3D Measurement Using Camera

Masters Thesis

Gunnar Hardarson
s991208

October 2005
IMM DTU

Abstract

This thesis is about the reconstruction of the depth in a scene from 2D images. A camera is moved around an object, creating a sequence of 2D images. Features are tracked through the images, where the movement is used for 3D reconstruction. Position, rotation and translation of the camera is known.

Using feature detection, no laser or structured light has to be used. Three methods of feature detection are proposed and compared, where one is chosen and used. These are corner detection, edge detection and optical flow. Tests are carried out, where edge detection is chosen to be the method used.

An edge tracking algorithm is implemented. This tracking algorithm tracks edges through numerous images, which is essential for a good 3D reconstruction.

A 3D reconstruction algorithm is implemented. This algorithm uses the movement of the edges compared to the movement of the camera to estimate the 3D position of the edges.

This implementation is tested extensively. The limitations of the system is found and solution to those proposed.

Keywords : 3D reconstruction, feature detection, edge tracking.

Preface

This thesis shows how Gunnar Hardarson, s991208, solved the problem of 3D measurement using camera. This project is written at the department of Informatics and Mathematical Modelling (IMM) at Technical University of Denmark (DTU) during the period March to October 2005.

I would like to thank my supervisors Henrik Aanæs and Jens Michael Carstensen for good suggestions, ideas and comments throughout the project.

Contents

Chapter 1

Introduction

1.1 Quality Control

In today's mass production society, with often great distances between production and assembly, the need for a good quality control system is great.

The most easy-to-use quality control is the human being. For many years, people have worked with quality control, looking at objects running past them on an conveyor belt, visually inspecting the objects. This approach has several disadvantages for the employees, and human error increases as time goes by. Automation in this field is of great interest, due to better work conditions and higher precision.

Mainly two types of quality control systems exist. Contact and non-contact systems. In a contact system, a censor touches the objects in various places, registering where there is resistance. These points are compared to the points on a build in model of the object. If those points are in agreement with the model, the object is good.

A non-contact system can be made with the help of camera vision. Many variations exist. One or several cameras can be used and sometimes lasers. The advantage of this solution is that the object is never touched during inspection.

1.2 The Problem

A three dimensional (3D) measurement system, used for quality control is to be designed and prototyped.

This system will use a single camera moved around an object. The object is placed inside a cage, where a robot arm moves a camera around the object. 3D inference will be made based on this input.

1.3 Quality Control Systems

Various quality control systems have been made trough the years, with different approach. As mentioned earlier quality control systems can be categorized into two groups; a contact and non-contact systems. A contact system is a system where the object to be controlled is contacted physically by a censor.

An example of a contact system is the one used in the railway construction industry [1]. The erection of the four biggest modules, the floor, roof and two walls needs big precision and the working environment is hazardous. Instead of having workers positioning and welding the modules in place, a contact system is made, positioning the modules and welding, without any human interaction.

As knowledge in image analysis and computer science has grown the use of image based quality control systems has increased. These systems are non-contact systems. Cameras are used to measure the object, either moving the camera or the object. The results are often compared to an underlying CAD model. These systems use many different techniques, of gathering data for 3D reconstruction. Some techniques are mentioned below.

1.3.1 Laser Systems

A laser system projects laser beam over the object to be measured. The beam is reflected on the surface on the object and tracked. This way one can get very detailed information of the surface structure of the object.

An advanced CAQ (Computer Aided Quality) system using different laser approaches is presented in [2].

1.3.2 Marker Systems

In marker systems, marks are put on the object and detected by the system. It recognizes the shapes of the markers and can thereby estimate the position of each marker.

An example of a marker system is when catching the movement of a complex non-rigid body like when a human is walking. This topic is discussed in [3]. Markers are placed on different places, on the body, and then tracked, using multiple cameras. This technique is used in gait analysis.

1.3.3 Using Feature Detection

In the before mentioned non-contact systems, structured illumination or markings on the object to be measured, is used. In the current project, this is not the case. The intention is to make a quality control system which uses features in the objects. This way, the only hardware needed is a camera. A outline of the thesis is given in next section.

1.4 Outline

A robot moves a video camera above an object. A framegrabber is used to collect a sequence of images. This image sequence is used to reconstruct the depth in the scene.

The thesis can be divided into four parts.

The first part is feature detection. Here three methods are considered and compared. One method is chosen and used. This feature detection must capture the essential features in an object.

The second part is feature tracking. In order to detect movement in the scene, features have to be tracked. Here the features are tracked through the image sequence.

Third part covers the 3D reconstruction. Here the features that have been tracked through the images are used to reconstruct the depth in the scene and to find the construction of the object.

The fourth part of the thesis deals with extensive testing of the system. Both the functionality and how it performs under different situations is tested.

Chapter 2

The system

2.1 Overview

In this chapter the system is described, both the theory behind it and its implementation. In Figure 2.1 an overview of the process is shown.

