intensity of each pixel.

When an image is taken, the sole interest rests on the reflex marks. The centre of the two reflex marks of each robot is to be determined and used to calculate the position and orientation of the robot. The exposure time of the camera is therefore set to achieve a maximum object (reflex marks) to background light intensity ratio. Below are four images taken from camera 1 with different exposure times.

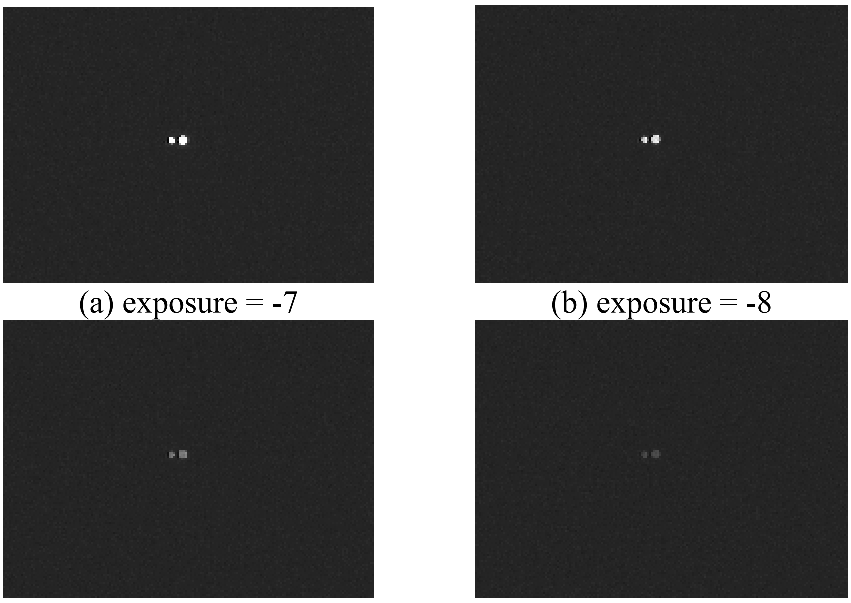


(a) exposure = auto (b) exposure = -9

(c) exposure = -10 (d) exposure = -11

Figure 3.1: Variation of exposure time for camera 1

As can be seen from the above images, the light intensity ratio of the object to the background varies with changes in the exposure time. However, in (d) the reflex marks can hardly be identified. (c) with exposure = -10 was chosen for the images taken with camera 1 in this project. The brightness and contrast of the images were set to 0 and 150 respectively. The case was different for camera 2 as demonstrated in the following figure;



(a) exposure = -7 (b) exposure = -8

(c) exposure = -9 (d) exposure = -10

Figure 3.2: Variation of exposure time for camera 2

As can be seen from the figure above, the image in (d) with exposure of -10 will be of little or no help. An exposure of -8 as in (b) was chosen for camera 2. The brightness and contrast of camera 2 was also set at 0 and 150 respectively.

For the calibration images of both cameras, an exposure of -5 was used. The calibration images have a lot of reflex marks in them which tend to reflect a lot of light. Due to the different angles of incidence of light on the operation field, the reflex marks at the centre of a cameras view reflect much more light than those at the borders. A small exposure here leads to a situation where some reflex marks at the borders can not be differentiated from their background. Therefore, a higher exposure of -5 was used for this project to take the calibration images.

3.2 Image Acquisition Procedure:

In this project, the camera driver is supported by the MATLAB image acquisition toolbox. So it is easy to acquire an image. The procedure used in the S-function developed by Wang and Ye [8] to perform the image initialization, image acquisition and image cleaning was used in this project. The image acquisition procedure can be divided into the following four steps;

1. Create a video input object.

In this step the created video input object is used to describe the connection between MATLAB and the image acquisition device. The properties of the acquisition device can be changed by setting the properties of the video object. This step is done in the initialization procedure of the S-function.

2. Configure the properties of the image acquisition object.

After creating a video input object, one can change the characteristics of an image or other aspects of the acquisition process by setting the values of the image acquisition object properties. This ensures that the device works in a desired way. There are two types of objects to represent the connection with an image acquisition device:

- Video input object.
- Video source object.

The properties supported by the video input object are the same for every acquisition device. The video source object represents single entities that correspondent to the physical device. At any one time, only one video source object can be active. The video source object properties differ with the different video devices.

In this project the image processing time is much longer than the automatic frame triggering time (30 frames/s). So the trigger time was manually set. A frame is gotten with every triggering time. The triggering process is repeated infinitely.

3. Acquire image data.

After creating the video input object and configuring the properties of an image acquisition object, the most important thing is to acquire the image data. This process includes the following three steps;

- Start video input object.

  This is a preparation procedure for the next step. It starts the acquisition by using the function *start()* in MATLAB, and is carried out in the initialization part of the S-function.

- Trigger the image acquisition.

  This is the core procedure in acquiring image data. Because the trigger is manually set in this project, the object waits for a call to the trigger function before it initiates data acquisition. It is done in the output part of the S-function.

- Bringing data into the MATLAB workspace.

  The image acquisition toolbox stores acquired data in a memory buffer. In order to work with this data, it must be brought into the MATLAB workspace. In MATLAB there is a *getdata()* function which brings multiple frames into the workspace. It is called in the output part of the S-function.

4. Clean up the video input object.

After the image acquisition has been done, the acquisition process should be stopped and the video object must be deleted, otherwise other applications cannot use the device. This is done in the termination of the S-function.

The whole image acquisition procedure is illustrated in the following figure. The frame numbers between two acquisitions depend on the step time and can vary. The acquired frames are those frames from *F6* to *F10*.

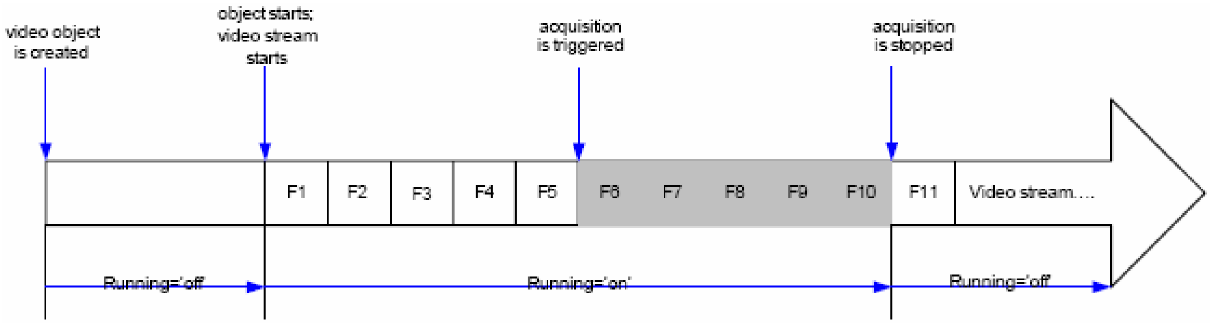


Figure 3.3: Image acquisition procedure

The function *get_image()* (see appendix) implements this image acquisition procedure.

Chapter 4: Digital Image Processing

Digital image processing is characterized by its significant level of testing and experimentation in order to arrive at an acceptable solution. The main problem faced is the reduction of time and cost taken up by digital image processing, i.e. developing algorithms fast enough to accomplish their task in a required time.

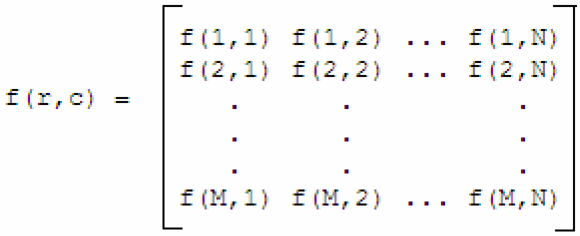
The two main digital image processing techniques used in this project were **thresholding** and **boundary tracing**. Thresholding was used to segment the grey images obtained from the camera to get binary images, in which the objects (reflex marks) were white and the rest of the image (background) is black. Boundary tracing was then applied to the computed binary images to obtain the boundaries of the white reflex mark objects.

An image can be defined as a two dimensional function, $f(x,y)$, where x and y are spatial coordinates and the amplitude of f at any pair of coordinates (x,y) is called the intensity or grey level at that point. The terms intensity and grey level are used interchangeably in this project report. If x, y and the amplitude values of f are all finite and discrete, the image is called a digital image. The field of digital image processing refers to processing a digital image with a digital computer. Digital images are made up of a finite number of elements with a particular location and value. These elements are referred to as picture elements, image elements, pels or pixels. Pixel is the term most commonly used and is also used throughout this project.

4.1 Digital Image Representation:

Consider an image $f(x,y)$ as described in the introduction of this chapter. The amplitude values of f and the spatial coordinates x and y may be continuous. The image may then be digitized to obtain finite discrete values for the amplitude and spatial coordinates. The process of digitizing the spatial coordinates is called sampling and the process of digitizing the amplitude is called quantization. A digital camera is used in this project. So the processes of sampling and quantization are done automatically by the camera and only digital images are obtained from the camera.

A digital image is of a given height (H) and a given length (L). H is the number of pixels in the y-direction and L is the number of pixels in the x-direction. x may take values from 0 to L-1 or from 1 to L. Similarly, y may take values from 0 to H-1 or from 1 to H. In MATLAB, images are simply represented as 2-dimensional matrices with M rows and N columns. Here the rows (r) correspond to the y-coordinates and columns (c) to the x-coordinates. So an MxN matrix is an image of height M and length N. In MATLAB, the rows always take values from 1 to M and columns from 1 to N as shown in the following figure.



$$f(r,c) = \begin{bmatrix} f(1,1) & f(1,2) & \dots & f(1,N) \\ f(2,1) & f(2,2) & \dots & f(2,N) \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ f(M,1) & f(M,2) & \dots & f(M,N) \end{bmatrix}$$

Figure 4.1: Matrix representation of an image in MATLAB

Note from *figure 4.1* that the rows are denoted before the columns. So a pixel location of *f*(x,y) in the real world representation corresponds to a matrix entry of `f(y,x)` in MATLAB.

4.2 Digital Image Types:

Most image processing operations are carried out on monochrome images (intensity and binary images) and these are the images used throughout this project. The operations can be extended to RGB images and applied to each component image but this will take a longer time. For this reason, the cameras are manually set to take just grey images.

4.2.1 Intensity Images:

An intensity image is also called a grey image. Intensity images in MATLAB are represented by a data matrix. The data may be of class uint8 (8-bit unsigned integer) or uint16 (16-bit unsigned integer) having values in the ranges [0,255] or [0,65535] respectively. It may also take scaled values in the range [0,1] if it belongs to the class double.

4.2.2 Binary Images:

The word binary says it all. A binary image has just two distinct intensities; 0 or 1. Pixels labelled 0 are black and those labelled 1 are white. Therefore it is not strange that binary images are commonly referred to as black and white images. In MATLAB, a binary image is also a data matrix but of the class logical. Therefore an image of class, say uint8, with 0s and 1s is not a binary image in MATLAB.

4.2.3 RGB Images:

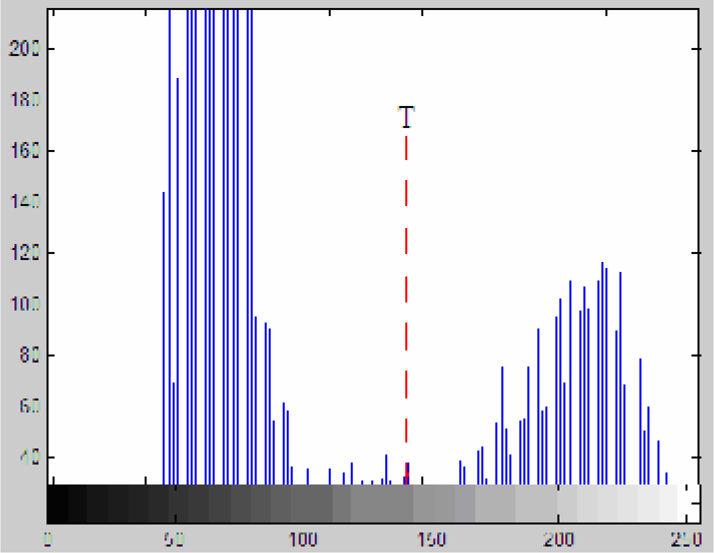
RGB (Red-Green-Blue) images are coloured images and are represented in MATLAB by an M x N x 3 array. They can be viewed as a stack of three M x N matrices, each containing the corresponding red, green and blue pixel component values of the RGB image at a specific spatial location. Each M x N matrix can be viewed as an intensity image of a specific class, having a specific range of values as explained above. The number of bits used to represent one pixel is known as the depth of the RGB image. If the image is of class uint8, then it has a depth of 24 bits. An image with a bit depth of b can have up to $(2^b)^3$ different colours. This corresponds to 16,777,216 colours for an RGB image of class uint8.

4.3 Thresholding:

Thresholding is an image process typically carried out on the image histogram. An image histogram (*figure 4.2*) is a plot of the pixel count in an image against the pixel intensities. The goal of thresholding is to separate light objects in an intensity image from a dark background resulting in a binary image. Usually the objects and the background are divided into two dominant modes as shown in the histogram of *figure 4.2*. Suppose an intensity image f(x,y) is to be thresholded with a threshold T to produce a binary image g(x,y). Then g(x,y) is defined as follows;

$$g(x,y) = \begin{cases} 1 \text{ if } f(x,y) \geq T \\ 0 \text{ if } f(x,y) < T \end{cases} \qquad (4.1)$$

In the binary image g(x,y), all pixels labelled 1 correspond to the objects and are white. On the other hand, all pixels labelled 0 correspond to the background and are black.



Vertical axis: Number of pixels
Horizontal axis: Pixel intensity

Figure 4.2: Image histogram and possible threshold (T)

4.3.1 Global Thresholding:

When the threshold T is constant, it is said to be global. Global thresholding makes use of a global threshold to segment the image. One way of choosing a global threshold is by visual inspection of the image histogram as shown in *figure 4.2*. Here the object and background pixels are clearly divided into two modes. Another way is by try and error where the user changes the threshold by means of a slide or buttons to increment and decrement the threshold or by simply entering different thresholds. Here an iterative algorithm is used and the user sees the results of his actions immediately on the image.

In this project, it is of course not possible to select a threshold by user interaction. This is due to the time constraints for updating the robot state in order to have better control of it. Therefore there is need for a fast algorithm to automatically determine the threshold. The algorithm used in this project will be discussed in section 4.3.3.

4.3.2 Local Thresholding:

Global thresholding can fail when the background of the image is uneven or when the pixel count of the objects in the image is very small as compared to that of the background. This is exactly the problem faced in this project. This problem is partly solved by choosing a suitable exposure time for the camera shutter to obtain an optimal illumination as earlier explained in section 3.1. A common practice is to preprocess the image to compensate for the illumination and then apply a global threshold. This process is equivalent to thresholding an image f(x,y) with a varying threshold T(x,y) defined as follows;

$$g(x,y) = \begin{cases} 1 \text{ if } f(x,y) \geq T(x,y) \\ 0 \text{ if } f(x,y) < T(x,y) \end{cases} \qquad (4.2)$$

where

$$T(x,y) = f_0(x,y) + T_0$$

The image $f_0(x,y)$ is the resulting image from pre-processing the image f and the constant T_0 is the result of applying a global threshold to the image f_0. Techniques for pre-processing an image before thresholding are complicated and take a lot of time. Therefore these techniques were not considered in this project. How then was this problem overcome in this project? That will be discussed later in section 4.3.4.

4.3.3 Ostu Thresholding Algorithm:

The Ostu algorithm [13] is a histogram-based method and computes a global threshold for its input image. The histogram is normalized and treated as a probability density function.

$$p_r(r_q) = \frac{n_q}{n} \qquad\qquad q = 0, 1, 2, \dots , L - 1$$

where n is the total number of pixels in the image, n_q is the number of pixels that have intensity level r_q and L is the total number of possible intensity levels in the image. A threshold k is chosen such that C_0 is the set of pixels with levels [0, 1, 2, … , k - 1] and C_1 is the set of pixels with levels [k, k +1, k + 2, … , L - 1]. Ostu's method chooses the threshold value k that maximizes the between-class variance σ_B2, which is defined as

$$\sigma_B^2 = \omega_0 (\mu_0 - \mu_T)^2 + \omega_1 (\mu_1 + \mu_T)^2$$

where

$$\omega_0 = \sum_{q=0}^{k-1} p_r(r_q)$$

$$\omega_1 = \sum_{q=k}^{L-1} p_r(r_q)$$

$$\mu_0 = \sum_{q=0}^{k-1} q\, p_r(r_q) \;/\; \omega_0$$

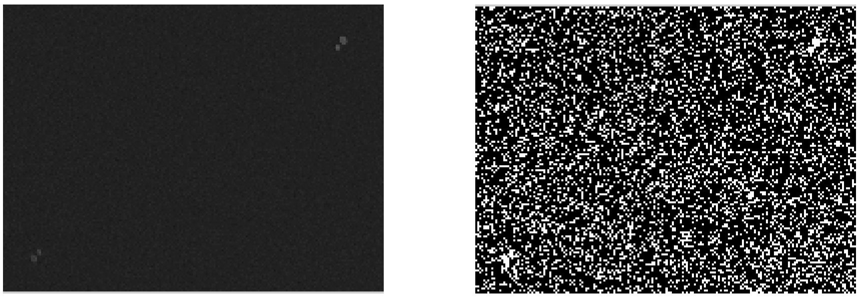
$$\mu_1 = \sum_{q=k}^{L-1} q\, p_r(r_q) \;/\; \omega_1$$

$$\mu_T = \sum_{q=0}^{L-1} q\, p_r(r_q)$$

The MATLAB function *graythresh* implements this algorithm. It can be called in the following way:

T = graythresh(f)

Its input f is an intensity image and the output T is the normalized global threshold (in the range [0,1]) computed using the Ostu algorithm. An intensity image with two robots (*figure 4.3 (a)*), one at the top-right of the camera view and the other at the bottom-left, was thresholded using *graythresh*. The resulting binary image (*figure 4.3 (b)*) contains a lot of noise. A boundary tracing algorithm can not be successfully applied to this image to obtain the boundaries of the four reflex marks (two for each robot). Therefore a customized adaptive thresholding algorithm applicable just to the images of this project is needed.



(a) Intensity image (b) Binary image

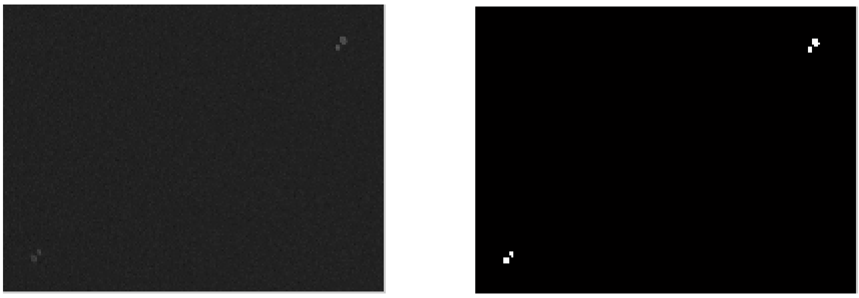
Figure 4.3: Thresholding using MATLAB function *graythresh*

4.3.4 Customized Adaptive Thresholding Algorithm:

The MATLAB function *graythresh* does not work globally as demonstrated in *figure 4.3* because the object is much too small as compared to the background. This problem was solved in this algorithm by applying *graythresh* locally and not globally. The image is divided into twelve square sub-images of dimensions 160 pixels. The function *graythresh* is then applied to each sub-image to calculate a local threshold for the region defined by the sub-image. But what about the regions which do not contain any reflex mark? Applying *graythresh* to these regions will result in exactly the situation in *figure 4.3(b)*, which is to be avoided.

This algorithm was developed in such a way that only regions which contain reflex marks are thresholded. This was achieved by setting a lower bound for the maximum grey level in the sub-image. The lower bound for each camera is gotten from the maximum grey level of the image taken from the camera with no robots in view. The default value for camera 1 is 46 and that for camera 2 is 47 as calculated in the lab. However, this will vary with luminance and should be set before running the program to track robots. These lower bounds are global variables and the user is given the option to set them when the global variables are being declared and initialised. The function *set_Tmin()* (see apendix) was used to initialise them.

Another difficulty encountered was with regions which cut off just a small part of a reflex mark. This small part of the reflex mark takes the maximum grey level of that region above the defined lower bound. As such *graythresh* will be applied to this region. Since the object here is still very small relative to the background, one arrives back at the situation in *figure 4.3(b)*. To solve this problem, the set lower bound was not only taken as a lower bound for the maximum grey level, but also the lower bound for the threshold. Below is a result of applying this algorithm to the image in *figure 4.3(a)*.



(a) Intensity image (b) Binary image

Figure 4.4: Thresholding using adaptive customised thresholding algorithm

The function *imthresh()* (see appendix) implements the thresholding algorithm just described. This algorithm is summarised below;

For each sub-image;

1. Calculate maximum grey level.
2. If maximum grey level is less than defined lower bound, go to step 3 else go to step 4.
3. Set corresponding region of sub-image in original image to black and continue with step 1 for next sub-image.
4. Compute local threshold with *graythresh*.
5. If local threshold is less than lower bound, set it to lower bound
6. Threshold sub-image with local threshold.
7. Assign binary sub-image to corresponding region in original image.

4.4 Boundary Tracing:

This is the most time costly step in this project. Boundary tracing is carried out on binary images. As already discussed, a binary image is made up of white objects on a black background. Boundary tracing is used to trace the inner or outer boundaries of the objects in a binary image. The inner boundary is made up of white pixels and is part of the object region while the outer boundary constitutes black pixels and is not part of the object region. The inner boundary tracing algorithm as described in [15] is as follows:

1. Search the image from top left until a pixel of a new region is found. This pixel P_0 then has the minimum column value of all pixels of that region having the minimum row value. Pixel P_0 is the starting pixel of the region border. Define variable dir which stores the direction of the previous move along the border from the previous border pixel to the current border pixel. Assign

   (a) dir = 0 if the border is detected in 4-connectivity (*figure 4.5 (a)*)
   (b) dir = 7 if the border is detected in 8-connectivity (*figure 4.5 (b)*)

2. Search the 3x3 neighbourhood of the current pixel in an anti-clockwise direction, beginning from the pixel position in the direction

   (a) (dir + 3) mod 4 (*figure 4.5 (c)*)
   (b) (dir + 7) mod 8 if dir is even (*figure 4.5 (d)*)
   (dir + 6) mod 8 if dir is odd (*figure 4.5 (e)*)

   The first pixel found with the same value as the current pixel is a new boundary pixel P_n. Update the dir value.

3. If the current boundary pixel P_n is equal to the second border pixel P_1, and if the previous border pixel P_{n-1} is equal to P_0, stop. Otherwise repeat step 2.

4. The detected inner border is represented by pixels $P_0 \ldots P_{n-2}$

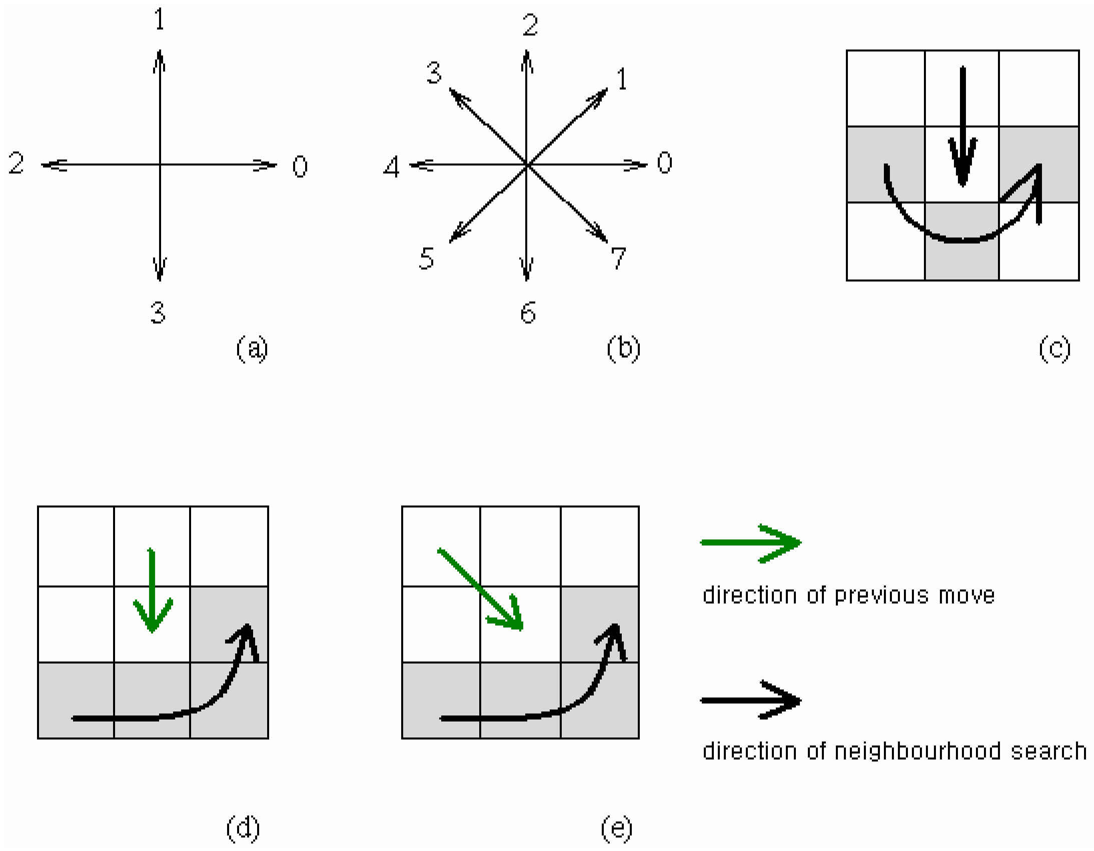


Figure 4.5: Inner boundary tracing

The above algorithm works for all regions larger than one pixel. It is able to find region borders but does not find borders of region holes. To search for hole borders as well, the border must be traced starting in each region or hole border pixel if the pixel has never been a member of any border previously traced. Objects of unit width may have border pixels repeated twice.

Outer boundary tracing may be done using the above algorithm with 4-connectivity. In this case, the outer boundary consists of all non-region pixels that were tested during the search process. So pixels are listed as many times as they are tested. This results in a situation where pixels may be repeated in the border up to three times as shown in the figure below.

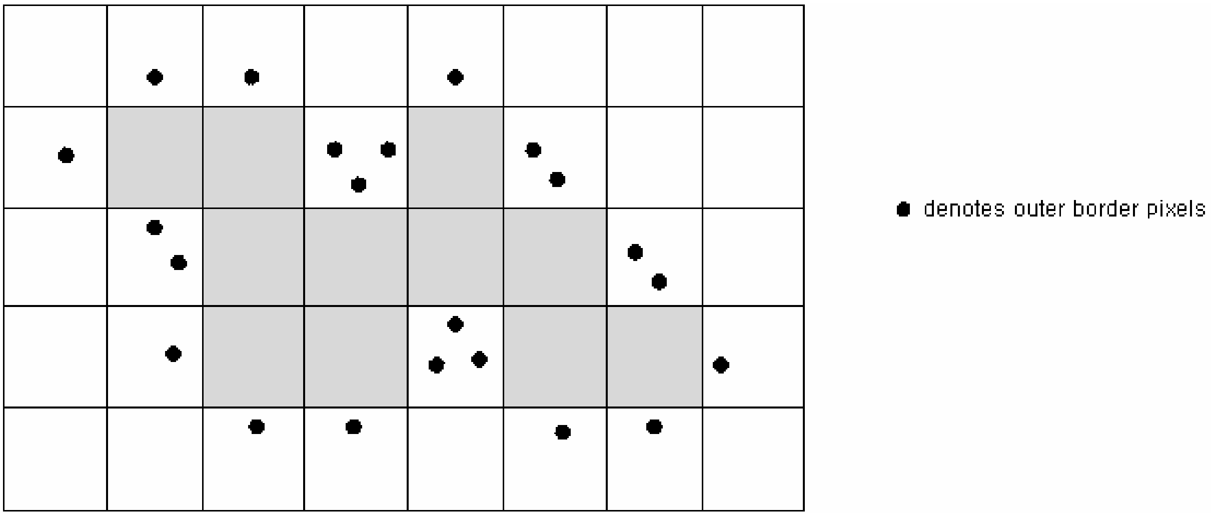


Figure 4.6: Outer boundary tracing

MATLAB provides a function *bwboundaries()* which computes the internal boundaries of the objects in a binary image. The following figure shows the boundaries of the two reflex marks of a robot traced by *bwboundaries()*.

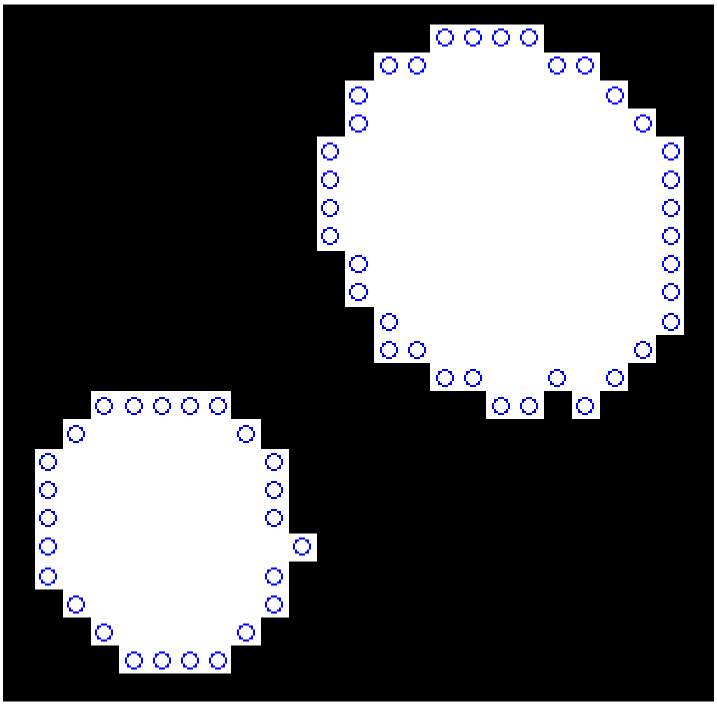


Figure 4.7: Boundary tracing by MATLAB function *bwboundaries*

Chapter 5: Data Processing

The data acquired from the camera image after image processing (i.e. the coordinates of the object boundaries in the binary image) have to be processed in some way to obtain the centres of the reflex marks. Here sub-pixel accuracy comes into play. This will be achieved by finding the best circle fit using the least squares method. The centre of the circle is then taken to be the centre of the reflex mark. Due to different angles of incidence from different points on the operation field, circular reflex marks far away from the centre of the image may be projected as ellipses. For this reason, an ellipse fit still using the least squares method will also be discussed in this chapter.

Performance and measurement analysis of the circle and ellipse fit methods was not done in this project due to time constraints. The analyses results obtained in the Master thesis of Wang & Ye [8] were assumed. There, three methods were compared; region properties and circle and ellipse fit. Circle and ellipse fit were found to take mush less time than region properties. The centre of reflex marks at the centre and at the corners of the operation field were measured and compared. The measured centre points using ellipse and circle fit differed with about 0.02 pixels at the corner of the operation field and 0.01 pixels at the centre. Therefore a detailed analysis was made only for the circle fit because it is less complex and a little faster. An overall accuracy of 0.1 pixel (sub-pixel accuracy) was obtained. This corresponds to about 0.5mm in the operation field which is far better than pixel accuracy which corresponds to about 5mm.

5.1 Sub-Pixel Accuracy:

This discussion on sub-pixel accuracy was taken from [16]. Sub-pixel accuracy in the case of this project can roughly be defined as: “Only pixels whose centre points lie inside a reflex mark appear in the image as part of the reflex mark.” Sub-pixel accuracy arises due to the nature of digital (pixel) displays. Every camera has a fixed resolution (640x480 in this project). Taking into consideration the plain projected onto the camera, a pixel corresponds to about 5mm. Therefore to obtain accurate results, sup-pixel accuracy is inevitable in this project. For certain applications with higher resolutions, say 1280x1024, sub-pixel accuracy may be evitable, if the projected plane is not too far away from the camera as compared to its focal length. Nevertheless, it could still be achieved.

5.2 Least Squares Method:

Least squares method describes a way to obtain the best approximation of a function $f(x)$ given a set of n experimentally measured data points (x_i,y_i), $1 \le i \le n$, where x is the independent variable and y the dependent variable. The best approximation of $f(x)$ is gotten by minimising the objective function F, defined as the sum of the squares of the deviations d_i of each data point, where

$$d_i = y_i - f(x_i),\ 1 \le \mathrm{i} \le n$$

and

$$F = \sum_{i=1}^{n} d_i^{\,2} = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Since the reflex marks of the robots are cut in circular shapes which may be projected as ellipses, circle and ellipse fit will be discussed next.

5.2.1 Circle Fit:

A circle in 2-dimensional space can be defined by its centre (x_0,y_0) and radius r as follows:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \qquad \textit{(5.1)}$$

Expanding the above equation yields an equation of the following form:

$$x^2 + y^2 + ax + by + c = 0 \qquad \textit{(5.2)}$$

where

$$x_0 = - a / 2$$

$$y_0 = - b / 2 \qquad \textit{(5.3)}$$

$$r = \sqrt{x_0^2 + y_0^2 - c}$$

With a set of N data points obtained from boundary tracing, the following over determined system of N linear equations can be obtained from *5.2*:

$$ax_i + by_i + c = - x_i^2 - y_i^2, \; 1 \leq i \leq N \qquad \textit{(5.4)}$$

A matrix representation of the above system of linear equations can be written in the following form:

$$A\vec{z} = \vec{b} \qquad \textit{(5.5)}$$

where A is an Nx3 matrix, $\vec{z}$ a 3-dimensional column vector and $\vec{b}$ an N-dimensional column vector defined as follows;

$$A = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ . & . & . \\ . & . & . \\ . & . & . \\ x_N & y_N & 1 \end{bmatrix}$$

$$\vec{z} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad \textit{(5.6)}$$

$$\vec{b} \quad \begin{bmatrix} -x_1^2 - y_1^2 \\ -x_2^2 - y_2^2 \\ . \\ . \\ -x_N^2 - y_N^2 \end{bmatrix}$$

It is now left to determine the best approximation of the coefficients a, b and c, i.e. the vector $\vec{z}$. This can be done by applying least squares method to *5.5* which will result in the following matrix operations:

$$\vec{z} = (A^T A)^{-1}(A^T \vec{b}) \qquad \textit{(5.7)}$$

The centre and radius of the circle can then be calculated from a, b and c using *5.3*. The function *circle_fit()* (see appendix) was developed to accomplish this task. The following figure shows an example of circle fit.

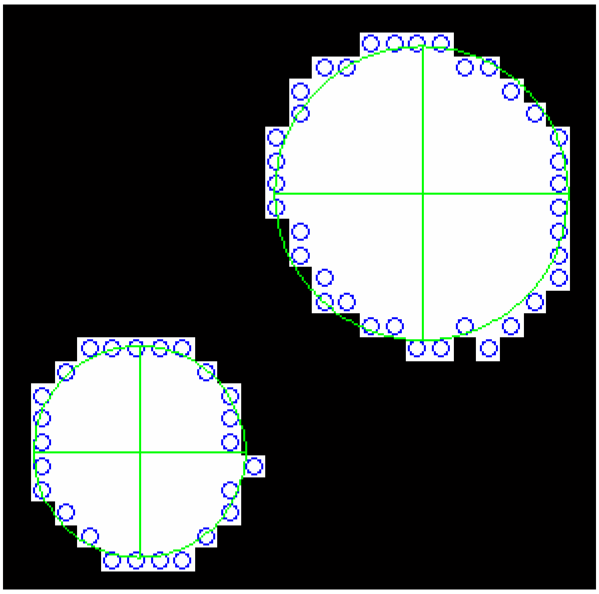


Figure 5.1: Circle Fit

5.2.2 Ellipse Fit:

Consider an ellipse with centre (x_0,y_0) and tilt α measured anti-clockwise from the positive x-axis as show in the following figure.

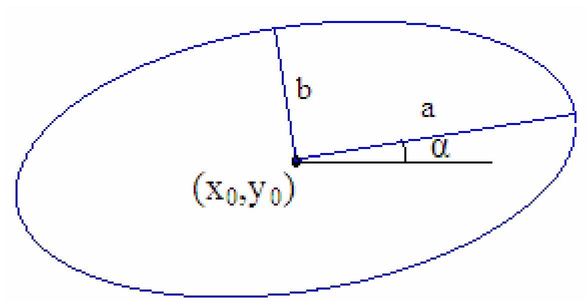


Figure 5.2: Ellipse

The general equation of the above ellipse can be expressed as:

$$Ax^2 + Bx^2 + Cxy + Dx + Ey + 1 = 0 \qquad \textit{(5.8)}$$

If the tilt $\alpha = 0$, then $C = 0$. The ellipse above could be rotated clockwise about the origin by an angle α to obtain an ellipse with coordinates (u,v) and centre (u_0,v_0) with a tilt of zero to the u-axis. Then

$$\begin{aligned} x &= cu - sv \\ y &= sx + cv \qquad \textit{(5.9)} \end{aligned}$$

where $c = \cos\alpha$ and $s = \sin\alpha$. The above equations could be substituted in *5.8* to obtain the following general equation in terms of u and v;

$$Pu^2 + Qv^2 + Ruv + Su + Tv + 1 = 0 \qquad \textit{(5.10)}$$

where

$$\begin{aligned} P &= Ac^2 + Bs^2 + Ccs \\ Q &= As^2 + Bc^2 - Ccs \\ R &= (B - A)\sin(2\alpha) + C\cos(2\alpha) \qquad \textit{(5.11)} \\ S &= Dc + Es \\ T &= Es - Ds \end{aligned}$$

Since the tilt of the ellipse in *5.10* is equal to zero, then $R = 0$. Therefore α can be computed from *5.11* in the following way:

$$\alpha = \frac{1}{2} \arctan\left(\frac{C}{A - B}\right) \qquad \textit{(5.12)}$$

The ellipse in *5.10* could also be expressed in terms of its centre (u_0,v_0) as follows:

$$\frac{(u - u_0)^2}{a^2} + \frac{(v - v_0)^2}{b^2} = 1 \qquad \textit{(5.13)}$$

The above equation could be expanded to give the general equation as in *5.10* and coefficients compared. The centre (u_0,v_0) of the ellipse above and the length of its axes a and b could then be computed from the coefficients in *5.10* as follows:

$$\begin{aligned} u_0 &= -0.5S / P \\ v_0 &= -0.5T / Q \\ a^2 &= (Pu_0^2 + Qv_0^2 - 1) / P \qquad \textit{(5.14)} \\ b^2 &= Pa^2 / Q \end{aligned}$$

The centre (x_0,y_0) of the original ellipse in *5.8* can be computed from (u_0,v_0) using the equations in *5.9*. Its axes are of the same length with that given above. What is now left is to determine an approximation of the coefficients in *5.8*. This is of course accomplished using the least square method. Given a set of N data points that describe the boundary of a reflex mark, the following over determined system of N linear equations can be derived from *5.8*.

$$Ax_i^2 + By_i^2 + Cx_iy_i + Dx_i + Ey_i = -1,\ 1 \leq i \leq N \qquad (5.15)$$

A matrix representation of the above system of linear equations can be written similarly to *5.5* in the following form:

$$H\vec{z} = \vec{b} \qquad (5.16)$$

where H is an Nx5 matrix, $\vec{z}$ a 5-dimensional column vector and $\vec{b}$ an N-dimensional column vector defined as follows:

$$H = \begin{bmatrix} x_1^2 & y_1^2 & x_1y_1 & x_1 & y_1 \\ x_2^2 & y_2^2 & x_2\,y_2 & x_2 & y_2 \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ x_N^2 & y_N^2 & x_Ny_N & x_N & y_N \end{bmatrix}$$

$$\vec{z} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} \qquad (5.17)$$

$$\vec{b} = \begin{bmatrix} -1 \\ -1 \\ . \\ . \\ . \\ -1 \end{bmatrix}$$

Least squares method can be applied similarly to *5.7* to obtain the best approximation of the coefficients A, B, C, D and E, i.e. the vector $\vec{z}$, as follows:

$$\vec{z} = (H^TH)^{-1}(H^T\vec{b}) \qquad (5.18)$$

The above algorithm is implemented by the function *ellipse_fit()* (see appendix). The following figure shows an example of ellipse fit.

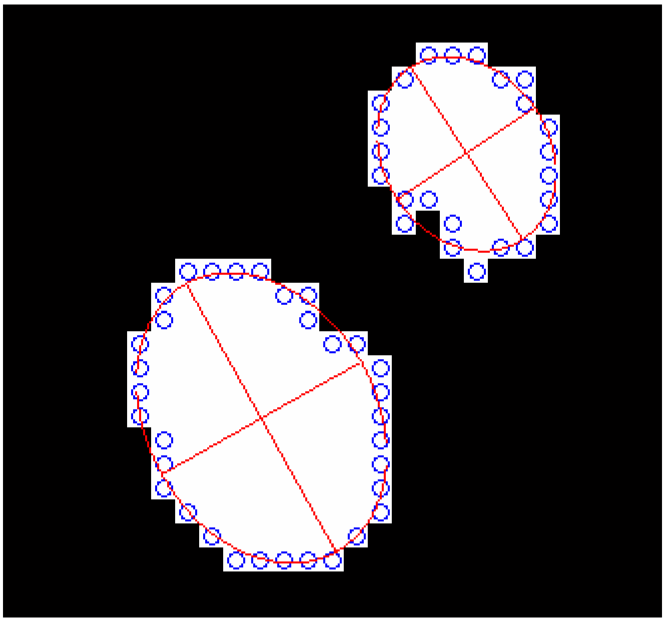


Figure 5.3: Ellipse fit

5.3 Additional Plausibility Checks:

For the circle fit, three coefficients are compute whereas for the ellipse fit, five are computed. At least as many points as there are unknown coefficients are needed for the circle and ellipse fit. If this condition is not satisfied, an underdetermined system of linear equations is obtained. This will cause the least squares method to fail. So an additional check is done in the functions which compute circle and ellipse fit. If there are not enough points, an error message is issued.

Due to variations in luminance, certain background pixels may have the same intensity as a reflex mark or close to it. This may produce a noisy binary image after thresholding as shown in the figure below. When boundary tracing is applied to this image, the noise in the background is also identified in addition to the reflex marks as objects and their boundaries are also traced.