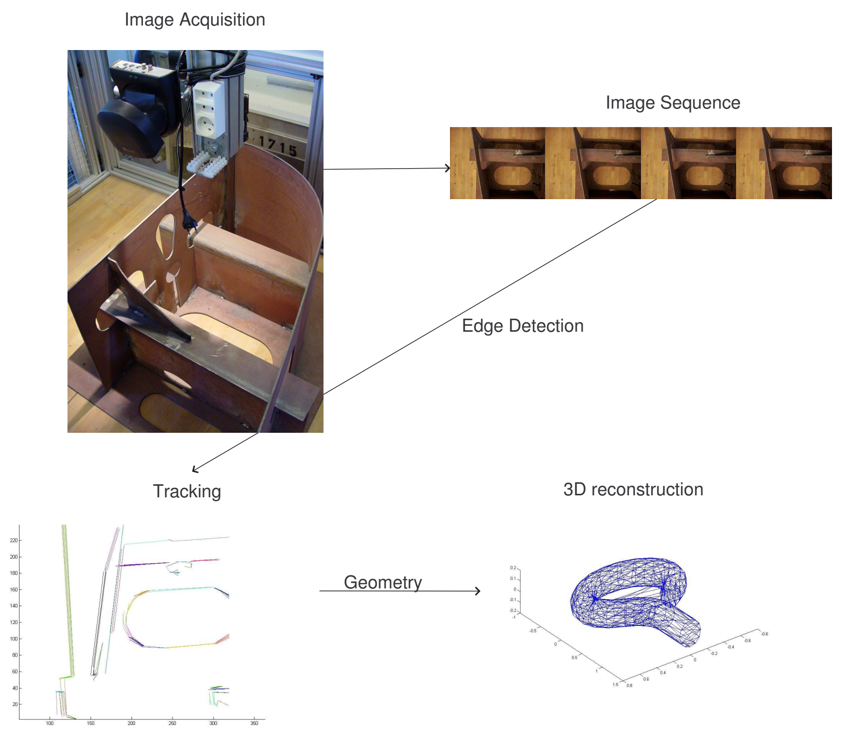


Figure 2.1: Overview of the process

First data is gathered by taking images. This is done with a robot arm moving a camera around the object to be measured. Using a robot arm the movement and position of the camera is known. This can be used in the computations. A frame grabber is used to grab individual frames. A photograph of the robot is shown in Figure 2.2. More extensive text and illustration of the robot is found in [9].

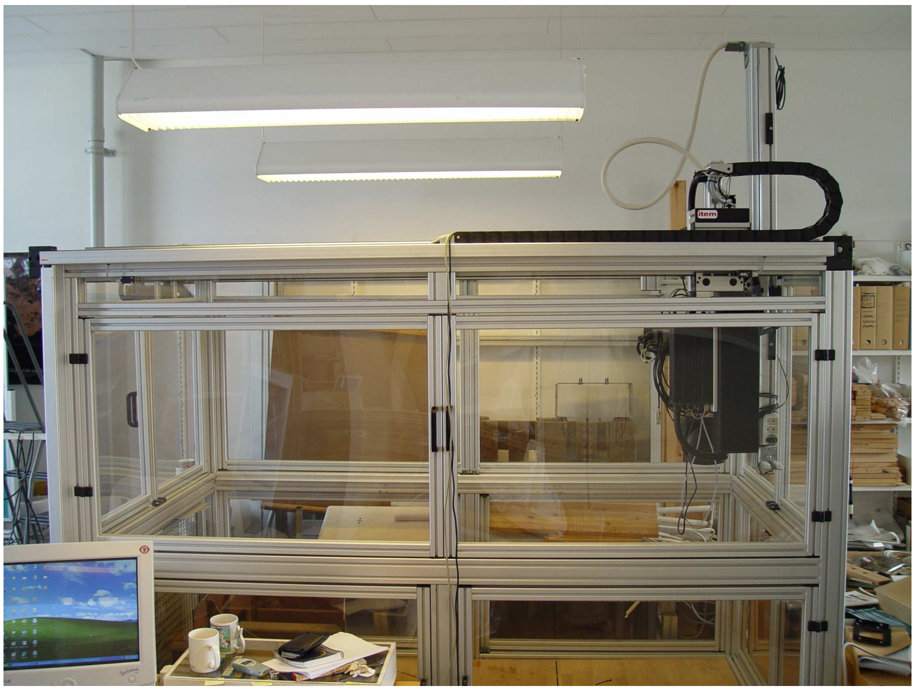


Figure 2.2: The robot. The arm can be moved in three directions

The consecutive images make an image sequence, where the camera has translated in small steps between images. This movement is to be used to reconstruct the depth in the scene.

For each image, features are detected. The features are then tracked between images, to detect the movement in the scene. The features detected have to describe the object very well.

Using the information from the above mentioned procedures, the geometry in the scene is estimated. From this estimation, a qualitative guess of the 3D construction in the scene is made.

2.2 Data

Three datasets are made. These are

- **Dataset1**. A simple synthetic dataset. 6 vertical lines are drawn and moved differently between 20 images. The movement is sketched in Figure 2.3.