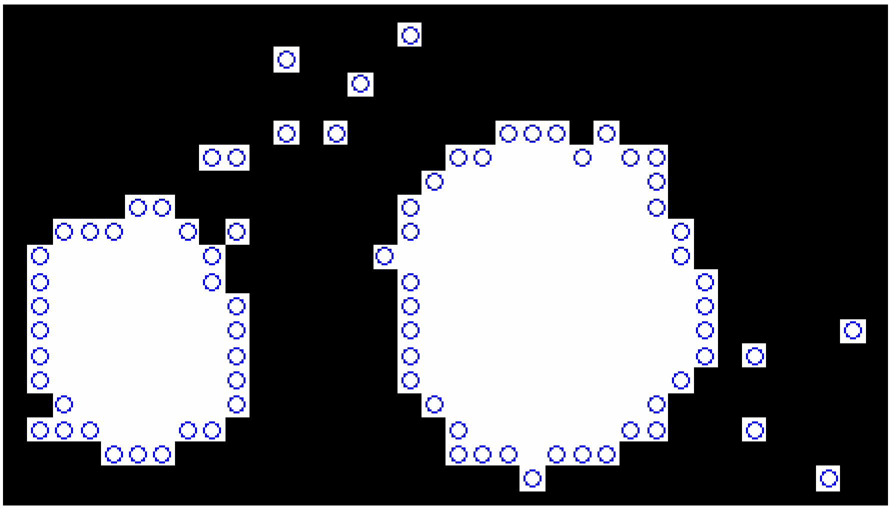


Figure 5.4: Noisy Image

Such noise as shown in the above figure could be eliminated by making sure that the length of each set of boundaries corresponds to a range of values defined by the circumferences of the reflex marks. The circle fit is then applied only to sets which fall in the defined range. It may nevertheless still occur that a set of boundaries describing a curve fits into the defined range. Thus, an additional check is made after circle fit. This time, the radius of each circle must be in a range defined by the radii of the reflex marks. Below is the complete algorithm used to process the images in this project.

1. Threshold the image to obtain a binary image in which the objects are white and the background is black.
2. Trace the boundaries of the objects in the binary image.
3. Loop through the set of boundaries and check the following;
   (a) If a set of boundaries corresponds to a defined range, proceed to (b) else ignore the set.
   (b) Compute the centre and radius of the set of boundaries using circle fit.
   (c) If the radius of the circle is in a defined range, then it corresponds to a reflex mark else it does not and the circle is as well ignored.

The function *get_reflex_marks()* (see appendix) implements step 2 and 3 of the above algorithm.

5.4 Projection Correction:

The reflex marks of the robot do not appear in the image at their right positions, even after calibration. They are radially projected to the image plane as depicted in *figure 5.4*. H is the height of the camera with reference to the operation field, h is the height of the robot, O is the foot of the perpendicular from the camera to the operation field, P is the actual position of the robot and P_c is the radial projection of the reflex mark on top of the robot, gotten from the image after calibration.

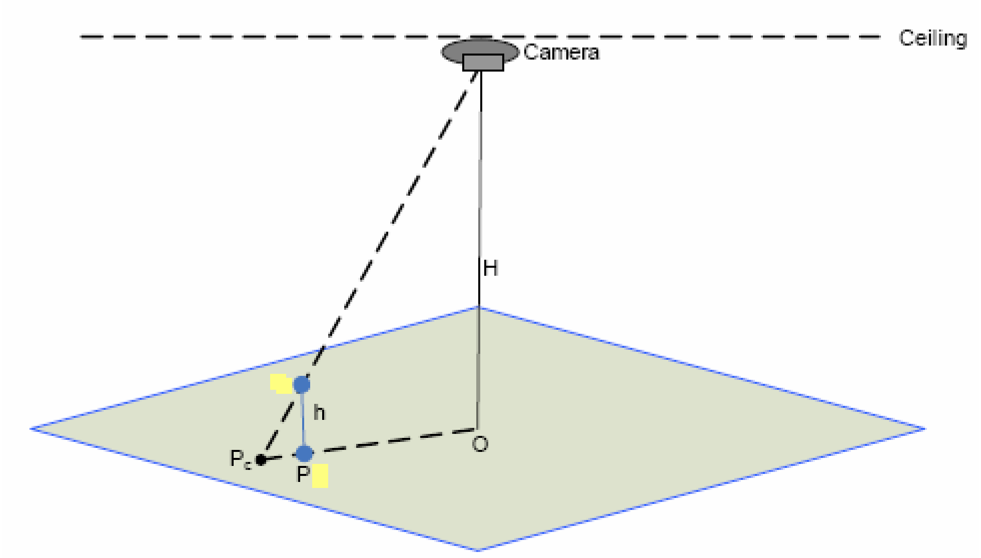


Figure 5.5: Projection of robot reflex mark on operation field

From the geometrical properties of triangles, the following relationship can be deduced:

$$\frac{h}{H} = \frac{|P_cP|}{|P_cO|}$$

Let $P_c = (x_c, y_c)$, $P = (x, y)$ and $O = (x_0, y_0)$. Then a unit vector in the direction P_cP can be expressed as follows:

$$u = \frac{1}{|P_cO|} [(x_0 - x_c) i + (y_0 - y_c) j]$$

where i and j are unit vectors in the x and y directions respectively. P_cP can be expressed in terms of the above unit vector as follows;

$$\begin{aligned} P_cP = [(x - x_c) i + (y - y_c) j] &= |P_cP| u \\ &= \frac{|P_cP|}{|P_cO|} [(x_0 - x_c) i + (y_0 - y_c) j] \\ &= \frac{h}{H} [(x_0 - x_c) i + (y0 - y_c) j] \end{aligned}$$

By comparing the coefficients of i and j in the above equation, the actual position of the robot given by P with coordinates (x,y) can be computed as follows;

$$x = x_c + \frac{h}{H} (x_0 - x_c) \qquad \& \qquad y = y_c + \frac{h}{H} (y_0 - y_c)$$

The robot height h is 165mm and the values of H and O measured for camera 1 and 2 are given below;

Camera 1:
H = 2850mm
O = (3555, 945)
Camera 2:
H = 2840mm
O = (1300, 1000)

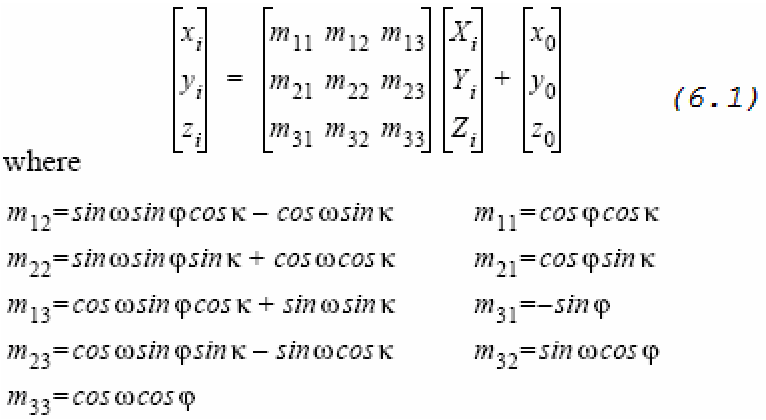
Chapter 6: Camera Calibration

The camera calibration procedures discussed here are taken from an article by Heikkilä J. and Silvén O. [1]. Camera calibration estimates a set of camera parameters. These parameters are used to establish the relationship between 3-D reference coordinates in the camera plane and 2-D image coordinates in the image plane. The parameters are divided into two classes: intrinsic and extrinsic parameters. Two methods for calibrating a camera will be discussed here: direct linear transformation (DLT) and nonlinear parameter estimation. Image distortion and possible calibration errors and how they can be corrected will also be discussed and taken into account.

6.1 Extrinsic Parameters:

These parameters determine the 3-D position and orientation of the camera frame relative to a chosen world coordinate system, i.e. they transform object coordinates to a camera centred coordinate frame. The pinhole camera model is based on the principle of co-linearity, where each point in the object space is projected by a straight line through the projection centre into the image plane. The origin of the camera frame is at the projection centre (X_0, Y_0, Z_0) with respect to the object space and the z-axis is perpendicular to the image plane.

The transformation of the coordinates (X_i, Y_i, Z_i) in the image space to the coordinates (x_i, y_i, z_i) in the camera frame is given by a triple rotation and a translation. The rotation is performed in the clockwise direction first about the x-axis by an angle ω, then about the y-axis by an angle φ and finally about the z-axis by an angle κ. A matrix representation of this transformation is given below.



$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \qquad (6.1)$$

where

$m_{12} = \sin\omega \sin\varphi \cos\kappa - \cos\omega \sin\kappa$ $\qquad m_{11} = \cos\varphi \cos\kappa$

$m_{22} = \sin\omega \sin\varphi \sin\kappa + \cos\omega \cos\kappa$ $\qquad m_{21} = \cos\varphi \sin\kappa$

$m_{13} = \cos\omega \sin\varphi \cos\kappa + \sin\omega \sin\kappa$ $\qquad m_{31} = -\sin\varphi$

$m_{23} = \cos\omega \sin\varphi \sin\kappa - \sin\omega \cos\kappa$ $\qquad m_{32} = \sin\omega \cos\varphi$

$m_{33} = \cos\omega \cos\varphi$

6.2 Intrinsic Parameters:

Intrinsic parameters determine the internal camera geometric and optical characteristics. These include the effective focal length f, scale factors s_u and image centre (u_0, v_0), also called principal point. As usual for digital images, the origin of the image space is at the top left corner of the image array. The units of the image coordinates are pixels. Therefore coefficients D_u and D_v are needed to change the metric units to pixels. Their precise values are not necessary because they are linearly dependent on the focal length and scale factors. By using the pinhole model, the projection of the point (x_i, y_i, z_i) on the image plane is expressed in the following equation.

$$\begin{bmatrix}\tilde{u}_i\\ \tilde{v}_i\end{bmatrix} = \frac{f}{z_i}\begin{bmatrix}x_i\\ y_i\end{bmatrix} \qquad (6.2)$$

The corresponding image coordinates (u_i', v_i') in pixels are obtained from the projection coordinates given in the above equation by applying the transformation given below.

$$\begin{bmatrix}u_i'\\ v_i'\end{bmatrix} = \begin{bmatrix}D_u s_u \tilde{u}_i\\ D_v \tilde{v}_i\end{bmatrix} + \begin{bmatrix}u_0\\ v_0\end{bmatrix} \qquad (6.3)$$

The pinhole model is a linear approximation of the real camera projection. It simply gives a mathematical relationship between object and image coordinates. It is however invalid if high accuracy is required as is the case in this project. Therefore a more accurate camera model must be used which takes into consideration the distorted image coordinates.

6.3 Image Distortion:

The most common distortion taken into consideration is radial distortion which causes the actual image points to be displaced radially in the image plane. The radial distortion can be corrected using the expression as shown below.

$$\begin{bmatrix}\delta u_i^{(r)}\\ \delta v_i^{(r)}\end{bmatrix} = \begin{bmatrix}\tilde{u}_i(k_1 r_i^2 + k_2 r_i^4 + \ldots)\\ \tilde{v}_i(k_1 r_i^2 + k_2 r_i^4 + \ldots)\end{bmatrix} \qquad (6.4)$$

where k_1, k_2, k_3, … are coefficients for radial distortion and $r_i = \sqrt{\tilde{u}_i^{\,2} + \tilde{v}_i^{\,2}}$. Typically, one or two coefficients are enough to compensate the radial distortion.

The centre of curvature of a lens is not always strictly co-linear because of imperfect lens design and manufacturing. This gives rise to decentring distortion which has both a radial and a tangential component. The expression for the tangential distortion is given below.

$$\begin{bmatrix}\delta u_i^{(t)}\\ \delta v_i^{(t)}\end{bmatrix} = \begin{bmatrix}2p_1\tilde{u}_i\tilde{v}_i + p_2(r_i^2 + 2\tilde{u}_i^{\,2})\\ p_1(r_i^2 + 2\tilde{v}_i^{\,2}) + 2p_2\tilde{u}_i\tilde{v}_i\end{bmatrix} \qquad (6.5)$$

where p_1 and p_2 are coefficients for tangential distortion.

There are of course other types of distortions found in the literature which may be caused by luminance, the medium in which light is propagated or when the image axes are not perpendicular. These distortions are however so insignificant and will not be discussed here. A proper camera model can now be derived from a combination of the pinhole model with the correction for radial and tangential distortion components as shown in the equation below.

$$\begin{bmatrix}u_i\\ v_i\end{bmatrix} = \begin{bmatrix}D_u s_u(\tilde{u}_i + \delta u_i^{(r)} + \delta u_i^{(t)})\\ D_v(\tilde{v}_i + \delta v_i^{(r)} + \delta v_i^{(t)})\end{bmatrix} + \begin{bmatrix}u_0\\ v_0\end{bmatrix} \qquad (6.6)$$

In this model, the set of intrinsic parameters (f, s_u, u_0, v_0) is augmented with the distortion coefficients $k_1,\ldots,\ k_n$, p_1 and p_2. These parameters are also collectively known as physical camera parameters. Generally, the objective of camera calibration is to determine optimal values for these parameters based on image observations of known 3-D targets.

6.4 Linear Parameter Estimation:

The direct linear transformation (DLT) was originally developed by Abdel-Aziz and Karara [2]. It was revised later in several publications as well.

The DLT method is based on the pinhole camera model (see *6.3*), and it ignores the nonlinear radial and tangential distortion components. The calibration procedure consists of two steps. In the first step the linear transformation from the object coordinates (X_i, Y_i, Z_i) to image coordinates (u_i, v_i) is solved. The 3x4 matrix **A** presented in the equation below describes this transformation.

$$\begin{bmatrix} u_i w_i \\ v_i w_i \\ w_i \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \qquad (6.7)$$

where w_i is the axis perpendicular to the image plane. The parameters $a_{11},\ldots,a_{34}$ of the DLT matrix **A** can be solved by eliminating w_i. Let **L** and **a** be defined as follows:

$$\mathbf{L}=\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -X_1u_1 & -Y_1u_1 & -Z_1u_1 & -u_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -X_1v_1 & -Y_1v_1 & -Z_1v_1 & -v_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -X_iu_i & -Y_iu_i & -Z_iu_i & -u_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -X_iv_i & -Y_iv_i & -Z_iv_i & -v_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_N & Y_N & Z_N & 1 & 0 & 0 & 0 & 0 & -X_Nu_N & -Y_Nu_N & -Z_Nu_N & -u_N \\ 0 & 0 & 0 & 0 & X_N & Y_N & Z_N & 1 & -X_Nv_N & -Y_Nv_N & -Z_Nv_N & -v_N \end{bmatrix}$$

$$\mathbf{a} = [a_{11}, a_{12}, a_{13}, a_{14}, a_{21}, a_{22}, a_{23}, a_{24}, a_{31}, a_{32}, a_{33}, a_{34}]^T$$

A matrix equation with N control points can be obtained as given below;

$$\mathbf{La = 0} \qquad (6.8)$$

By replacing the correct image points (u_i, v_i) with observed values (U_i, V_i) parameters $a_{11},\ldots,a_{34}$ can be estimated using the least squares method. In order to avoid a trivial solution $a_{11} = \ldots = a_{34} = 0$, a proper normalization must be applied. Abdel-Aziz and Karara [2] used the normalization $a_{34} = 1$. A singularity is introduced here with this normalization if the actual value of a_{34} is close to zero. Instead of $a_{34} = 1$, Faugeras and Toscani [3] suggested the constraint $a_{31}^2 + a_{32}^2 + a_{33}^2 = 1$ which is singularity free.

The estimation of the parameters $a_{11},\ldots,a_{34}$ is the first calibration step. These parameters in themselves have no physical meaning. There are several techniques for extracting the physical camera parameters from the DLT matrix **A**. Melen [4] proposed a method where a set of eleven physical camera parameters are extracted from the DLT matrix. The matrix equation used by Melen is given in the following equation.

$$\mathbf{A} = \lambda\mathbf{V}^{-1}\mathbf{B}^{-1}\mathbf{FMT} \qquad \textit{(6.9)}$$

where **λ** is an overall scaling factor (Von-Bold constant) and the matrices **M** and **T** define the rotation and translation from the object coordinate system to the camera coordinate system (see *6.1*). Matrices **V**, **B** and **F** contain the focal length f, principal point (u_0, v_0) and the coefficients for the linear distortion (b_1, b_2). **V**, **B** and **F** are defined as follows:

$$\mathbf{V} = \begin{bmatrix} 1 & 0 & -u_0 \\ 0 & 1 & -v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1+b_1 & b_2 & 0 \\ b_2 & 1-b_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A five step algorithm for solving the parameters is given in [3] and would not be discussed here. In this algorithm, the scale factors s_u and s_v are assumed to be 1.

The problem with the linear parameter estimation is that it does not take any distortions into consideration and therefore it is not accurate enough for this project. The nonlinear parameter estimation discussed next comes closer to the target.

6.5 Nonlinear Parameter Estimation:

Linear methods of parameter estimation are comparably faster than nonlinear methods because they involve no iterations. They however have two great disadvantages. Firstly, distortion effects are not taken into consideration and secondly, the actual constraints in the intermediate parameters used to compute the actual parameters can not be fixed. Consequently, the lens distortion can not be incorporated into the calibration procedure and in the presence of noise, the intermediate solutions do not satisfy the constraints. This makes the non iterative linear methods not accurate enough for this project.

In the real world, camera images are always contaminated by white noise (mostly caused by improper illumination of electromagnetic interferance). There are also various error components incorporated in the measurement process. If the systematic parts of the error measurements are carefully minimised or compensated for, it is convenient to assume that the error is white Gaussian noise. Then, the best estimate for the camera parameters can be obtained by minimizing the residual between the model and N observations (U_i, V_i), where i = 1, 2,…, N. In the case of Gaussian noise, the objective function F is expressed as a sum of squared residues as given in the equation below.

$$F = \sum_{i=1}^{N} (U_i - u_i)^2 + \sum_{i=1}^{N} (V_i - v_i)^2 \qquad (6.10)$$

The least squares method can be used to minimise F. Due to a nonlinear nature of the camera model, simultaneous parameter estimation has to be applied. This involves an iterative algorithm. The Levenberg-Marquardt optimization method has been found to provide fastest convergence. Details of this method can be found in the internet [4]. However, this method works only with proper initialization of the camera parameters. Without proper initialization, the optimization may stick in a local minimum of *F* causing the calibration to fail. This problem can be solved by using a DLT method to initialize the camera parameters. A global minimum of *F* is usually achieved after a few iterations.

6.6 Calibration Function go_calib_optim():

This function is found in the MATLAB calibration toolbox. It computes the final intrinsic and extrinsic calibration parameters of a camera by minimizing the re-projection error in the least squares sense trough gradient descent. The following table shows the descriptions of its input and output.

- M is the number of calibration images used.
- i = 1, 2, … , M is the i-th image.
- N_i is the number of control points in the i-th image.

	Parameter	Type	Description
INPUT	x_1, x_2,…, x_M	3xN_i matrix	Feature locations on the images
	X_1, X_2,…, X_M	3xN_i matrix	Corresponding grid coordinates
OUTPUT (intrinsic parameters)	fc	2x1 vector	Camera focal length
	cc	2x1 vector	Principal point coordinates
	alpha_c	double	Skew coefficient
	kc	5x1 vector	Distortion coefficients
OUTPUT (extrinsic parameters)	KK	3x3 matrix	The camera matrix (contains fc and cc)
	Tc_1,Tc_2,…,Tc_M	3x1 matrix	3D translation vectors attached to the grid points in space
	Rc_1,Rc_2,…,Rc_M	3x3 matrix	3D rotation matrices attached to the grid points in space

Table 6.1: Description of parameters of function *go_calib_optim()*

6.6.1 Definition of the Intrinsic Parameters:

Let **P** be a point in space of coordinate vector **XX_c = [X_c;Y_c;Z_c]** in the camera reference frame. Let the point now be projected on the image plane according to the intrinsic parameters (**fc**,**cc**,**alpha_c**,**kc**). Let **x_n** be the normalized (pinhole) image projection:

$$x_n = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Let $\mathbf{r^2} = \mathbf{x^2} + \mathbf{y^2}$. After including lens distortion, the new normalized point coordinate **x_d** is defined as follows:

$$x_d = \begin{bmatrix} x_d(1) \\ x_d(2) \end{bmatrix} = \left(1 + kc(1)\,r^2 + kc(2)\,r^4 + kc(5)\,r^6\right)x_n + dx$$

where **dx** is the tangential distortion vector:

$$dx = \begin{bmatrix} 2\,kc(3)\,x\,y + kc(4)\left(r^2 + 2x^2\right) \\ kc(3)\left(r^2 + 2y^2\right) + 2\,kc(4)\,x\,y \end{bmatrix}$$

Therefore, the vector **kc** of length 5 contains both radial and tangential distortion coefficients (observe that the coefficient of 6th order radial distortion term is the fifth entry of the vector **kc**). It is worth mentioning that this distortion model was first introduced by Brown in 1966 and called "Plumb Bob" model (radial polynomial + "thin prism"). The tangential distortion is due to "decentring", or imperfect centring of the lens components and other manufacturing defects in a compound lens. For more details, refer to Brown's original publications listed in [7]. Once distortion is applied, the final pixel coordinates **x_pixel = [x$_p$;y$_p$]** of the projection of **P** on the image plane is:

$$\begin{cases} x_p = fc(1)\left(x_d(1) + alpha\_c * x_d(2)\right) + cc(1) \\ y_p = fc(2)\,x_d(2) + cc(2) \end{cases}$$

Therefore, the pixel coordinate vector **x_pixel** and the normalized (distorted) coordinate vector **x$_d$** are related to each other through the linear equation:

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = KK \begin{bmatrix} x_d(1) \\ x_d(2) \\ 1 \end{bmatrix}$$

where **KK** is known as the camera matrix, and defined as follows:

$$KK = \begin{bmatrix} fc(1) & alpha\_c * fc(1) & cc(1) \\ 0 & fc(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix}$$

In MATLAB, this matrix is stored in the variable **KK** after calibration. Observe that **fc(1)** and **fc(2)** are the focal distance (a unique value in mm) expressed in units of horizontal and vertical pixels. Both components of the vector **fc** are usually very similar. The ratio **fc(2)/fc(1)**, often called "aspect ratio", is different from 1 if the pixel in the CCD array are not square. Therefore, the camera model naturally handles non-square pixels. In addition, the coefficient **alpha_c** encodes the angle between the x and y sensor axes. Consequently, pixels are even allowed to be non-rectangular. Some authors refer to that type of model as "affine distortion" model.

In addition to computing estimates for the intrinsic parameters **fc**, **cc**, **kc** and **alpha_c**, the toolbox also returns estimates of the uncertainties on those parameters. The MATLAB variables containing those uncertainties are **fc_error**, **cc_error**, **kc_error**, **alpha_c_error**. For information, those vectors are approximately three times the standard deviations of the errors of estimation.

Pixel coordinates are defined such that **[0;0]** is the center of the upper left pixel of the image. As a result, **[nx-1;0]** is center of the upper right corner pixel, **[0;ny-1]** is the center of the lower left corner pixel and **[nx-1;ny-1]** is the center of the lower right corner pixel where **nx** and **ny** are the width and height of the image (for this project, **nx=640** and **ny=480**).

In the first three sections of this chapter, the camera model according to Heikkilä [1] was introduced. There the skew factor (alpha_c) was not estimated but assumed to be zero, i.e. the model is only for cameras with square pixels. The radial component of the distortion was estimated up to the 4th order. Here, it is estimated till the 5th order. The model used here in MATLAB is similar to that of Heikkilä. The table below shows the correspondence between the camera parameters estimated here and those of Heikkilä.

Notations in go_calib_optim()	**Heikkil□'s notation**
fc(1)	**f . Du . su**
fc(2)	**f . Dv**
cc(1)	**u0**
cc(2)	**v0**
alpha_c	**0**
kc(1)	**f^3 . k1**
kc(2)	**f^5 . k2**
kc(3)	**f^2 . p1**
kc(4)	**f^2 . p2**
kc(5)	**0**

Table 6.2: Heikkilä Camera Model vs. MATLAB Camera Model

6.6.2 Definition of the Extrinsic Parameters:

Consider a calibration grid in the camera's view as depicted in the following figure. The reference frame (O,X,Y,Z) of this grid is as shown. It's origin is at the top left corner of the grid, the x- and y-axes lie on the plane of the grid while the z-axis is perpendicular to the plane of the grid.

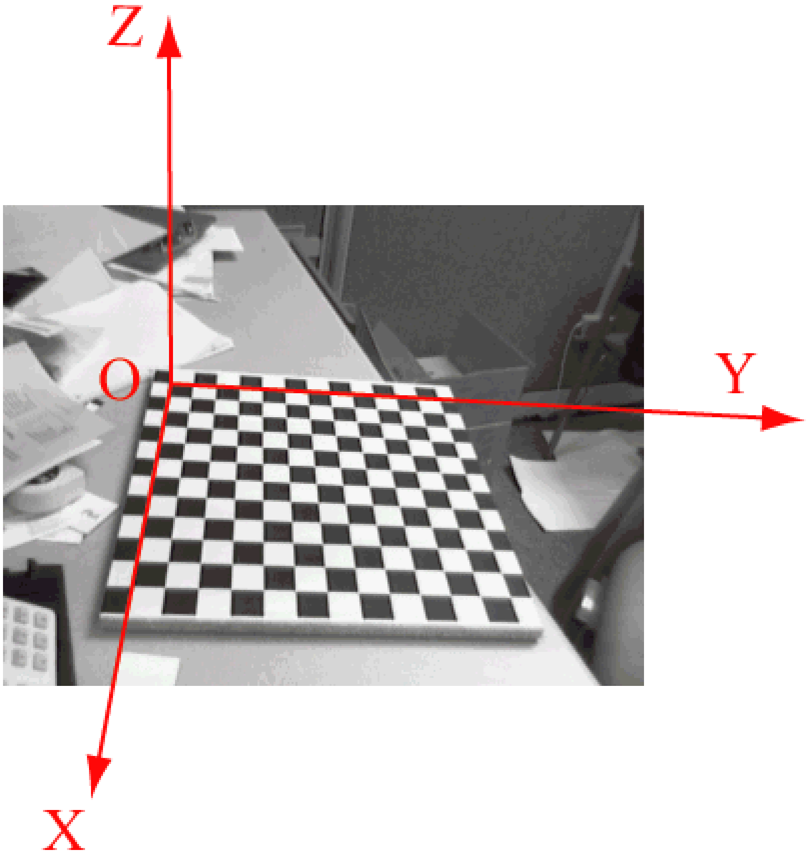


Figure 6.1: Coordinate axes of reference frame for a calibration grid

Let **P** be a point space of coordinate vector **XX = [X;Y;Z]** in the grid reference frame (reference frame shown in figure above). Let $\mathbf{XX_c} = \mathbf{[X_c;Y_c;Z_c]}$ be the coordinate vector of **P** in the camera reference frame. Then **XX** and $\mathbf{XX_c}$ are related to each other through the following rigid motion equation:

$$\mathbf{XX_c = Rc\_i * XX + Tc\_i} \qquad \text{where } i = 1, 2, \ldots, M$$

6.6.3 First Calibration Attempt:

For this project a 1400mm x 1000mm calibration sheet, with reflex marks spaced 200mm form each other, was used. The figure below shows the different positions of the calibration sheet in the view of the camera.

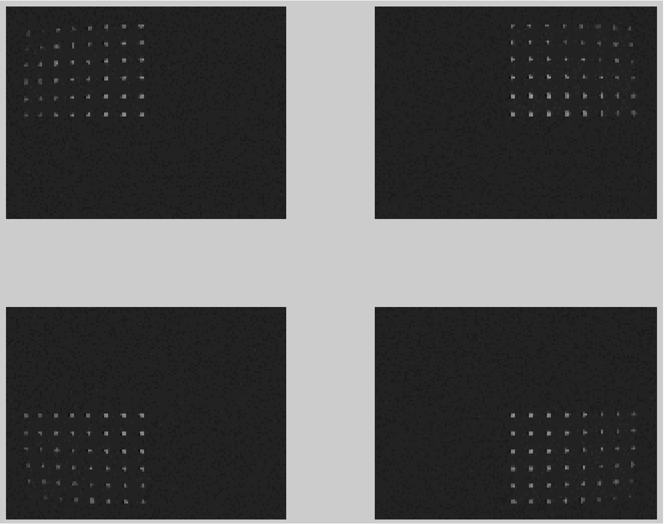


Figure 6.2: Positions of calibration sheet in camera view

The centre points of the reflex marks were extracted (using the image and data processing procedures as described in chapters 4 and 5) and used for calibrating the camera. The following values for the intrinsic parameters of camera 1 and their respective errors were obtained using the function *go_calib_optim()* from the MATLAB calibration toolbox.

Focal Length: fc = [861.72921   861.05362] ± [455.66248   455.54862]
Principal point: cc = [330.39080   240.89507] ± [6.44623   8.54029]
Skew: alpha_c = [0.00000] ± [0.00000]
          => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion: kc = [-0.70114   0.69313   0.00091   -0.00104 0.00000] ± [0.74197   1.47280
                    0.00571   0.00419 0.00000]
Pixel error: err = [0.13253   0.14211]

The focal lengths are given in mm and the principal point in pixels. Notice their high errors. Coming to the distortion coefficients (**kc**), their errors are even higher than their actual values. The pixel error calculated by the calibration function is approximately 0.1 which equals about 0.5mm which should actually be negligible for this project.

Given the pixel coordinates x_pixel of a robot from an image, the actual position of the robot in the operation field can be calculated using the following steps:

- Compute the normalized coordinates x_n (see section 6.6.1) using the MATLAB function *normalize()* as follows:

  x_n = normalize(x_pixel, fc, cc, kc, alpha_c)

- Compute the coordinates of the camera frame XX_c as follows:

$$\mathbf{XX_c} = \mathbf{Z_c} \begin{bmatrix} x_n(1) \\ x_n(2) \\ 1 \end{bmatrix}$$

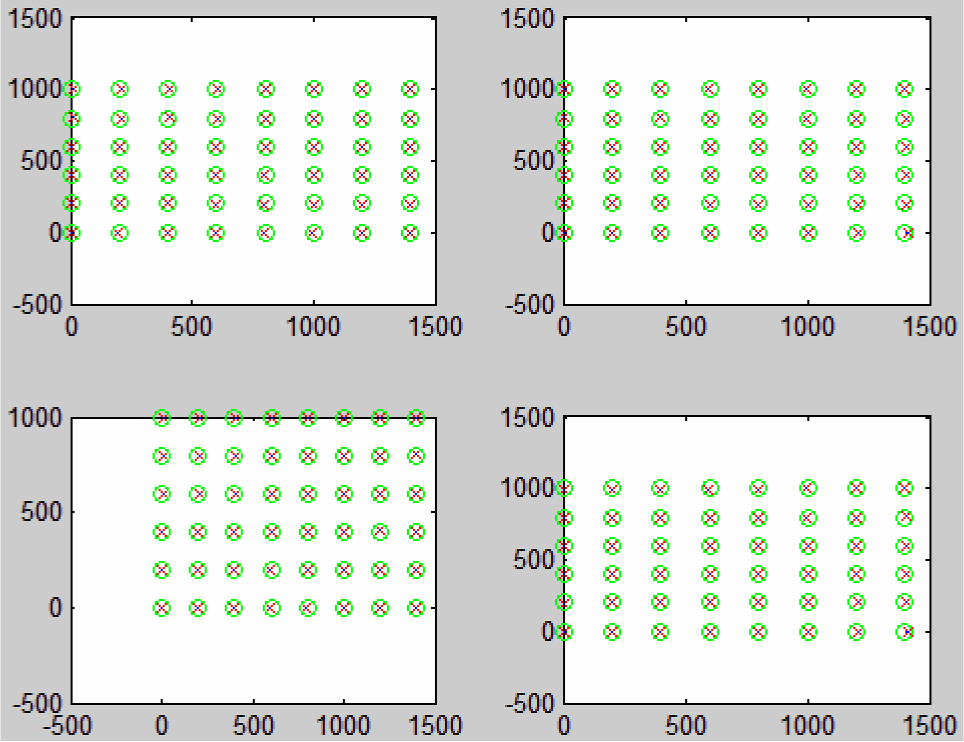
  Z_c can be computed from the extrinsic parameters.

- Use the rotation matrix Rc and translation vector Tc (both extrinsic parameters) of the reference frame of a calibration sheet in which the robot lies and compute the coordinates XX of this reference frame as follows:

$$XX = Rc^{-1} * (XX_c - Tc)$$

The actual coordinates in the operation field can then be calculated from XX since the location of the calibration sheet in the operation field is known.

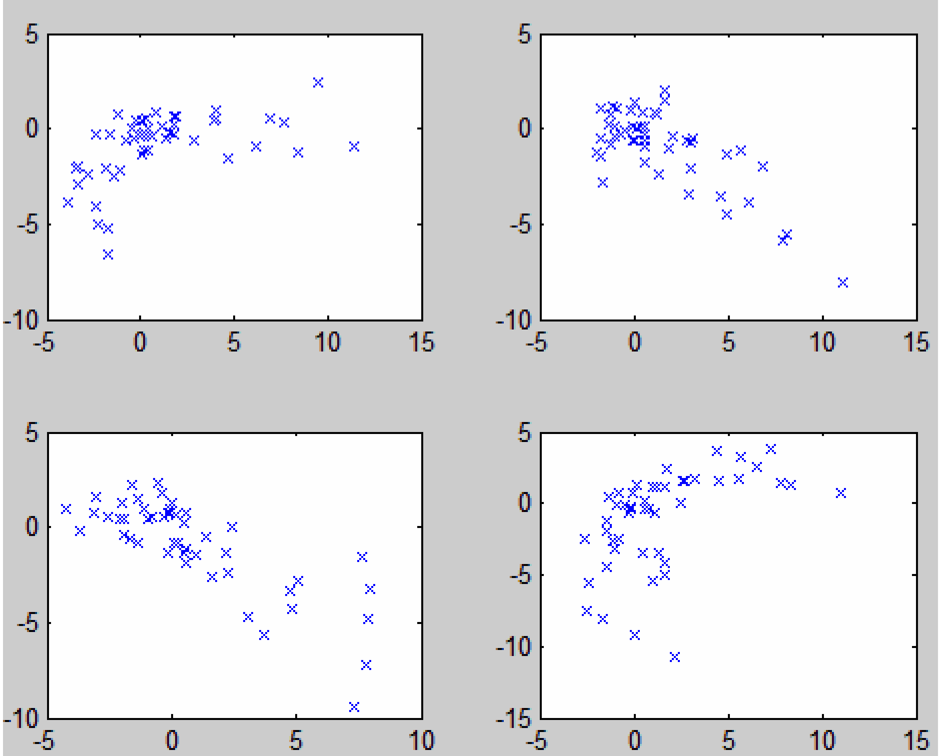
The following figure shows the actual points of the calibration sheets (in mm) for camera 1 with red crosses and the calibrated points obtained from the images with green circles. The radial error is drawn with a blue line.



Vertical axes: y-coordinates in mm
Horizontal axes: x-coordinates in mm

Figure 6.3: Actual and calibrated points for the calibration sheets of camera 1

The following figure shows a plot of only the x and y errors (also in mm) of the above calibration. Here the x and y errors are the differences between the corresponding coordinates of the actual and calibrated points.



Vertical axes: y-errors in mm
Horizontal axes: x-errors in mm

Figure 6.4: x and y calibration errors for the different calibration sheets of camera 1

The above errors show a maximum distortion of about 14mm on the operation field which actually does not correspond to 0.1 pixels as calculated by the calibration function. Another sad news is that the calibration of camera 2 with the MATLAB calibration function failed. The reason for this failure is that a singularity was introduced in the estimation of direct linear transformation matrix as described in section 6.4.

The calibration procedure used in the MATLAB calibration toolbox is a somewhat general case and much too complex for this project. The estimation of the intrinsic camera parameters is strongly dependent on the z-coordinate of the camera frame. This is no surprise because a line in the camera frame is mapped onto a single point in the image. As such a proper excitation of the z-coordinate in the camera frame is needed. This requires calibration points in different planes with different orientations in space. This is actually the contrary in this project. All calibration points lie in a single plane of interest, i.e. the plane in which the operation field lies. As such, a customized adaptive camera calibration procedure was developed for this project.

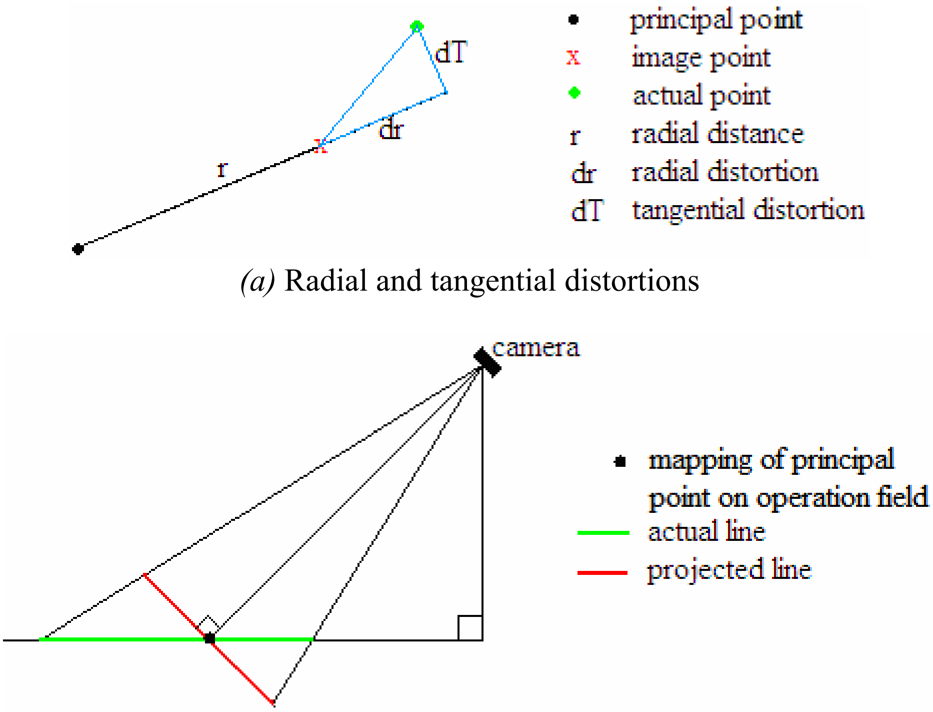
Chapter 7: Customized Adaptive Camera Calibration

The first attempt to calibrate the cameras with the MATLAB calibration toolbox was unsuccessful. As such, a customised adaptive camera calibration procedure was developed for this project to calibrate the cameras. This chapter explains in detail all the assumptions, considerations and steps used in calibrating the cameras for this project.

The camera model assumed here is similar to the pinhole camera model. The possible sources of error and their effects will first be discussed before introducing the calibration parameters. The calibration parameters considered depended on the errors investigated. The same calibration procedure was used for both camera 1 and 2. Below is a summary of the possible error sources and their effects:

1. Improper mounting of the camera chip behind the lens will cause a deviation of the principal point from the centre of the image. This occurs when the camera chip is not parallel to the lens axis.
2. The camera lens introduces radial distortions. These are distortions along the radial distance with reference to the principal point of the lens.
3. If the x- and y-axes of the operation field are not aligned with that of the camera, this will give rise to tangential distortions. These distortions are tangential to circles drawn with their centres at the principal point and are perpendicular to the radial distortions.
4. If the z-axis of the camera frame is not perpendicular to the plane in which the operation field lies, a projection distortion will be noticed.

The following figures show the distortions described above.

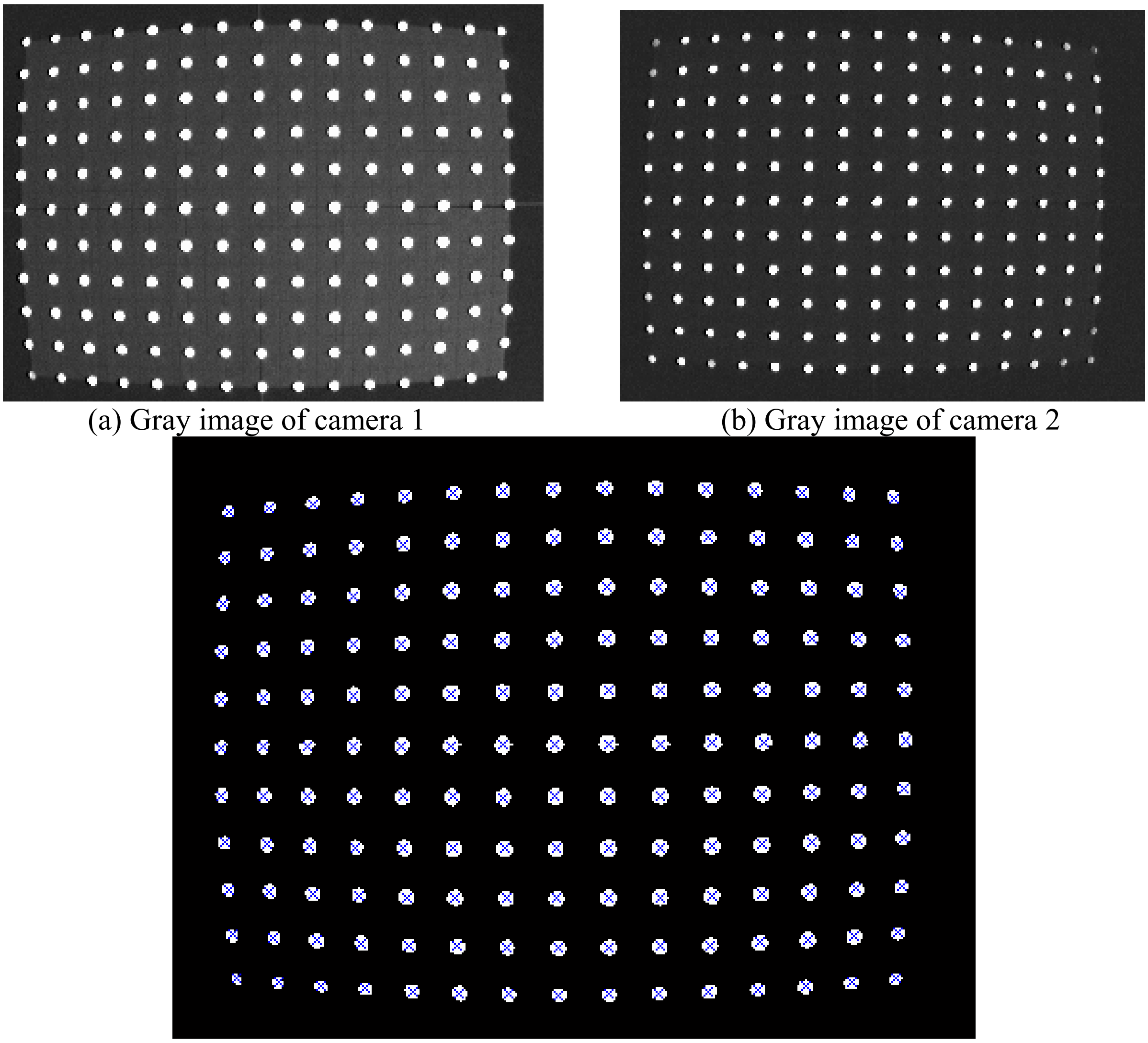


(a) Radial and tangential distortions

(b) Projection distortion

Figure 7.1: Sources of image distortions

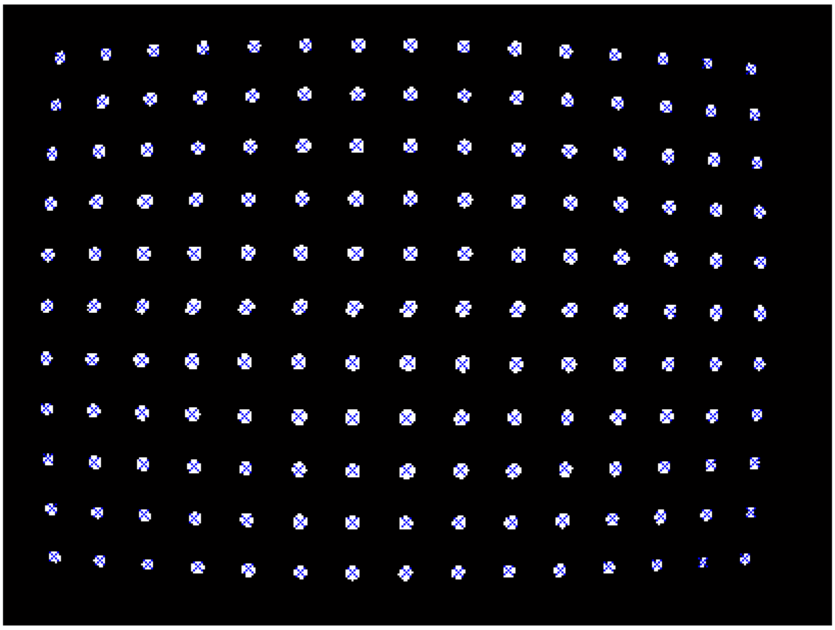
The calibration images from *figure 6.2* were added up to form one image. In order to reduce the background noise in the resulting image, the pixels which do not lie in the area of the image covered by the calibration sheet were set to zero before the images were added. Below are the resulting images for camera 1 and 2 from the addition and their corresponding binary images after thresholding. The centres of the white circles in the binary images were extracted by the boundary tracing and circle fit algorithms described earlier. The centres of the circles are also plotted with blue crosses on the binary image.