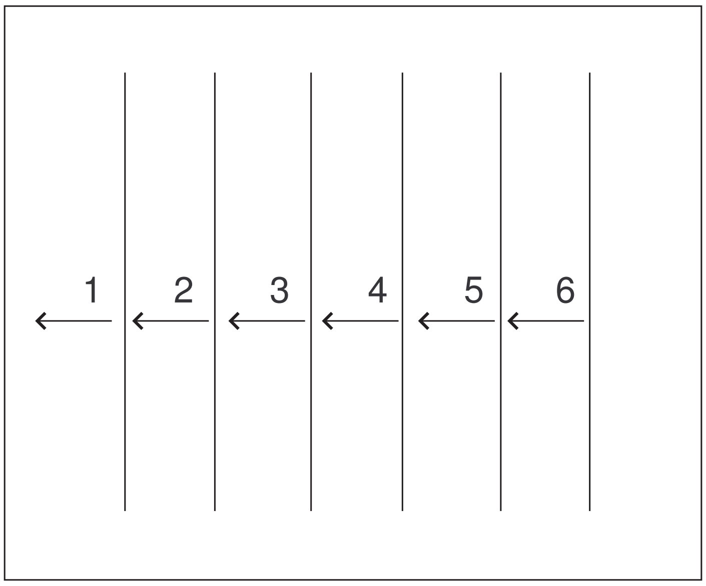


Figure 2.3: Different movement of the lines. The numbers represent the movement in pixels

- **Dataset2**. Images taken from the robot, as the arm moves in the x and y direction. The object is a black box on a white background. The edges are colored white. An example of the dataset is shown in Figure 2.4.
- **Dataset3**. Images taken from the robot, as the arm moves in the x-direction. The object is a model of a ship. An example of the dataset is shown in Figure 2.5.

These datasets will be used through out the report when testing the system. The datasets represent a typical input to the system.

2.3 Feature Detection

In order to make a system as described above, one needs to detect features in the object. Feature detection is an essential part of 3D reconstruction.

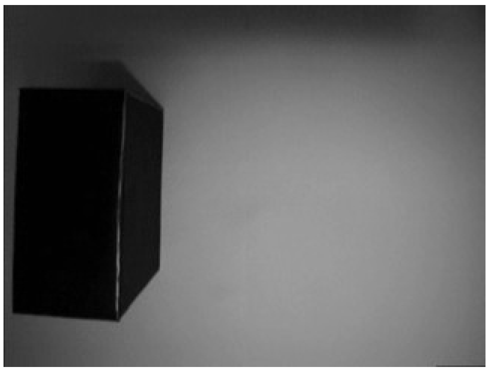


Figure 2.4: Image from Dataset 2

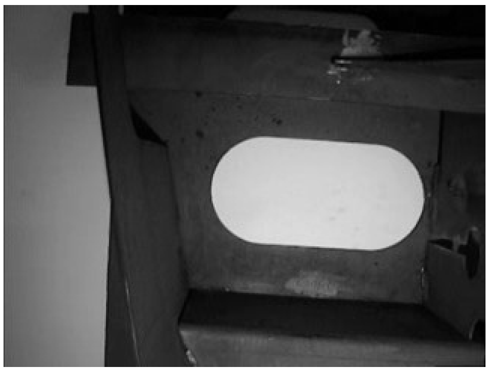


Figure 2.5: Image from Dataset 3

Features in an image can be of various kinds. The most obvious ones are corners, edges, shadings and colors. Shadings and colors are very hard to describe and even harder to compare as they often reach over large areas of the image. On the other hand, corners and edges are easier to describe and compare. This is due to a sudden swift in gradients in an image around corners and edges. Some methods of feature detection are discussed below.

2.3.1 Edge Detection

The present system is most likely to be used in the welding industry or dealing with objects that are made of straight plates, rather that curved objects like a chassis of a car. Due to this edges are a good feature to use. An edge is a well defined form, and not likely to be misidentified.

Examples are taken from Dataset 2 and 3 to compare the methods.

Edges are detected in a random image from Dataset 2. The results are shown in Figure 2.6.

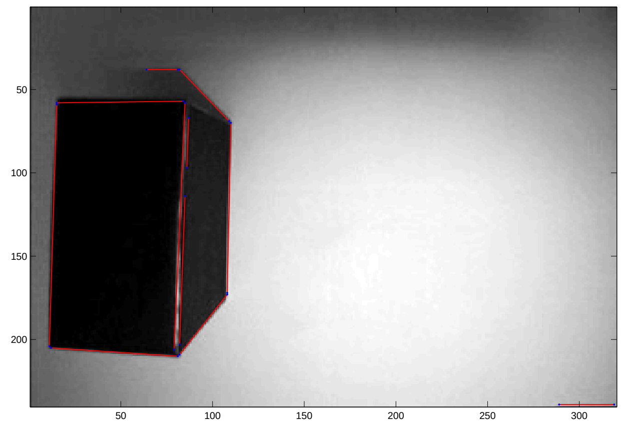


Figure 2.6: Edges found

The edges detected in Figure 2.6 describe the object well, with one exception though, the shadows. These will always be a problem.

Figure 2.7 shows the result from detecting edges in a image taken from Dataset 3. One can come to the same conclusion here about the edge detection. It detects the most obvious edges, but still the shadows are a problem. Notice the surface damage just below the big hole in the bottom of the model. No edges are detected there.

2.3.2 Corner Detection

Corners are detected in the same image from Dataset 2. The results are shown in Figure 2.8.

The problem with corners are that they do not necessary describe an object very well. Corners can be found on surfaces, which may lead to misidentification of a real corner on the object. In Figure 2.8 are some obvious misidentification's of corners which can not be used for 3D reconstruction.

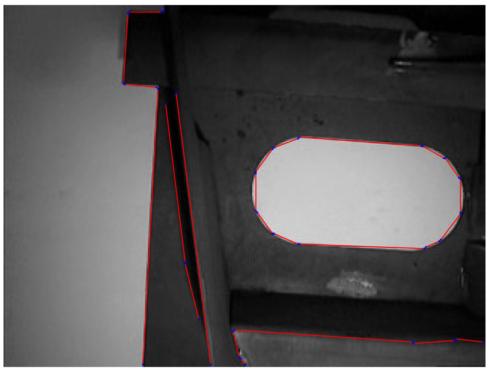


Figure 2.7: Edges found

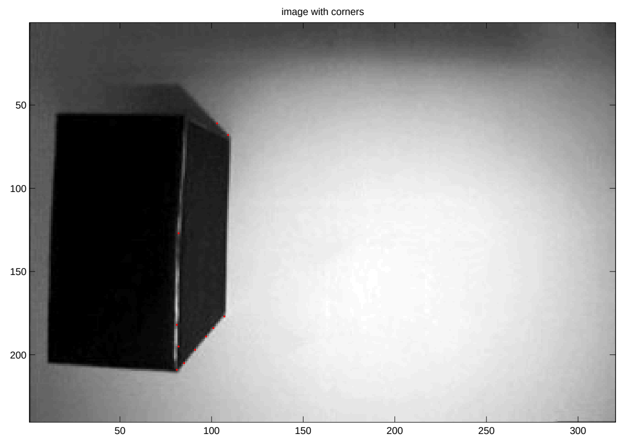


Figure 2.8: Corners found

Figure 2.9 shows an image taken from Dataset 3 and corners found. Notice the corners found on the surface damage just below the hole in the bottom. These corners can not be used for 3D reconstruction.

An excellent corner detection algorithm is presented in [5].

2.3.3 Optical Flow

Optical flow is a way of estimating the movement in a scene. One stereo-image is warped into another and the velocity field is computed. This velocity

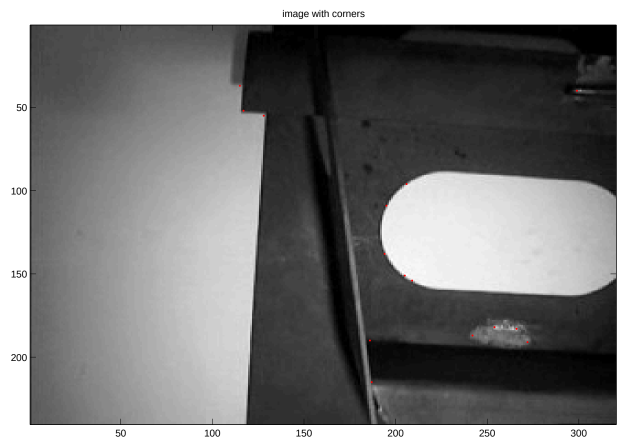


Figure 2.9: Corners found

field represents which parts of the scene are moving and how fast. This can be used as a feature detection.

A freely available software in C is used for the velocity field calculations[1]. The software package returns the velocity fields in the form of gray-scale images. The pixels are coded in the following way

- Gray pixels represent no movement
- Dark pixels represent movement in the negative direction
- Light pixels represent movement in the positive direction

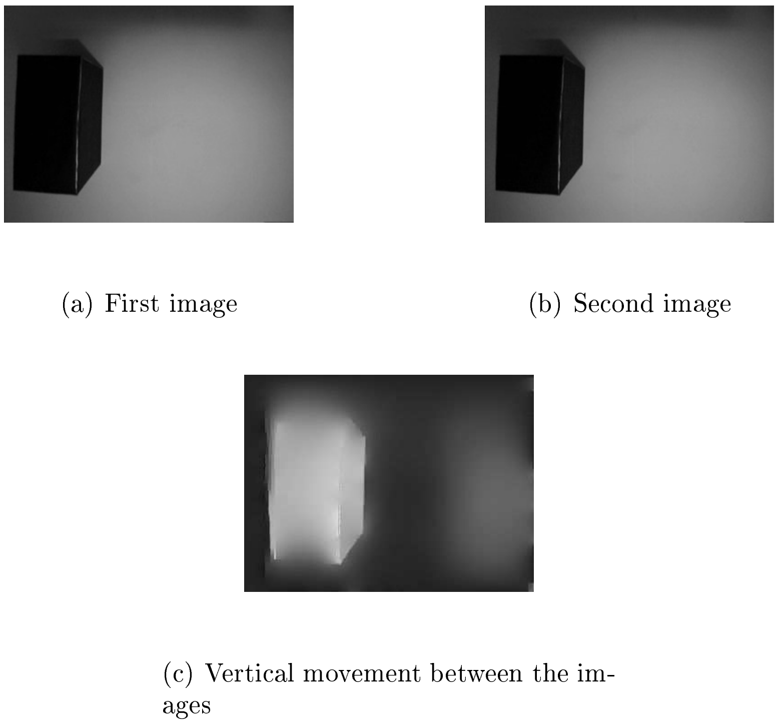
Two images are returned, one for the horizontal direction (x) and one for the vertical direction (y).