(a) Gray image of camera 1

(b) Gray image of camera 2

(c) Binary image of camera 1



(d) Binary image of camera 2

Figure 7.2: Calibration images

The above reflex marks actually form a rectangular grid on the operation field. For camera 1, the top left reflex mark is at the point (2000,0) of the operation field while the bottom right reflex mark is at the point (4800,2000) of the operation field forming a rectangular grid of dimensions 2800x2000. The reflex marks of camera 2 also form a rectangular grid of dimensions 2800x2000, but its top left reflex mark is at the point (0,0) of the operation field while the bottom right is at the point (2800,2000). So there is an overlap of the view of both cameras as depicted in *figure 1.6*. The coordinates and dimensions of the operation field are given in mm. The reflex marks are spaced at distances of 200mm from each other. So the actual coordinates of the reflex mark centres in the operation field are known and shall be denoted by (x_i,y_i), $1 \leq i \leq N$, while their corresponding image points shall be denoted by (u_i,v_i), $1 \leq i \leq N$, where N is the number of calibration points and equals 165 for both cameras.

The calibration procedure presented here is non-linear. It minimises the objective function as defined in *6.10*. A step-by-step calibration procedure shall be presented in this chapter. The distortions discussed above shall be minimised one after the other at each step of the calibration procedure. An overall investigation of the radial and tangential distortions for both cameras shall also be presented at each step, as well as the minimised value of the objective function.

7.1 Extrinsic and Intrinsic Parameters:

In this section, the mapping of points from the operation field to their corresponding undistorted points in the image will be described. The extrinsic parameters describe the transformation from the operation field to the camera frame while the intrinsic parameters describe the mapping from the camera frame to the image. The calibration points in the operation field will be denoted by (x_i,y_i) and are given in mm while their corresponding image points in pixels will be denoted by (u_i,v_i).

Let (u_0,v_0) be the principal point of the lens in pixels and (x_0,y_0) its corresponding point in the camera frame in mm. The following mappings from image points to their corresponding points in the camera frame were used for this project.

$$x_i'' - x_0 = s_u (u_i - u_0 + du_i),\ 1 \le i \le N$$
$$y_i'' - y_0 = s_v (v_i - v_0 + dv_i),\ 1 \le i \le N \qquad (7.1)$$

where s_u and s_v are scale factors in the u- and v-directions respectively, du_i and dv_i are the pixel distortions in the u- and v-directions respectively, (x_i'',y_i'') are points in the camera frame in mm and (u_i,v_i) are their corresponding distorted image points in pixels. The intrinsic parameters used in this calibration are the principal point (u_0,v_0) in pixels, its mapping in the operation field (x_0, y_0) in mm and the scale factors s_u and s_v.

In this project, it was assumed that the camera chips were mounted correctly, i.e. the principal point (u_0,v_0) is at the centre (320,240) of the image. The corresponding principal point in the operation field and the scale factors are calculated from the three calibration points closest to it, since the calibration points at the centre of the image are least distorted. It was also assumed that the camera has rectangular pixels. That is why different values were calculated for the scale factors in the u- and v-directions. As will be seen later, the scale factors in both directions are approximately the same for a given camera, which simply means the camera has square pixels.

The projection distortion is caused by a tilt of the camera with respect to the operation field as depicted in *figure 7.1(b)*. As such the points on the operation field have to be transformed into the camera frame before being scaled to pixels in the image. The radial distance from the principal point to some given point in the camera frame may be longer or shorter than its corresponding radial distance in the operation field, depending on the location of this point. This is because a point may be projected closer or further away from the camera as depicted in *figure 7.1(b)*.

Two perpendicular lines in the operation field may not appear perpendicular in the camera frame. This is a draw back because the mappings of the axes of the operation field in the camera frame have to be used to describe the points in the camera frame. This can not be done if the axes of the operation field are no more perpendicular when projected into the camera frame. Therefore, two perpendicular axes have to be chosen which satisfy the condition that they remain perpendicular when projected into the camera frame. The two axes chosen in this project were the line of intersection of the operation field and the plane in the camera frame onto which it is projected and the line perpendicular to it passing through the foot of the perpendicular from the camera to the operation field. Their point of intersection is then the origin. Notice that the origin here corresponds to the mapping of the principal point (x_0,y_0) on the operation field. Before projecting points from the operation field to the camera frame, they have to be translated and rotated to get the points (x_i',y_i') as shown in the figure below;

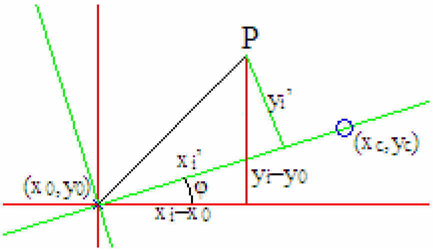


Figure 7.3: Translation and rotation of points in operation field before projection

The red axes are the translated coordinate axes of the operation field with points $(x_i\text{-}x_0,y_i\text{-}y_0)$ and origin at the point (x_0,y_0). The point (x_c,y_c) is the foot of the perpendicular from the camera to the operation field and was measured in the lab. The angle φ is the angle of the line from (x_0,y_0) through (x_c,y_c) measured anti-clockwise from the positive x-axis of the operation field. When the axes of the operation field are projected into the camera frame, they no more appear perpendicular. But the axes described by the green lines do appear perpendicular when projected into the camera frame. As such, an arbitrary point P in the operation field is calculated with reference to the green axes before being projected into the camera frame. This corresponds to translating and rotating the points (x_i,y_i) by an angle φ clockwise about the point (x_0,y_0) to obtain the points (x_i',y_i') as follows;

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} x_i - x_0 \\ y_i - y_0 \end{bmatrix} \qquad (7.2)$$

In *figure 7.1(b)*, the demonstration of the projection distortion is in 2-dimensions, i.e. the projection of the line vertically below the camera passing through the point (x_0,y_0). This figure is not appropriate to use in calculating the projection distortion since a 3-dimensional space has to be taken into consideration. As such, the following figure was used. Here, there is a somewhat better visualisation of the parameters used.

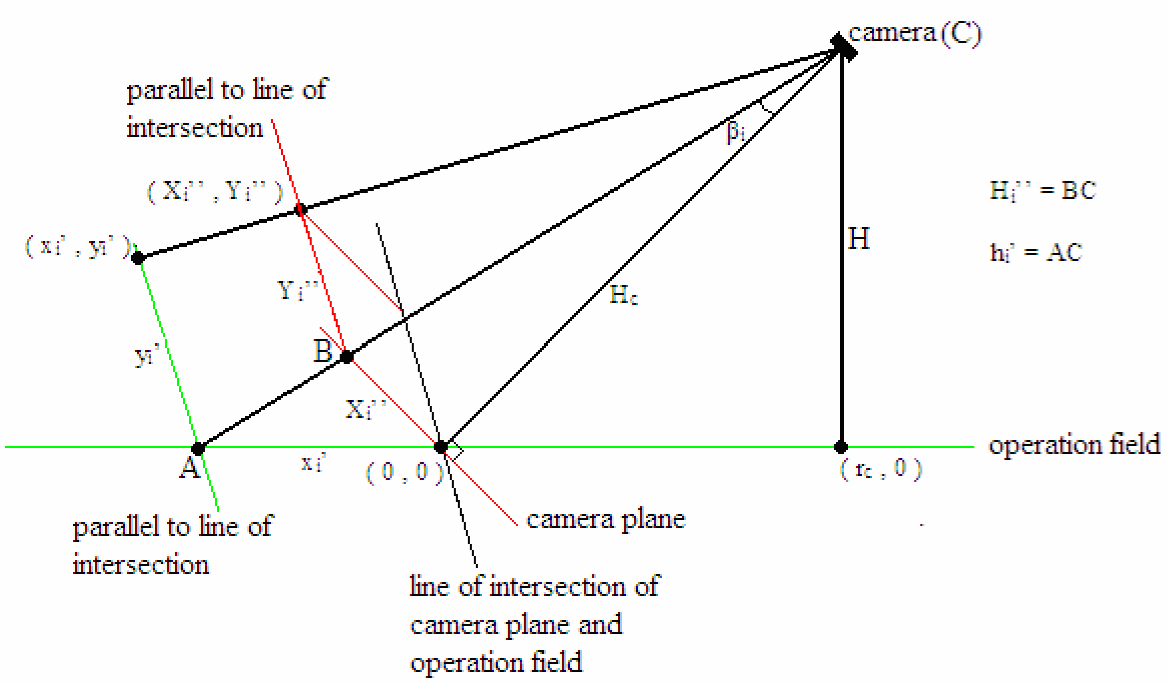


Figure 7.4: 3-dimensional visualisation of projection distortion

In the above figure, the green lines lie in the plane of the operation field and the red ones in the projection plane of the camera frame. The point (0,0) is the mapping of the principal point on the operation field and corresponds to (x_0,y_0) since (x_i',y_i') now assumes negative values. The point $(r_c,0)$ is the foot of the perpendicular projection from the operation field to the camera and corresponds to the point (x_c,y_c). r_c is the radial distance from (x_0,y_0) to (x_c,y_c). H is the perpendicular distance from the operation field to the camera and was measured in the lab. The point (X_i'',Y_i'') is the projection of the point (x_i',y_i') in the camera frame.

In *figure 7.4*, the x-axis of the operation field now lies along the radial distance r_c and the y-axis lies on the line of intersection of the camera plane and the operation field. As can also be noticed, the chosen axes of the operation field remain perpendicular after projection. This was the reason for the initial rotation and this also eases the computation of the projected points. Also notice that lines parallel to the line of intersection of both fields still remain parallel to the line of intersection after projection. From the above observations, one could compute the values of X_i'' as shown in the following equations. The parameters used here are as shown in *figure 7.4*.

$$r_c^2 = (x_c - x_0)^2 + (y_c - y_0)^2$$
$$H_c^2 = H^2 + r_c^2$$
$$h_i'^2 = (r_c - x_i')^2 + H^2$$

$$\beta_i = \arccos\left(\frac{h_i'^2 + H_c^2 - x_i'^2}{2 * h_i * H_c}\right) \qquad (7.3)$$

$$X_i'' = H_c * \tan(\beta_i) * x_i' / |x_i'|$$

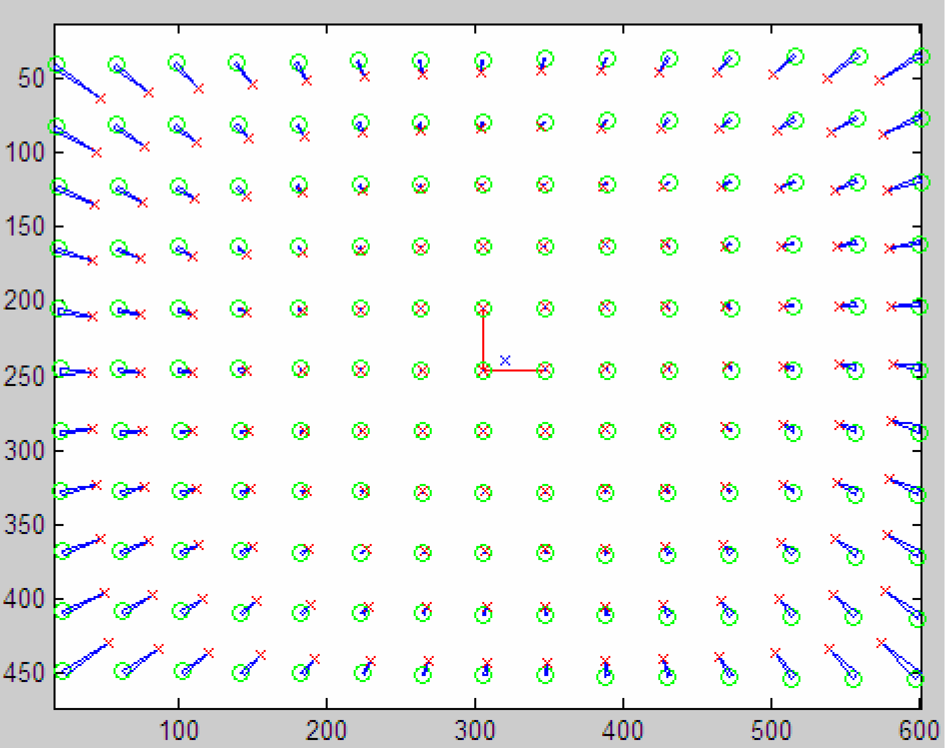
Note from the above computation of X_i'' that if x_i' is negative, then the value obtained above is also negative. Y_i'' could now be computed from X_i''. The triangle described by the camera and the points A and (x_i',y_i') is equivalent to that described by the camera and the points B and (X_i'',Y_i''). From this observation, Y_i'' is computed as follows;

$$H_i''^2 = X_i''^2 + H_c^2$$
$$Y_i'' = y_i' * H_i'' / h_i' \qquad (7.4)$$

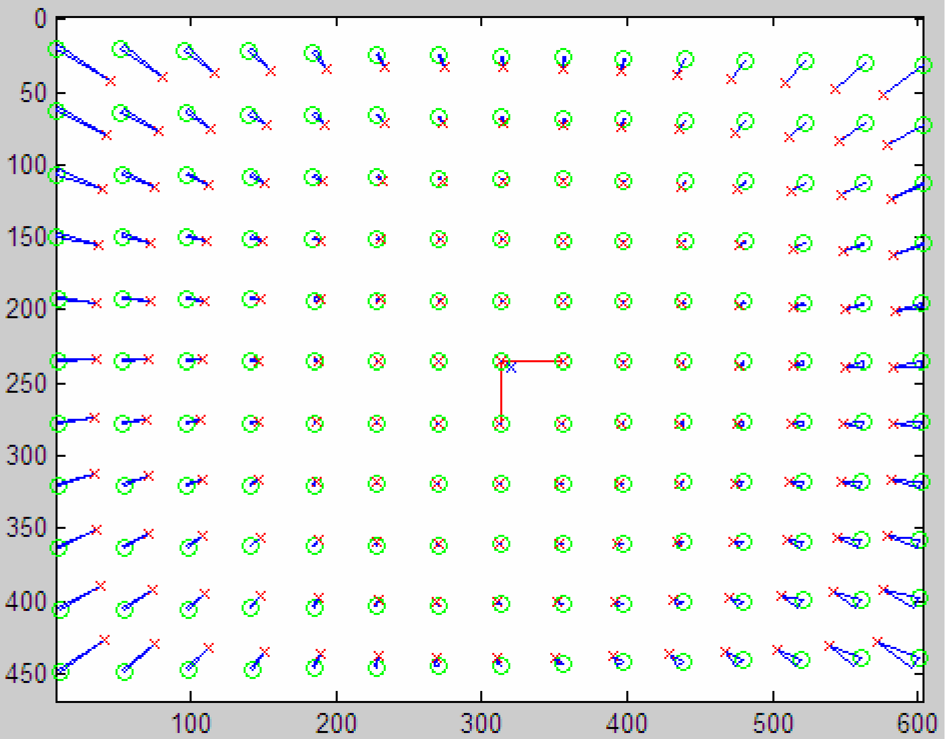
The projected points (X_i'',Y_i'') were then rotated by an angle φ anti-clockwise about their origin and translated to obtain the points (x_i'',y_i'') which were then used to estimate the scale factors as described by *7.1*. The following equation describes this transformation;

$$\begin{bmatrix} x_i'' \\ y_i'' \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} X_i'' \\ Y_i'' \end{bmatrix} \qquad (7.5)$$

The points (x_i'',y_i'') were then scaled to pixels to get (U_i'',V_i''). These scaled points are plotted with the image points (u_i,v_i) as shown below. The scaled points are plotted with green circles and the image points with red crosses. The extra blue cross in each image denotes the principal point of the camera (which is the centre of the image in this calibration). The red lines in the images connect the points closest to the principal points, which were used to estimate the intrinsic parameters. The lines in blue show the radial, tangential and effective distortions.