Some experiments were made with the optical flow to check its potentials and limitations.

Images were taken from Dataset 2. The images are shown in Figure 2.10 (a) and (b). The result is shown in Figure 2.10 (c). The method does well finding the movement in the scene, where the box is shown in white pixels, which represent movement in the positive direction.

But how well does the method cope with larger movements in the scene? Next test is done with images taken where the camera has translated more. The images are shown in Figure 2.11 (a) and (b). Observing Figure 2.11(c) one can see that the method needs the images to be taken close to each other.

[1]`http://www.cs.brown.edu/people/black/`


(a) First image (b) Second image

(c) Vertical movement between the images

Figure 2.10: Images and result from optical flow

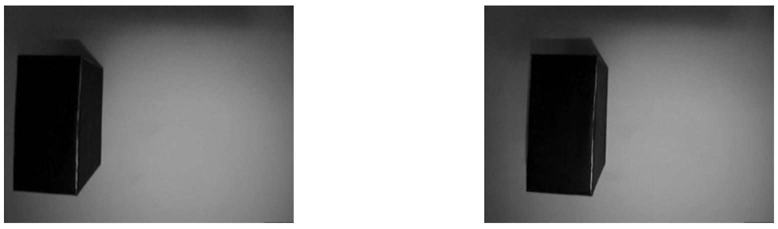
Two images are taken from Dataset 3 and the optical flow computed. The images are shown in Figure 2.12 (a) and (b). The result is shown in Figure 2.12 (c). Observing Figure 2.12 (c) some features are detected, but the algorithm has some trouble finding the direction the objects are moving in. This can be due to complexity in the scene.

More extensive text and implementation on optical flow can be found in [8].

2.3.4 Choosing the method for this project

The methods discussed above are all fitted for feature detection, but only one is to be used in this project. The main criteria for method selection were to find the most describing features, that are easy to track.

Edges are the features that describe an object the best and are easy to detect and track. Comparing Figures 2.9 and 2.7 showing the results from edge detection and corner detection one can see that the information gathered in Figure 2.7, by the edge detection method is much greater than the information in Figure 2.9. The edges found describe the object better.


(a) First image

(b) Movement between images is greater now

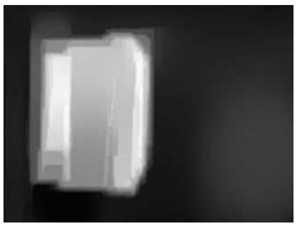

(c) Vertical movement between the images

Figure 2.11: Optical flow with greater distance

The same is when using a simpler object as in Dataset 2. The corners found in Figure 2.8 can not be used for 3D reconstruction.

The features found using optical flow were good when the movement was small, (Figure 2.10), but as the movement increased, (Figure 2.11), the method performed worse. The method also fails when the object gets more complex as in Figure 2.12.

Furthermore in order to detect features in a scene using optical flow, there has to be movement in the scene, and the detection depends on this movement. This makes this method less appealing.

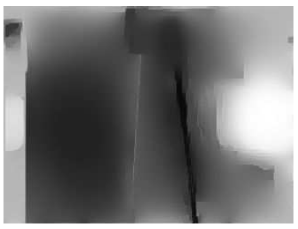
Considering the above the edge detection method in chosen. The positive sides of edge detection are listed below.

- Describes an object well
- Easy to detect
- Easy to track

As the chosen feature detection method is edge detection the following assumptions about the objects to be measured have to be made

(a) First image

(b) Movement between images is greater now



(c) Vertical movement between the images

Figure 2.12: Optical flow used on Dataset 3

- The object has to be described well by it's edges
- The edges have to be visually easy to detect
- The edges have to be straight

2.4 Edge Detection

As the chosen method is edge detection, this method is explained in this section more extensively.

To detect edges the sudden swift in the gradients in an image is used. Before the gradients are computed the image is smoothed with a Gaussian filter. This is done to eliminate noise in the image. Gradients are calculated both horizontally and vertically in the following way.
Vertically:

$$G_v(j,k) = A(j,k+1) - A(j,k-1) \tag{2.1}$$

Horizontally:

$$G_h(j,k) = A(j+1,k) - A(j-1,k) \tag{2.2}$$

Where j and k are row and column pixel indices. A is the matrix representing the image. G is the gradient matrix. The vertical (G_v) and horizontal (G_h) gradients are then combined into one gradient matrix (G) by (2.3).

$$G = \sqrt{[G_h]^2 + [G_v]^2} \tag{2.3}$$

After computing the gradients, the pixels with the highest probability of representing an edge are sorted out with a threshold. The edges are thinned to the width of one pixel and segmented into lines.