Vertical axis: v-coordinates in pixels
Horizontal axis: u-coordinates in pixels
(a) Camera 1



Vertical axis: v-coordinates in pixels
Horizontal axis: u-coordinates in pixels
(b) Camera 2

Figure 7.5: Initial distortions

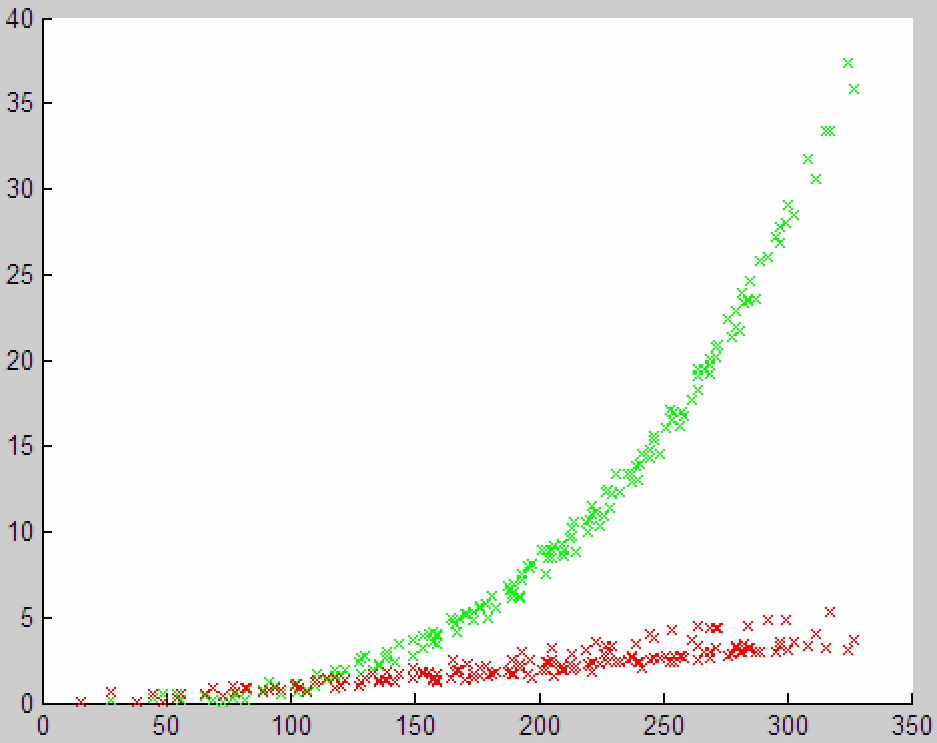
The projected scaled points (U_i'',V_i'') are undistorted and therefore equal to the distorted image points plus their corresponding distortions as shown below.

$$U_i'' = u_i + du_i \qquad \text{and} \qquad V_i'' = v_i + dv_i \qquad (7.6)$$

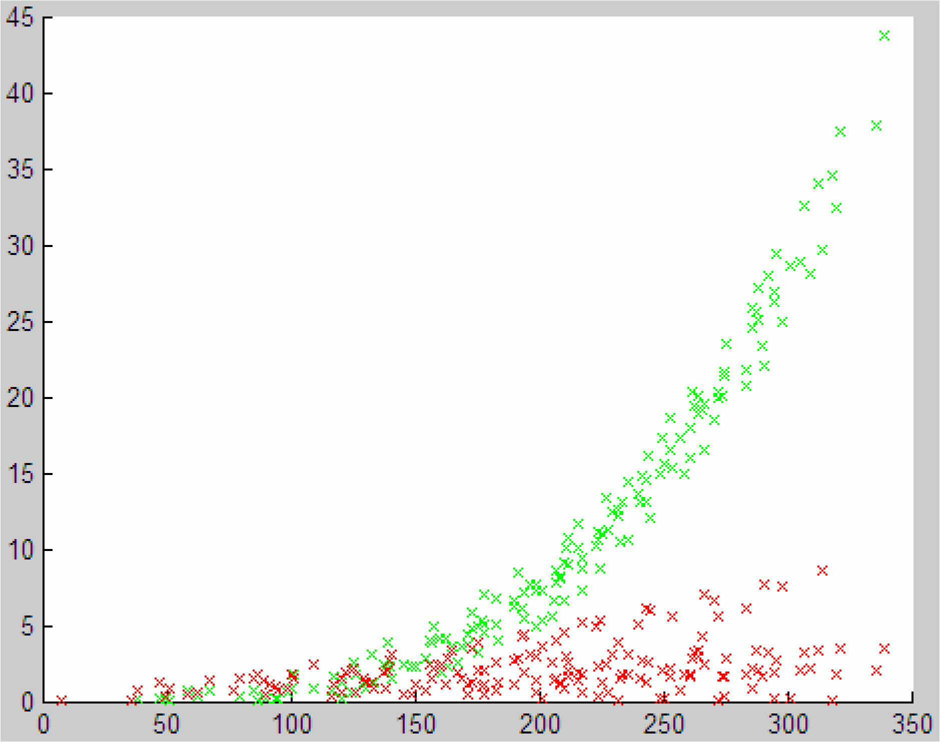
Since du_i and dv_i had not been known at this point, U_i'' and V_i'' were calculated from *7.1* as follows;

$$U_i'' = u_0 + (x_i'' - x_0) / s_u \quad \text{and} \quad V_i'' = v_0 + (y_i'' - y_0) / s_v \qquad (7.7)$$

As expected from the images in *figure 7.3* above, there are little or no distortions at the centre of the images. The distortions increase from the centre of the images towards their borders. The scale factors were calculated from the division of the distance between the mappings of the points connected by the red lines in the camera frame by their image distances. Below is a plot of the initial radial and tangential distortions. They are both plotted against the radial distance from the principal point. The unit of both axes is pixel. The radial distortion is plotted in green and the tangential in red.



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance
(a) Camera 1



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance

(b) Camera 2

Figure 7.6: Initial radial and tangential distortions

The initial value of the objective function for camera 1 was 33440 square pixels and that of camera 2 was 35846 square pixels. The following values were obtained for the extrinsic and intrinsic parameters of camera 1 and 2;

Intrinsic parameters:

Camera 1:

Principal point in pixels (u_0,v_0): (320,240)
Mapping of principal point in mm (x_0,y_0): (3469,971.45)
Scale factor along the x-axis, s_u: 4.8258 mm/pixel
Scale factor along the y-axis, s_v: 4.8146 mm/pixel

Camera 2:

Principal point in pixels (u_0,v_0): (320,240)
Mapping of principal point in mm (x_0,y_0): (1432.2, 1020.6)
Scale factor along the x-axis, s_u: 4.7188 mm/pixel
Scale factor along the y-axis, s_v: 4.7843 mm/pixel

Extrinsic Parameters:

Camera 1:

$\varphi = -17.0941^{o}$
Camera position in mm (x_c,y_c,H): (3555,945,2850)

Camera 2:

$\varphi = -171.131^{o}$
Camera position in mm (x_c,y_c,H): (1300,1000,2840)

So far, just the extrinsic and intrinsic parameters have been estimated but no minimisation has been carried out as yet. However, an initial investigation of the radial and tangential distortions has been done and the initial values of the objective function given. The radial distances are given by;

$$r_i = \sqrt{(u_i - u_0)^2 + (v_i - v_0)^2} \quad , 1 \leq i \leq N \qquad \textit{(7.8)}$$

7.2 Radial Distortions:

Radial distortions are also referred to as lens distortions. They are non-linear and increase radially from the centre of the image to its borders. These distortions were estimated and minimised by using the least squares method to fit a curve into the plot of the radial distortions in *figure 7.6*. Here, a third degree polynomial was used. The radial distortion was estimated as follows;

$$dr_i = k_1 r_i + k_2 r_i^2 + k_3 r_i^3,\ 1 \leq i \leq N \qquad \textit{(7.9)}$$

where k_1, k_2 and k_3 are coefficients of radial distortion and r_i is the radial distance as defined in *7.8*. As no distortions are expected at the principal point of the lens (i.e. the centre of the image in this case), there is no term independent of r_i. The above equation can be expressed in matrix form as follows;

$$A\vec{k} = \vec{b} \qquad \textit{(7.10)}$$

where A is an Nx3-matrix, $\vec{k}$ a 3-dimensional column vector and $\vec{b}$ an N-dimensional column vector, all defined as follows;

$$A = \begin{bmatrix} r_1 & r_1^2 & r_1^3 \\ r_2 & r_2^2 & r_2^3 \\ . & . & . \\ . & . & . \\ . & . & . \\ r_N & r_N^2 & r_N^3 \end{bmatrix}$$

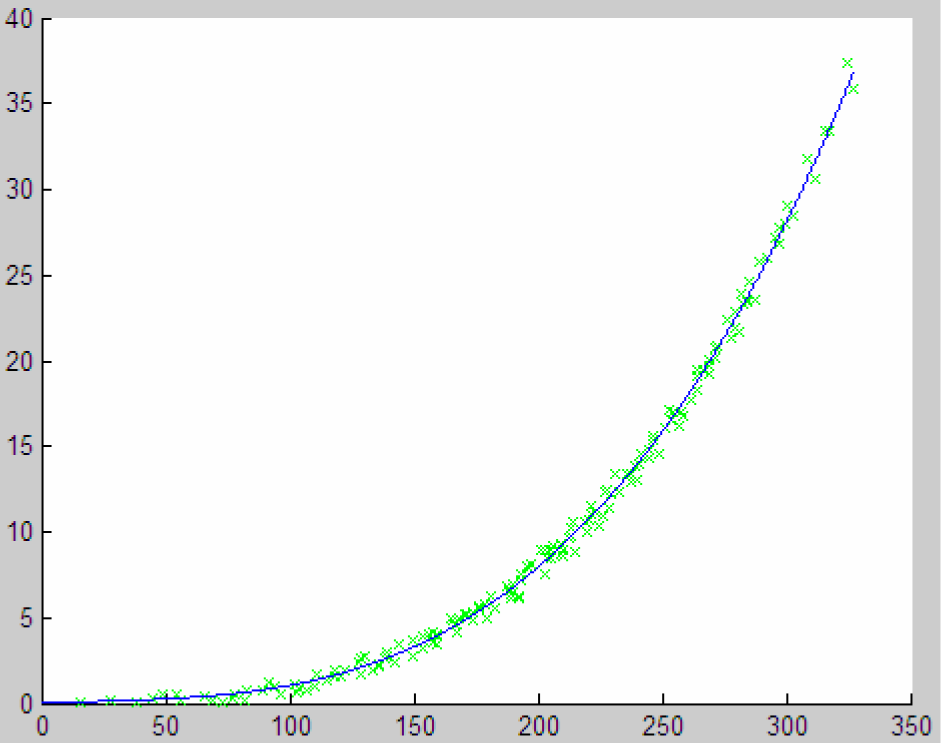
$$\vec{k} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} \qquad \textit{(7.11)}$$

$$\vec{b} = \begin{bmatrix} dr_1 \\ dr_2 \\ . \\ . \\ . \\ dr_N \end{bmatrix}$$

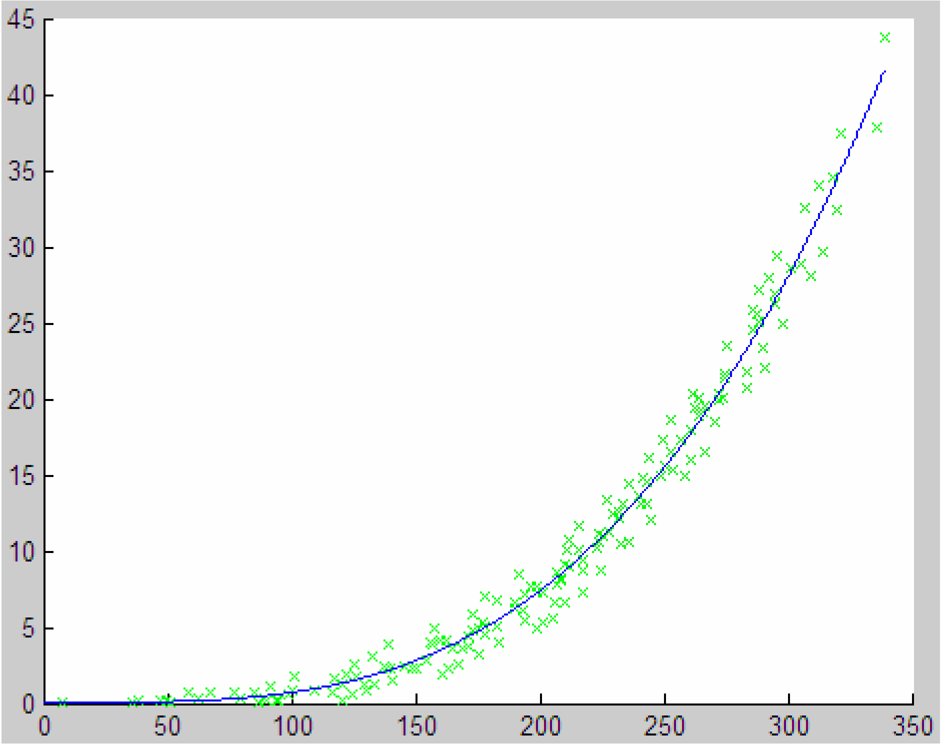
The least squares method can be applied to *7.10* to get the best estimate of the coefficients of radial distortions, i.e. the vector $\vec{k}$. This is done in the following equation;

$$\vec{k} = (A^TA)^{-1}(A^T\vec{b}) \qquad (7.12)$$

The estimation of the radial distortions obtained from the least squares method by a polynomial fit is plotted with blue in the following figure. The actual radial distortions are also plotted with green crosses.



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance
(a) Camera 1



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance

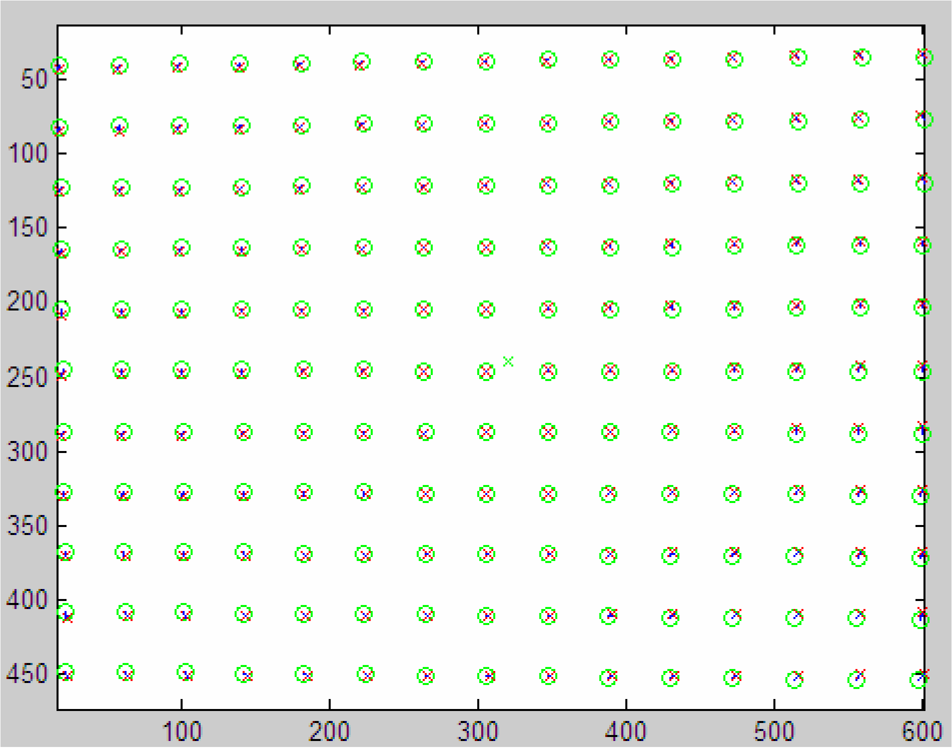
(b) Camera 2

Figure 7.7: Polynomial fit for radial distortions

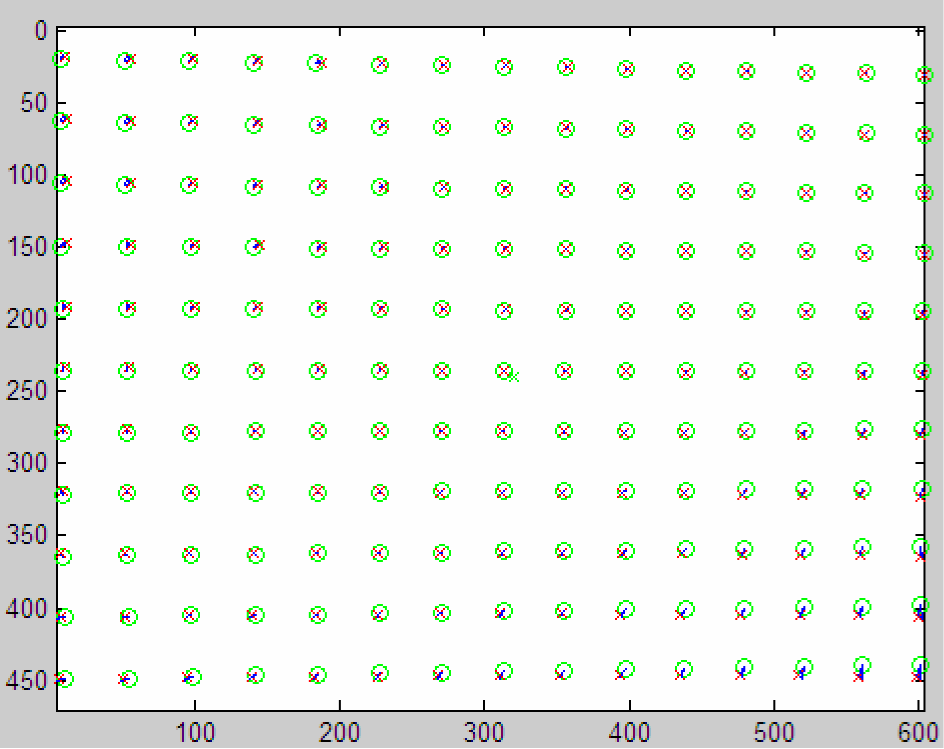
In the estimation of the extrinsic and intrinsic parameters, transformations were carried out only on the actual points of the operation field. These parameters are used to transform points from the operation field into the camera frame and further scale them to get the undistorted image coordinates. Minimisation of the radial distortion is carried out on the image points. Here, the polynomial obtained above from the least squares method is used to estimate the scaled points (U_i'',V_i''). The resulting u- and v- component distortions caused by the radial distortions were calculated according to [14] as follows;

$$du_i = (u_i - u_0) * dr_i / r_i,\ 1 \leq i \leq N$$
$$dv_i = (v_i - u_0) * dr_i / r_i,\ 1 \leq i \leq N \qquad (7.13)$$

The values of dr_i used in the equations above were the estimated values calculated from the polynomial in *7.9* and not the actual values. The values of du_i and dv_i estimated above were used to estimate (U_i'',V_i'') as given by *7.6*. The estimated and actual values of (U_i'',V_i'') are plotted in the following figure with red crosses and green circles respectively;



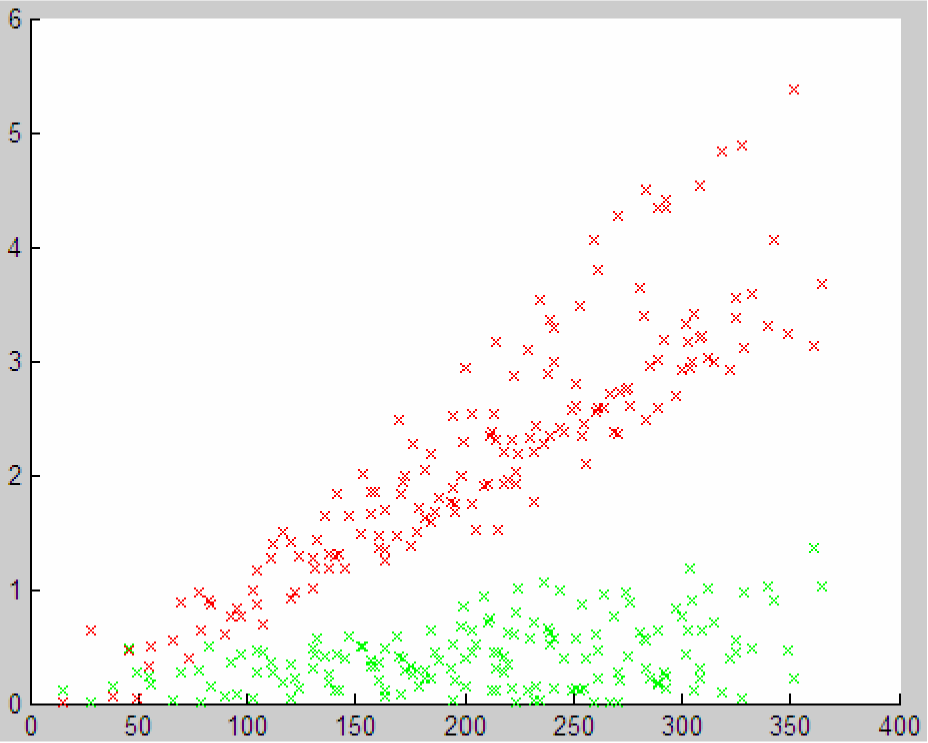
Vertical axis: v-coordinates in pixels
Horizontal axis: u-coordinates in pixels
(a) Camera 1



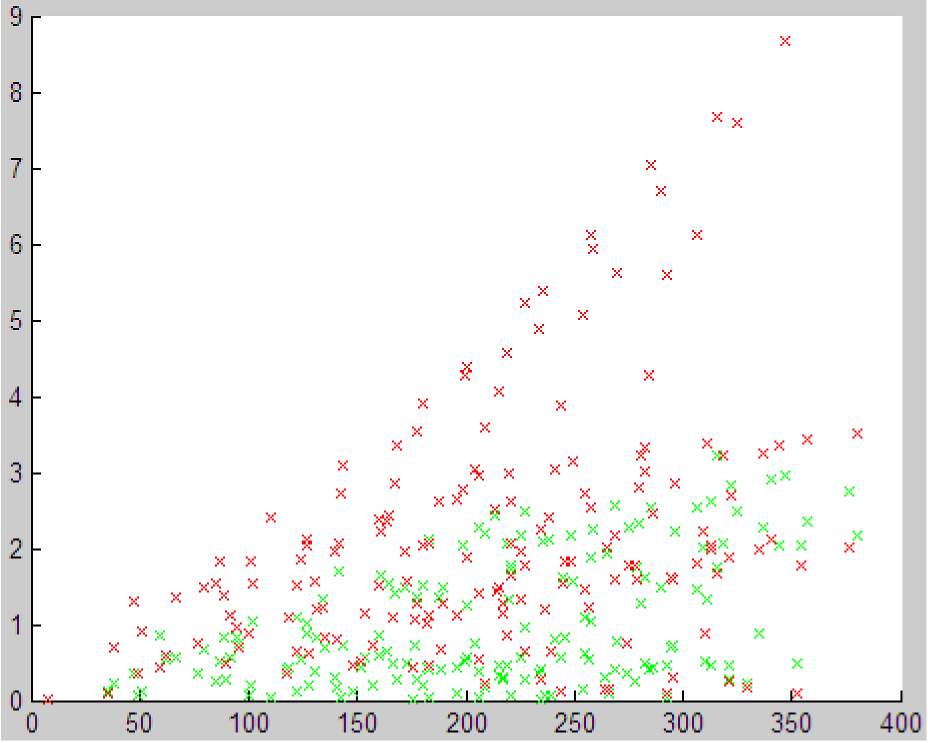
Vertical axis: v-coordinates in pixels
Horizontal axis: u-coordinates in pixels
(b) Camera 2

Figure 7.8: Distortions after minimisation of radial distortions

The resulting radial and tangential distortions are as shown below;



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance
(a) Camera 1



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance
(b) Camera 2

Figure 7.9: Radial and tangential distortions after minimisation of radial distortions

The value of the objective function here was 1075 square pixels for camera 1 which corresponds to an overall reduction of 32365 square pixels from its initial value. That of camera 2 reduced to 1347.6 square pixels, corresponding to an overall decrease of 34498 square pixels from its initial value. Below are the values obtained for the coefficients of radial distortions for camera 1 and 2.