2.5 Implementation of the Edge Detection

A freely available `Matlab` implementation[2] was applied for the edge detection. What it does is listed below.

1. Load image
2. Find edges
3. Link edge pixels together
4. Make list of edges

These steps are discussed in more detail below.

2.5.1 Loading the images

This is a standard procedure, load the image and make it gray-scale. The system works only with gray-scale images. An original image, taken from Dataset 3 is shown in Figure 2.13.

2.5.2 Finding the edges

An edge detection algorithm is used to detect the edges in the image. The following call is used for this

```
edgeim=edge(im, 'canny', thresh, sigma).
```

The parameters are explained below

- The output `edgeim` is an image with the edges thinned to a single pixel.

[2] `http://www.csse.uwa.edu.au/ pk/Research/MatlabFns/`

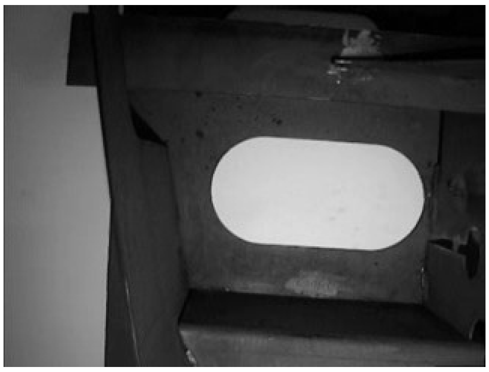


Figure 2.13: An original image, taken by the robot

- `im` is the image to be detected, in gray scale.
- `canny` is the edge detection algorithm used.
- `thresh` is a two-element vector where the elements are low and high thresholds. These thresholds are used after a non-maximum suppression, which suppresses all pixels not at a maximum to a non-edge pixel. The algorithm sweeps over the remaining pixels, suppressing the pixels below the lower threshold to a non edge pixel, and those above to an edge pixel. Those in-between are made either edge or non-edge pixels, depending on if there is a path from this pixel to an edge pixel.
- `sigma` is the standard deviation of the Gauss filter used to smooth the images.

Figure 2.14 shows the edges (white) found from the original image, shown in Figure 2.13.

2.5.3 Linking edge pixels

The edge-points are linked together, forming sequential list of edge-points. This is done in the following call
`[edgelist, labeledgeim]=edgelink(edgeim, l)`.
The parameters are explained below

- `edgelist` is a cell array, where each cell contains a list of edge pixels.

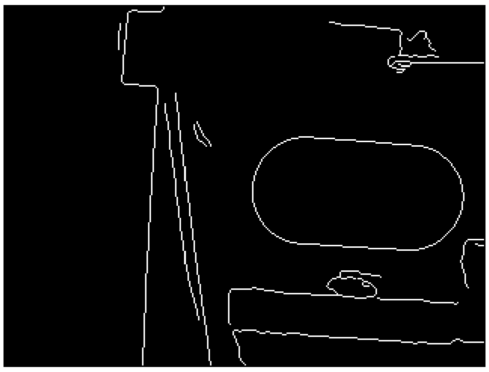


Figure 2.14: The edges found from Figure 2.13

- `labeledgeimage` is the edge-image, with different edges colored in different colors.
- `edgeim` is the edge-image returned from the `edge` call mentioned earlier.
- `l` is the minimum length of edges to be detected. Edges with length below this threshold are not classified as edges.

Figure 2.15 shows the edges linked together and segmented.

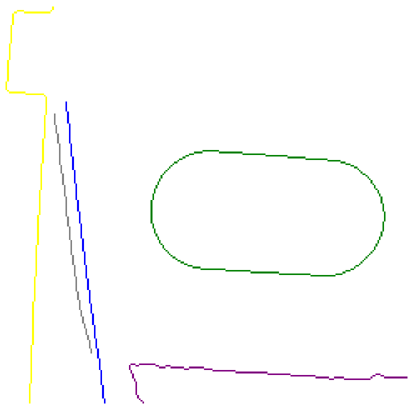


Figure 2.15: The linked and segmented edges from the edges in Figure 2.14

2.5.4 Making the list of edges

The last thing the edge detection algorithm does is to make straight lines from the edges found. This is a task of finding the lines, and returning the endpoints.

The command used here is
`seglist=lineseg(edgelist, tol, angtol,linkrad)`.
The parameters are explained below

- `seglist` is the list of endpoints. It is on the form

$$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & y_{n1} & x_{n2} & y_{n2} \end{bmatrix}$$

  Where (x_{11}, y_{11}) and (x_{12}, y_{12}) are the two endpoints of the first line.

- `edgelist` is the cell array returned from the `edgelink` command.

- `tol` is one of three restraints, it is an upper threshold on the maximum deviation from original edge.

- `angtol` is a threshold on difference in angle.

- `linkrad` is a measure of how close endpoints are allowed to be before they are merged.

The line-image is shown in Figure 2.16, along with the original image. One can see that most of the important edges are detected.

2.6 Edge Tracking

The human vision system constructs 3D view from two 2D images, one from each eye. The difference between objects in the two images is used to reconstruct the depth in the scene. This concept is used in 3D reconstruction systems.

In order for such a system to achieve 3D information, there has to be either two cameras or a movement, either in the scene or the camera. Moving the camera is chosen here. More information can be gathered this way. As the camera is moved, it captures video sequence of the object. Then using a frame grabber, individual images are obtained. Taking two images, from two consecutive frames, corresponds to taking two images with an ordinary camera, side by side. But as the camera is moved for some distance, more

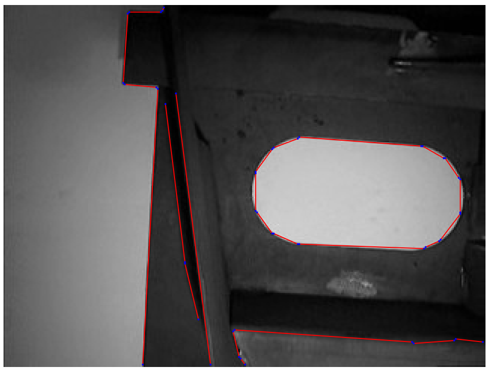


Figure 2.16: The lines found in the original image

information can be gathered and thus, a more accurate model of the object can be constructed.

In order to measure the movement of the edges through the scene, an edge tracking algorithm is implemented. This application is able to track edges through a series of consecutive images.

The edge detection algorithm described earlier, returns a list of edges on the form

$$\begin{bmatrix} x_{11} & y_{11} & x_{12} & y_{12} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & y_{n1} & x_{n2} & y_{n2} \end{bmatrix}$$

for each image. The tracking algorithm compares two lists of edge endpoints, searching for matches within a given range. If there is a match, the index of the two edges is inserted into an index-matrix. If there is no match in next image in the sequence a zero is inserted into the matrix. When the matrix is made, the algorithm runs through it again. Where there are zeros, which means that an edge is not matched, the edge is sought in the rest of the images. This is done to prevent losing edges if they disappear in one image and reappear later.

Edges that appear after the first image is then dealt with last. An edge found in an image after the first one, which has not been matched is sought for in the rest of the images. This is concatenated to the end of the index-matrix. This completes the tracking algorithm, which by now should be able to track most lines through a number of images, assuming not too big translation between images.

The index matrix is shown below

$$\begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,m} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,m} \\ \vdots & \vdots & u_{i,j} & \vdots \\ u_{n,1} & u_{n,2} & \cdots & u_{n,m} \end{bmatrix}$$

Where $u_{i,j}$ denotes the jth edge in image i.

The results of running the algorithm on several images, taken from Dataset 2 is shown in Figure 2.17.

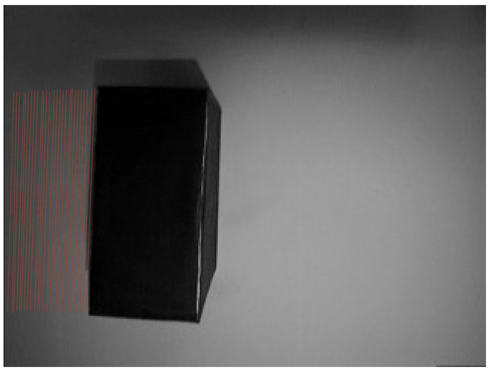


Figure 2.17: The edge is tracked through a number of images

2.7 Implementation of the tracking algorithm

In this section the implementation of the tracking algorithm is described in more detail.

The algorithm runs as follows

- Matches the first two images
- Fills in the indeces where edges have occluded and reappeared
- Runs through the index matrix matching lines not seen in the first image

2.7.1 Matching the first two images

The edges in the first image are numbered as they appear in the edge-matrix. These are then matched to the edges in the second image. The matching process is made in the following steps

- Make a box around each endpoint in the first image
- Search within those boxes for the endpoints in the other image
- Both endpoints must match both endpoints in the other image

This is done by the following call

```
index = match1and2(totalseglist, box, sizes);
```

where

- `index` is the list of matches. The first row consists of numbers from one to the number of edges in the first image. The second row represents the matches to the lines in the second image.
- `totalseglist` is a 3D list of edges, for all the images.
- `box` is the size of the search-box. Setting this box too small may risk the algorithm to fail to find the right edges, too big will slow down the execution time. For the camera used in this project the recommended size is 5 pixels.
- `sizes` is the sizes of the edge-lists.