Camera 1:

$k_1 = 0.0070026$

$k_2 = -0.000087083$

$k_3 = 0.0000012575$

Camera 2:

$k_1 = 0.0011844$

$k_2 = -0.000071939$

$k_3 = 0.0000012518$

7.2 Tangential Distortion:

As discussed in the introduction of this chapter, tangential distortions result from the fact that the axes of the operation field may not be aligned with that of the camera. But in the discussions so far, this was assumed. So in order to minimise the tangential distortions, the angle between the axes of the operation field and that on the camera have to be estimated and the coordinates of the operation field rotated appropriately about the mapping of the principal point (x_0,y_0) on the operation field. This is equivalent to rotating the scaled undistorted points (U_i'',V_i'') by an angle α_0 clockwise about the principal point (u_0,v_0) in the image. The estimation was achieved by minimising the objective function F with respect to α_0. Consider three values of the objective function F_0, F_1 and F_2 as plotted in the graph below. They are calculated by rotating (U_i'',V_i'') anti-clockwise about the principal point at angles α_0, α_1 and α_2 respectively where $\alpha_1 = \alpha_0 - \Delta\alpha$ and $\alpha_2 = \alpha_0 + \Delta\alpha$ for some given $\Delta\alpha$.

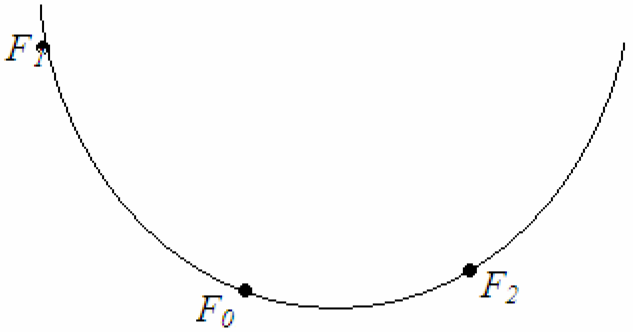
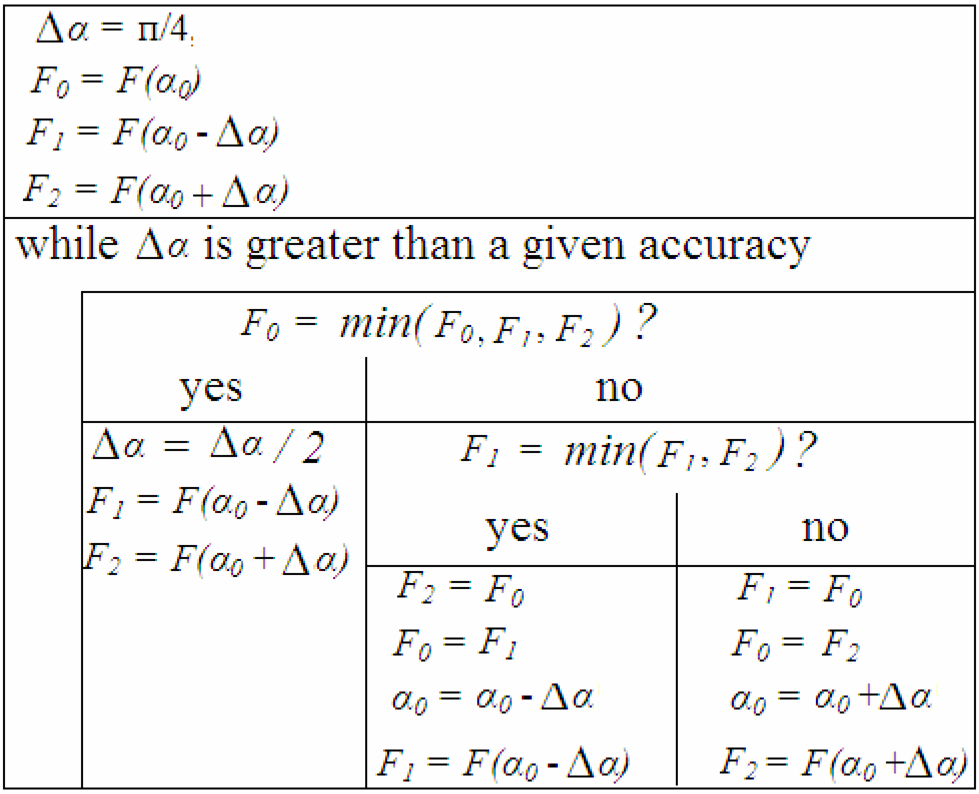


Figure 7.10: Objective function

The following flow chart describes the iterative algorithm used to minimise *F*.



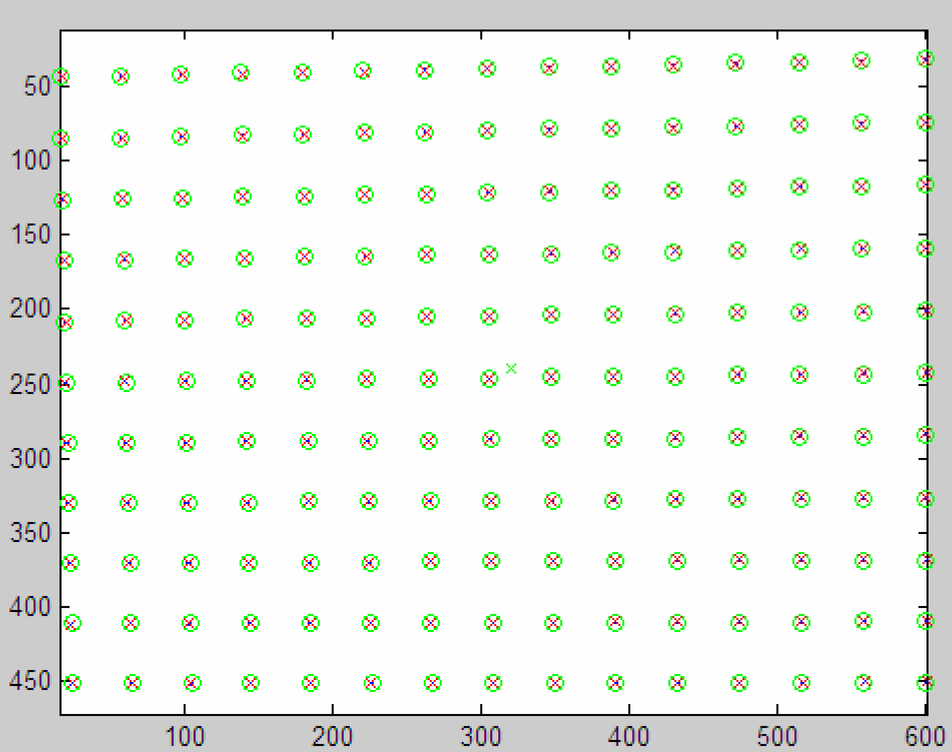
$\Delta\alpha = \pi/4$;
$F_0 = F(\alpha_0)$
$F_1 = F(\alpha_0 - \Delta\alpha)$
$F_2 = F(\alpha_0 + \Delta\alpha)$
while $\Delta\alpha$ is greater than a given accuracy
$F_0 = min(F_0, F_1, F_2)$?
yes
no
$\Delta\alpha = \Delta\alpha / 2$
$F_1 = F(\alpha_0 - \Delta\alpha)$
$F_2 = F(\alpha_0 + \Delta\alpha)$
$F_1 = min(F_1, F_2)$?
yes
no
$F_2 = F_0$
$F_0 = F_1$
$\alpha_0 = \alpha_0 - \Delta\alpha$
$F_1 = F(\alpha_0 - \Delta\alpha)$
$F_1 = F_0$
$F_0 = F_2$
$\alpha_0 = \alpha_0 + \Delta\alpha$
$F_2 = F(\alpha_0 + \Delta\alpha)$

Figure 7.11: Flow chart for minimisation of tangential distortion

When the loop in the algorithm above terminates, then the objective function *F* assumes its minimum at α_0. So the scaled points (U_i'',V_i'') were rotated anti-clockwise about the principal point by the angle α_0 to obtain the points (U_i',V_i') as follows;

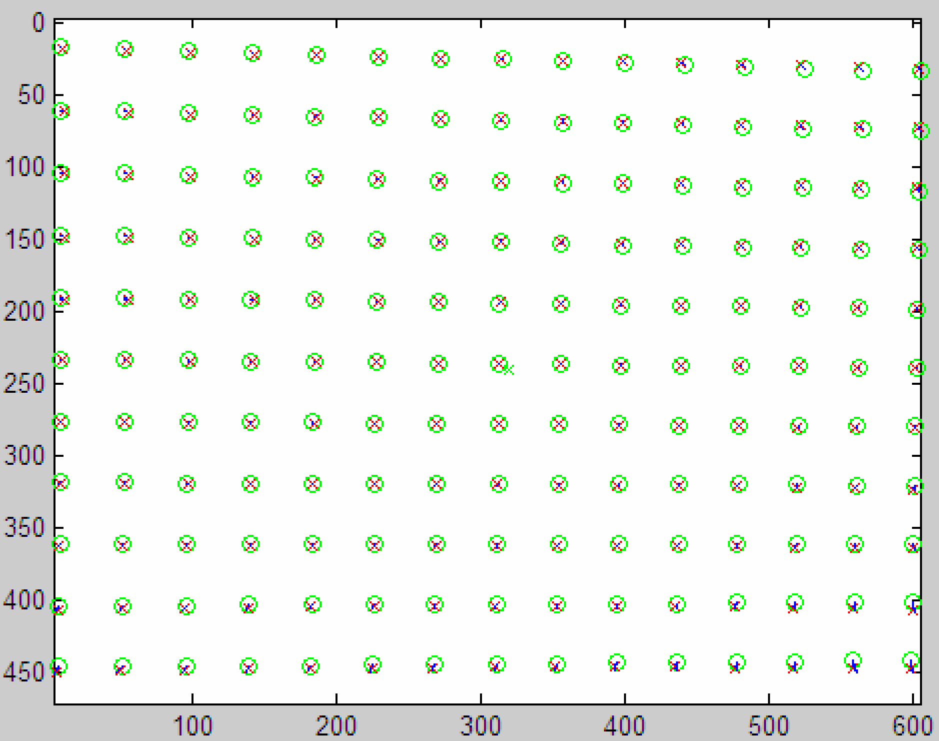
$$U_i' = u_0 + (U_i'' - u_0) * \cos(\alpha_0) – (V_i'' - v_0) * \sin(\alpha_0)$$
$$V_i' = v_0 + (U_i'' - u_0) * \sin(\alpha_0) + (V_i'' - v_0) * \cos(\alpha_0) \qquad (7.14)$$

The angle α_0 was then negated for use in the mapping from the image to the operation field, in which case, (U_i',V_i') will be rotated in the opposite direction to go get (U_i'',V_i''). The newly obtained scaled points (U_i',V_i'), from the above equations, were then plotted with the image points as shown in the following images for camera 1 and 2. Notice the considerable decrease in the tangential distortions in comparison to those in *figure 7.8*.



Vertical axis: v-coordinates in pixels
Horizontal axis: u-coordinates in pixels

(a) Camera 1

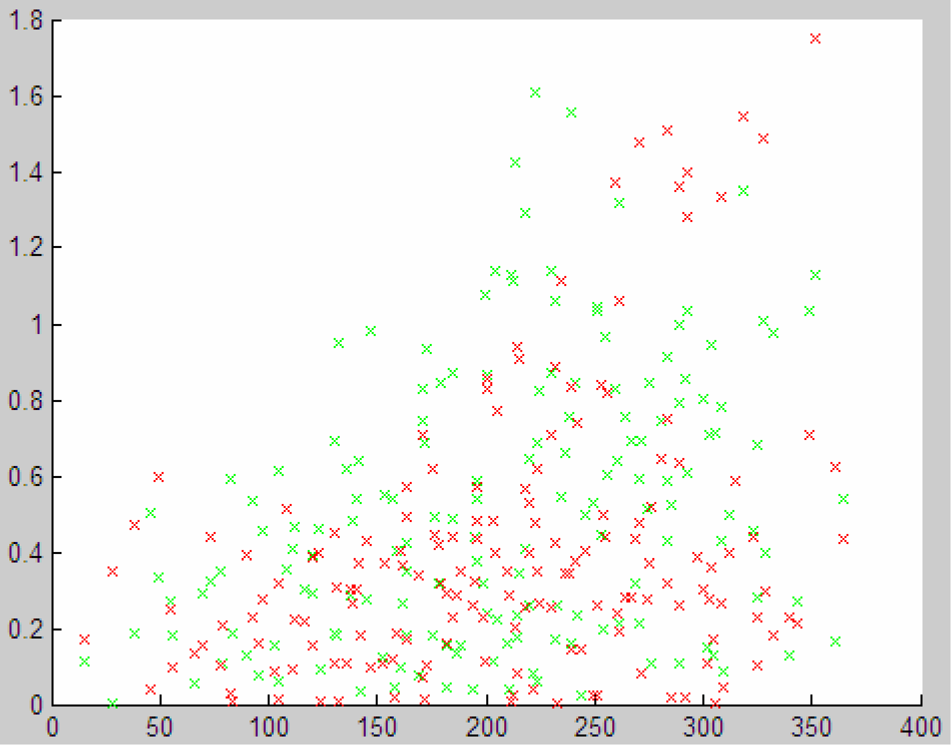


Vertical axis: v-coordinates in pixels
Horizontal axis: u-coordinates in pixels

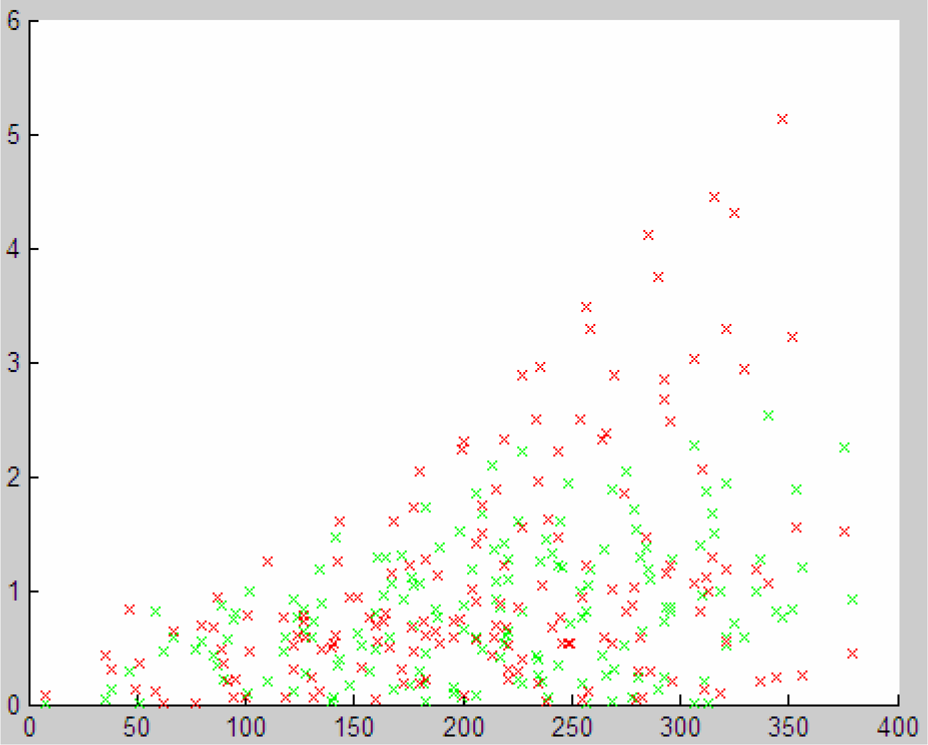
(b) Camera 2

Figure 7.12: Distortions after minimisation of tangential distortions

The residual distortions which were not removed are as shown in the following plots;



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance
(a) Camera 1



Vertical axis: Pixel distortion
Horizontal axis: Radial pixel distance
(b) Camera 2

Figure 7.13: Residual radial and tangential distortions after minimisation of tangential distortions

As can be observed from the figure above, the residual tangential and radial distortions attain a maximum of about 0.6 pixels at the centre of the image and vary up to a maximum of about

1.8 pixels at the borders for camera 1. These correspond to about 2.8mm and 8.6mm respectively in the operation field. For camera 2, the residual tangential error is more than double that for camera 1. The distortions for camera 2 attain a maximum of about 1 pixel at the centre and about 5 pixels at the borders. These correspond to about 4.7mm and 23.5mm respectively in the operation field. As can be observed from the above figure, the tangential distortions were considerably reduced when compared to their values in *figure 7.9*.

The new value of the objective function for camera 1 was reduced to 111.17 square pixels. This corresponds to a reduction of 33329 square pixels from its initial value. That of camera 2 on the other hand was reduced to 515.86 square pixels, which corresponds to a reduction of 35330 square pixels from its initial value. The following angles were computed for camera 1 and 2;

Camera 1: α_0 = 0.010879 rad
Camera 2: α_0 = -0.0099246 rad

The function *calibrate()* (see appendix) implements this calibration procedure. Below is a summary of the three main steps used in this calibration procedure;

1. In the first step, the extrinsic and intrinsic camera parameters were estimated. This involved the minimisation of the projection distortion. The calibration points (x_i,y_i) on the operation field were translated and rotated by the angle φ clockwise about the mapping of the principal point (x_0,y_0) on the operation field to obtain the points (x_i',y_i') as given in *7.2*. The points (x_i',y_i') were then projected into the camera frame to obtain the points (X_i'',Y_i'') which were further rotated by the angle φ anti-clockwise about (x_0,y_0) and translated to get the points (x_i'',y_i''). These points were then used to estimate the scale factors s_u and s_v.
2. In the second step, the radial distortion was estimated with a polynomial of third degree as shown in *7.9*. Here the least squares method was used to estimate the coefficients of this polynomial.
3. In the third and final step, the tangential error was minimised by rotating the image points (u_i,v_i) by an angle α_0 clockwise about the principal point (u_0,v_0).

7.4 Mapping from Image to Operation Field:

So far the transformation from the operation field into the camera frame has been discussed. However, in this project one is interested in the mapping from the image to the operation field. This involves a transformation from the camera frame to the operation field. So the procedure used in the transformation from the operation field to the camera frame will simply be walked back.

Given the image coordinates (u,v), it is required to compute its mapping (x,y) in the operation field. This is done in the following steps;

1. Estimate the radial distortion dr with the help of the coefficients of radial distortion as given in *7.9*.
2. Calculate the u- and v-component distortions, du and dv respectively, from the radial distortion estimated in the first step using *7.13*.
3. Scale u + du and v + dv from pixels to mm to obtain their mappings (x'',y'') in the camera frame as given by *7.1*.

4. Rotate (x'',y'') about (x_0,y_0) by an angle $\varphi - \alpha_0$ clockwise to get the point (X'',Y''). Remember α_0 was used to minimise the tangential error.
5. Project the points (X'',Y'') onto the operation field to obtain the points (x',y').
6. Finally, rotate the points (x',y') by an angle of φ anti-clockwise about the point (x_0,y_0) to obtain (x,y).

All the formulas used in the steps above have been given except for step 5, the projection transformation from the camera frame to the field. Consider the following figure which demonstrates a projection from the camera frame to the operation field. It is the same with that in *figure 7.4*. Only some additional parameters are indicated to describe this projection.

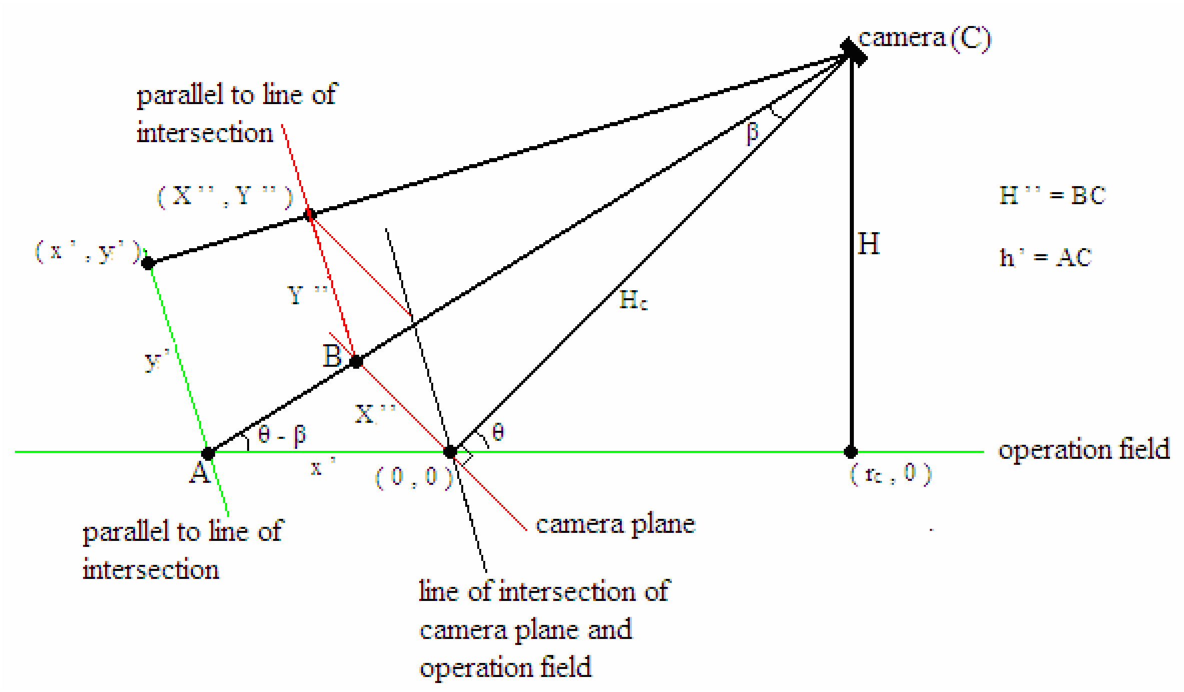

camera (C)
parallel to line of intersection
(X'', Y'')
(x', y')
β
H'' = BC
h' = AC
H
H_c
Y''
y'
B
X''
θ - β
θ
A
x'
(0, 0)
(r_c, 0)
operation field
parallel to line of intersection
camera plane
line of intersection of camera plane and operation field

Figure 7.14: 3-dimensional visualisation of projection transformation

In this case, (X'',Y'') is known and it is required to calculate (x',y'). H_c and r_c have already been calculated in *7.3*. The procedure used in the projection from the operation field to the camera frame is similar to that here. The coodinate x' will first be calculated and then used to calculate y'. The observations made in section 7.1 still hold here of course, and will not be repeated. The calculation of (x',y') from (X'',Y'') is as follows;

$$\begin{aligned}
&\beta = \tan(X'' / H_c) \\
&x' = H_c * \sin(\beta) / \sin(\theta+\beta) \\
&H_i''^2 = X_i''^2 + H_c^2 \qquad\qquad (7.15) \\
&h_i'^2 = (r_c - x_i')^2 + H^2 \\
&y_i' = Y_i'' * h_i' / H_i''
\end{aligned}$$

Remember that the calculations above are with the negative x-axis of the camera frame. This would give β a negative value from the first equation above. This is why the angle used to calculate x' in the second equation above is θ+β and not θ-β as depicted in *figure 7.14*.

The point (x,y) calculated from step 6 above is the point on the operation field. Remember the robot has a certain height from the operation field which gives rise to a projection error as described in section 7.4. So in addition to the mapping described above, a projection correction has to be done in order to properly track the robot. The complete algorithm used in tracking the robot is implemented in the function *track_robot()* (see appendix). The steps used in this algorithm are as follows;

1. Configure and acquire the image from both cameras, one after the other as described in chapter 3. Remember only one video object can exist at a time.
2. Extract the reflex marks from the images by using the thresholding and boundary tracing algorithms described in chapter 4.
3. Calculate the centre and radius of each reflex mark using the circle fit method described in chapter 5.
4. Map the centre of the reflex mark to its corresponding point in the operation field with the help of the calibration parameters.
5. Correct the projection error resulting from the robots height.
6. Group the reflex marks into pairs. Each pair should have the minimum distance between them.
7. Determine which mark in the pair is the larger one by comparing their radii. The centre of the larger mark gives the robot position while the smaller one gives the orientation of the robot as explained in section 1.1.
8. Reflex marks without any partner are ignored.

Chapter 8: Conclusion

This work was all about calibrating cameras to track a mobile robot with two overhead video cameras. The robots were equipped with reflex marks which are highly reflexive to light. These reflex marks were placed at the top of the robot and used to deduce its position and orientation. The cameras were manually set to take only grey images and the exposure time of the shutter was further controlled to take images in which only the reflex marks can be seen. This greatly helped reduce the effort invested in image processing to extract the reflex marks from the images.

The cameras were positioned in such a way that their views on the operation field overlapped. This made it possible for the robots to always be in view as long as they were in the operation field and the robots could move from the view of one camera to the other without lose of track. This project did not involve robot control anyway.

The main theme of this project was camera calibration. Some image and data processing techniques were also presented. A step-by-step camera calibration procedure, which involved a stepwise elimination of the distortions taken into consideration, was also presented. The main difficulty encountered was the description of the projection transformation to and from the camera frame and the operation field.

In previous works on robot control in the lab, the camera calibration was done by interpolation. The disadvantage here was the large parameter set which had to be stored. All the 165 points used for calibrating each camera and their image mappings had to be stored, making a total of 1320 floating point values. The time used in computing the mapping of an image point on the operation field was also location dependent as the algorithm had to search for the three closest calibration points. As compared to the camera calibration procedure presented here, there are only 14 calibration parameters stored for each camera and the algorithm is location independent. It has a fixed time to compute the mapping for any point on the operation field.

One thing which will be of great interest is a thorough investigation of the accuracy of the camera calibration by interpolation and that presented here. One could definitely say for the interpolation method that there is no error at the calibration points since all their mappings were stored. But the errors in between the calibration points will be of interest. A further investigation of the possible sources of distortions and their minimisations in order to reduce the residual error in this calibration procedure could also be done. These were not done in this project due to time constraints.

References

[1] Heikkilä, J. & Silvén, O. (1997) A four-step camera calibration procedure with implicit image correction. Infotech Oulu and Department of Electrical Engineering, University of Oulu, FIN-90570 Oulu, Finland

[2] Abdel-Aziz, Y. I. & Karara, H. M. (1997) Direct linear transformation into object space coordinates in close-range photogrammetry. Proc. Symposium on Close-Range Photogrammetry, Urbana, Illinois, p. 1-18

[3] Faugeras, O. D. & Toscani, G. (1987) Camera calibration for 3-D computer vision. Proc. International Workshop on Industrial Applications of Machine Vision and Machine Intelligence, Silken, Japan, p. 240-247

[4] Sam Roweis, Levenberg-Marquardt Optimization
http://www.cs.toronto.edu/~roweis/notes/lm.pdf

[5] Klaus Strobl, Wolfgang Sepp, Stefan Fuchs, Christian Paredes and Klaus Arbter (2007) Camera Calibration Toolbox for Matlab
http://www.vision.caltech.edu/bouguetj/calib_doc/

[6] Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins (2004) Digital Image Processing using MATLAB

[7] Brown D. C. (1971) Close Range Camera Calibration. DBA Systems, Inc. Melbourne, Fla. 32901. http://www.vision.caltech.edu/bouguetj/calib_doc/papers/Brown71.pdf

[8] Wang Q. & Ye G. (2005) Observation Camera Based Navigation System for Mobile Robots. Master thesis at Hamburg University of Applied Sciences.

[9] Chung J. (2006) Formation Control for Multiple Robots. Master thesis at Hamburg University of Applied Sciences.

[10] AmigoBot website: http://www.activrobots.com/ROBOTS/amigobot.html

[11] 3M Diamond Grade TM VIP Reflexive Sheeting (2005) www.3M.com

[12] MATLAB User's Guide (2003) The MathWorks, Inc.

[13] Ostu N. (1979) A Threshold Selection Method from Gray-Level Histograms. IEEE Trans. System, Man and Cybernetics, vol. 79, no. 6, pp. 745-754

[14] Rojas Raul (2004) Calibrating an Overhead Video Camera. Freie Universität Berlin.
http://www.fu-fighters.de

[15] (Aug. 2005) Image Analysis and Understanding. Lecture notes, College of CSS Engineering, University of Iowa
http://www.icaen.uiowa.edu/~dip/LECTURE/Segmentation2.html#tracing

[16] Nettle P. (1997) WWH1: Sup-Pixel Accuracy.
http://www.gamedev.net/reference/articles/article661.asp

APPENDIX A: Alphabetic Function List

calcF	
Function	F = calcF(U, V, u, v, u0, v0, alpha)
File	calcF.m
Description	Calculates the value of the objective function obtained by rotating (U,V) by an angle alpha anti-clockwise about the point (u0,v0). U, V, u and v must be vectors of same length.
Usage	Estimation of the tangential distortion
Input	U: x-coordinates of undistorted image points V: y-coordinates of undistorted image points u: x-coordinates of distorted image points v: y-coordinates of distorted image points u0: x-coordinate of principal point v0: y-coordinate of principat point alpha: angle of rotation in radians
Output	F: value of objective function

calibrate	
Function	calibparam = calibrate(cpos, fpoints, impoints)
File	calibrate.m
Description	Specifically calibrates the cameras used in this project.
Usage	Camera calibration (see chapter 7)
Input	cpos: 3D coordinate of the camera position in the operation field fpoints: 2x165 matrix of the calibration points on the operation field in mm. The 1st and 2nd rows are the x- and y-coodinates respectively. The coordinates must form a grid of dimensions 15 by 11 points spaced at 200mm from each other and arranged columnwise. impoints: Corresponding image points of the calibration points.
Output	calibparam: Structure containing all the calibration parameters. calibparam.pos: camera position given by cpos calibparam.pp: principal point and its mapping in operation field calibparam.angles: transformation and tagential distortion angles calibparam.s: scale factors calibparam.k: coefficients of radial distortion
Function calls	project, calcF

circle_fit	
Function	[circle, z] = circle_fit (x, y)
File	circle_fit.m
Description	Computes the best circle fitting using the least square's method. x and y must be column vectors of same dimension greater than 2.
Usage	Calculation of the centre and radius of a reflex mark (see section 5.2.1)
Input	(x,y): approximate coordinates of circle
Output	circle: Structure containing centre and radius of a circle circle.pos: centre of circle circle.r: radius of circle z: coefficients of circle defined by x^2 + y^2 + z(1)x + z(2)y + z(3) = 0

ellipse_fit	
Function	[ellipse, z] = ellipse_fit(x, y)
File	ellipse_fit.m
Description	Computes the best ellipse fitting using the least square's method. x and y must be column vectors of same dimension greater than 4.
Usage	Calculation of the centre of a reflex mark (see section 5.2.2)
Input	(x,y): approximate coordinates of ellipse
Output	ellipse: Structure defining an ellipse ellipse.pos: centre of ellipse ellipse.a: length of x-axis of ellipse ellipse.b: length of y-axis of ellipse ellipse.angle: angle of rotation of ellipse measured anti-clockwise from positive x-axis z: coefficients of ellipse defined by z(1)x^2 + z(2)y^2 + z(3)xy + z(4)x + z(5)y + 1 = 0.

get_image	
Function	img = get_image(c_config)
File	get_image.m
Description	Resets, configures and aquires image from camera.
Usage	Camera configuration and image acquisition (see chapter 3)
Input	c_config: Structure containing camera configuration parameters. c_config.num: Camera number. Must be 1 or 2. c_config.exposure: exposure time of camera shutter
Output	img: gray image obtained from camera

get_pos_mm	
Function	pos_mm = get_pos_mm(pos_px, calibparam)
File	get_pos_mm.m
Description	Calculates the mapping of an image point on the operation field in mm
Usage	Caculation of robot position
Input	pos_px: image coordinates calibparam: Structure containing all the calibration parameters. calibparam.pos: camera position given by cpos calibparam.pp: principal point and its mapping in operation field calibparam.angles: transformation and tagential distortion angles calibparam.s: scale factors calibparam.k: coefficients of radial distortion
Output	pos_mm: mapping of pos_px in mm on operation field
Function call	reproject

get_reflex_marks	
Function	circles = get_reflex_marks(BW)
File	get_reflex_marks.m
Description	Computes the centre and radius of each reflex mark in an image.
Usage	Boundary tracing and circle fit (see chapter 5)
Input	BW: binary image
Output	circles: Array of structures describing a circle. circles{i}.pos: centre of circle circles{i}.r: radius of circle
Function call	circle_fit

imthresh	
Function	BW = imthresh(Img)
File	imthresh.m
Description	Segments a gray image to a binary image by local thresholding.
Usage	Thresholding (see section 4.3.4)
Input	Img: gray image
Output	BW: binary image

plot_circle	
Function	plot_circle(circle, str)
File	plot_circle.m
Description	Plots a circle with its horizontal and vertical diameter.
Usage	Visual verification
Input	circle: Structure containing centre and radius of a circle circle.pos: centre of circle circle.r: radius of circle str: string used to define mode of plot

plot_ellipse	
Function	plot_ellipse(ellipse, z, str)
File	plot_ellipse.m
Description	Plots an ellipse with its axes.
Usage	Visual verification
Input	ellipse: Structure defining an ellipse ellipse.pos: centre of ellipse ellipse.a: length of x-axis of ellipse ellipse.b: length of y-axis of ellipse ellipse.angle: angle of rotation of ellipse measured anti-clockwise from positive x-axis z: coefficients of ellipse defined by z(1)x^2 + z(2)y^2 + z(3)xy + z(4)x + z(5)y + 1 = 0. str: string used to define mode of plot

project	
Function	W = project(w, fc, cpos)
File	project.m
Description	Projects points from operation field into camera frame.
Usage	Projection transformation
Input	w: points on operation field fc: mapping of the principal point in the operation field cpos: camera position in the operation field
Output	W: projected points in camera frame

reorder	
Function	w = reorder(w)
File	reorder.m
Description	Reorders the image points to correspond to that of the operation field.
Usage	Camera calibration.
Input	w: unordered image points. Must be a 2x165 matrix with x- and y-coordintes in the 1st and 2nd rows respectively.
Output	w: columnwise ordered image points

reproject	
Function	W = reproject(w, fc, cpos)
File	reproject.m
Description	Reprojects points from camera frame onto operation field
Usage	Projection transformation
Input	w: points in camera frame fc: mapping of the principal point in the operation field cpos: camera position in the operation field
Output	W: reprojected points on operation field

set_Tmin	
Function	set_Tmin()
File	set_Tmin.m
Description	Computes the maximum gray levels of the background images of both cameras and uses them to initialize the minimum threshold levels for both cameras.
Usage	Thresholding (see section 4.3.4)

track_robots	
Function	robots = track_robots()
File	track_robots.m
Description	Gets images from both cameras and tracks all robots in view
Usage	Robot tracking (see section 7.4)
Output	robots: Array of structures containing robot position and orientation robots{i}.pos: position of robot robots{i}.angle: angle of orientation of robot measured anti-clockwise from the positive x-axis
Function calls	get_image, imthresh, get_reflex_marks, get_pos_mm

Appendix B: Source Code

The following MATLAB m-file is used to declare and initialize all global variables needed to track the robot. This file must be executed before the function *track_robot()* (see Appendix A) is used. At the end of this file, the user is prompted to set the minimum threshold levels. This is done automatically by the function *set_Tmin()* (see Appendix A). It is advised to always accept this option since the minimum threshold may vary with luminance.

```
% File: init.m
%
% Description:
% Defines and initializes all global parameters.


% minimum threshold levels
global Tmin;
Tmin = 47;    % default value
global Tmin1; % minimum threshold for camera 1
Tmin1 = 46;   % default value
global Tmin2; % minimum threshold for camera 1
Tmin2 = 47;   % default value

% calibration parameters
global c1_calibparam;
global c2_calibparam;
load calibparam; % load calibration parameters

% camera configuration parameters
global c1_config;
c1_config.num = 1;
c1_config.exposure = -10;
global c2_config;
c2_config.num = 2;
c2_config.exposure = -8;

% compute minimum threshold levels
if 'y' == input('Do you want to compute the minimum threshold
levels?\nEnter y for yes: ', 's'),
    set_Tmin;
end
```

On the following page is the main calibration function used in this project. Its input variables are stored in the MATLAB data-file *calibdata.mat*. These variables are prefixed with c1 and c2 for camera 1 and camera 2 respectively. So the mat-file must be loaded into the workspace and the function called appropriately to calibrate camera 1 or 2. Note that the version of this function on the next page is summarized. The code used to automatically generate the images as shown in chapter 7 and to compute the objective function after each minimisation has been deleted. This code is of course not deleted in the CD delivered with this project.

```
function calibparam = calibrate(cpos, fpoints, impoints)
% File: calibrate.m
%
% Description:
% Specifically calibrates the cameras used in this project.
%
% Inputs:
%   cpos: 3D coordinate of the camera position in the operation field
%   fpoints: 2x165 matrix of the calibration points on the operation field
%       given in mm. The 1st and 2nd rows are the x- and y-coodinates
%       respectively. The coordinates must form a grid of dimensions 15 by
%       11 points spaced at 200mm from each other and arranged columnwise.
%   impoints: Corresponding image points of the calibration points.
% Output:
%   calibparam: Structure containing all the calibration parameters.
%       calibparam.pos: camera position given by cpos
%       calibparam.pp: principal point and its mapping in operation field
%       calibparam.angles: transformation and tagential distortion angles
%       calibparam.s: scale factors
%       calibparam.k: coefficients of radial distortion



%============= estimation of extrinsic & extrinsic parameters =============

u = impoints(1,:);
v = impoints(2,:);
x = fpoints(1,:);
y = fpoints(2,:);
t = cpos;
L = length(u);
u0 = 320;
v0 = 240;
Dx = 200;
Dy = 200;

r = ((u - u0).^2 + (v - v0).^2).^0.5;
[rmin, pos] = min(r); % calculate position of point closest to (u0,v0)

% calculate x0 and y0 from u0 and v0
if u(pos) < u0,
    x0 = x(pos) + Dx * (u0 - u(pos)) / (eps + u(pos+11) - u(pos));
    upos = pos + 11;
else
    x0 = x(pos-11) + Dx * (u0 - u(pos-11)) / (eps + u(pos) - u(pos-11));
    upos = pos - 11;
end
if v(pos) < v0,
    y0 = y(pos) + Dy * (v0 - v(pos)) / (eps + v(pos+1) - v(pos));
    vpos = pos + 1;
else
    y0 = y(pos-1) + Dy * (v0 - v(pos-1)) / (eps + v(pos) - v(pos-1));
    vpos = pos - 1;
end

% calculate transformation angle phi from camera position
phi = angle(j * (t(2) - y0) + (t(1) - x0));

% rotate (x,y) clockwise about (x0,y0) by phi
XY = [x0 * ones(1,L);y0 * ones(1,L)] + [cos(phi) sin(phi); -sin(phi)
cos(phi)] * [x - x0; y - y0];
```

```
% project (X,Y) into camera frame
XY = project(XY, [x0;y0], t);
X = XY(1,:);
Y = XY(2,:);

% rotate (X,Y) anti-clockwise about (x0,y0) by phi
XY = [x0 * ones(1,L);y0 * ones(1,L)] + [cos(phi) -sin(phi); sin(phi)
cos(phi)] * [X - x0; Y - y0];
X = XY(1,:);
Y = XY(2,:);

% calculate scale factors
if u(pos) < u0,
    Su = ((X(pos+11) - X(pos))^2 + (Y(pos+11) - Y(pos))^2)^0.5 / (eps +
((u(pos+11) - u(pos))^2 + (v(pos+11) - v(pos))^2)^0.5);
else
    Su = ((X(pos) - X(pos-11))^2 + (Y(pos) - Y(pos-11))^2)^0.5 / (eps +
((u(pos) - u(pos-11))^2 + (v(pos) - v(pos-11))^2)^0.5);
end
if v(pos) < v0,
    Sv = ((X(pos+1) - X(pos))^2 + (Y(pos+1) - Y(pos))^2)^0.5 / (eps +
((u(pos+1) - u(pos))^2 + (v(pos+1) - v(pos))^2)^0.5);
else
    Sv = ((X(pos) - X(pos-1))^2 + (Y(pos) - Y(pos-1))^2)^0.5 / (eps +
((u(pos) - u(pos-1))^2 + (v(pos) - v(pos-1))^2)^0.5);
end

%================== estimation of radial distortion ===================

% scale (X,Y) to pixels
U = u0 + (X - x0) / (eps + Su);
V = v0 + (Y - y0) / (eps + Sv);

dU = U - u;
dV = V - v;
R = ((U - u0).^2 + (V - v0).^2).^0.5;
alpha = abs(atan((v-v0) ./ (u-u0+eps)) - atan((V-v0) ./ (U-u0+eps)));
d = (dU.^2 + dV.^2).^0.5;
dT = R .* sin(alpha);
dr = (d.^2 - dT.^2).^0.5;
r = r.';
k = [r r.^2 r.^3] \ dr.'; % estimate coefficients of radial distortion
r = r.';
dr = k(1) * r + k(2) * r.^2 + k(3) * r.^3; % estimate radial distortion
u = u0 + (u - u0) .* (1 + dr ./ (eps + r));
v = v0 + (v - v0) .* (1 + dr ./ (eps + r));

%============= estimation of tangential distortion ===================

alpha0 = 0;
step = pi/4;
F0 = calcF(U, V, u, v, u0, v0, 0);
F1 = calcF(U, V, u, v, u0, v0, alpha0 - step);
F2 = calcF(U, V, u, v, u0, v0, alpha0 + step);
while(step ~= 0)
    [F,p] = min([F0 F1 F2]);
    if p == 1
        step = step / 2;
        F1 = calcF(U, V, u, v, u0, v0, alpha0 - step);
        F2 = calcF(U, V, u, v, u0, v0, alpha0 + step);
    elseif p == 2
        F2 = F0;
        F0 = F1;
        alpha0 = alpha0 - step;
        F1 = calcF(U, V, u, v, u0, v0, alpha0 - step);
    else
        F1 = F0;
        F0 = F2;
        alpha0 = alpha0 + step;
        F2 = calcF(U, V, u, v, u0, v0, alpha0 + step);
    end
end
alpha0 = -alpha0;

% create calibparam structure
calibparam.pos = cpos;
calibparam.pp = [u0 v0; x0 y0];
calibparam.angles = [phi alpha0];
calibparam.s = [Su Sv];
calibparam.k = k;
```

Declaration

I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Bachelor Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

____

City, Date and Signature