2.7.2 Matching the edges in the rest of the images

Now the index matrix is two rows. The rest of the edge-images are now matched to each other, using the numbering from the first image, and the endpoint coordinates from the preceding image. This way, the camera can move over a considerable distance, without losing track of the edges. This is done in the following command

```
newindex=moreindex(totalseglist, finalindex)
```

where

- `newindex` is a two row vector, where the second row represents the edges matched to the ones in the first image, using the coordinates from the preceding image.
- `totalseglist` is a 3D list of edges, for all the images.

- `finalindex` is the index matrix so far. From this matrix the coordinates of the edges in the last image are taken.

The above command has to be called $N - 2$ times where N is the number of images, to capture all of the remaining images.

2.7.3 Filling in the occluded edges

Now edges that are lost in an image, and reappear few images later are detected and matched. If an edge is occluded the algorithm so far does not look for it in the remaining images. This is mended by the function
`newindex=fillin(finalindex, totalseglist);`

- `newindex` is the newest version of the index-matrix, now with occluded edges.
- `finalindex` is the index made from the previous call.
- `totalseglist` is a 3D list of edges, for all the images.

2.7.4 Matching the lines appearing after the first image

As the camera is moved, the lines that are on the first image, are not all the lines the camera detects. Edges appearing on later images must be taken into account in the computations. This is done in the following command
`totalindex = extralines(newindex, totalseglist);`
where

- `totalindex` is the latest version of the index-matrix. This version includes occlusions and new appearing lines.
- `newindex` is the index from the previous call.
- `totalseglist` is a 3D list of edges, for all the images.

This completes the index matrix, edges have been tracked throughout the whole sequence.

2.8 Geometry

Now that the edges have been identified and tracked, the next step is to reconstruct the depth in the scene. As mentioned earlier, the human vision-system constructs the 3D in a scene from the difference between the 2D images, from each eye. This is used in the 3D reconstruction algorithm. The information about the movement in the scene is used here to get 3D information, or the depth in the scene. The movement of the camera is also used.

This problem is closely related to that of the structure from motion problem. The difference is that in the current problem, the movement of the camera, translation and rotation is known. This makes the computations more simple.

Figure 2.18 shows a sketch of the problem.

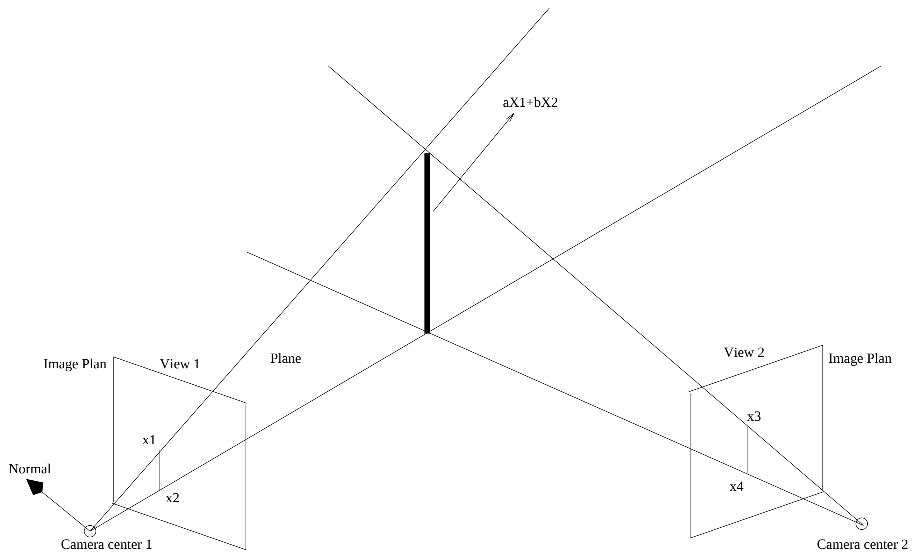


Figure 2.18: Sketch of the problem. The edge is observed from two cameras. x are the endpoints of the edges observed.

From the edge images taken from the image sequence, only 2D information is collected. No information is given about the position in space of the edge. This setup corresponds to $View$ 1 in Figure 2.18. The only information is that the line lies on a plane, made from the center of the camera, and the edge-line, seen on the image. Now, moving to $View$ 2, the same edge is detected in that image. Making a plane the same way as before, the two planes intersect in space. The line in 3D space is found. To be more accurate, more views and planes are made, to get a better estimate of where the planes intersect. In the ideal world, where there is no noise in the images, the intersection of all the planes, would be exactly at one line. In the real world, this is rarely, or never the case. To compensate for this, the line which is closest to describe the intersection of the planes is sought and set to be the reconstructed line in 3D. From this it is easy to see that more information (images) give more planes and more accurate estimate of the line.

2.9 3D Reconstruction

The reconstruction algorithm is described more thoroughly step by step below.

2.9.1 Computing the Camera Matrix

The camera matrix maps world points $\mathbf{X}$ to image points $\mathbf{x}$ according to

$$\mathbf{x} = \mathbf{PX} \tag{2.4}$$

The camera matrix $\mathbf{P}$ is written on the form

$$\mathbf{P} = [\mathbf{KR}| - \mathbf{Kt}] \tag{2.5}$$

where $\mathbf{K}$ is the camera calibration matrix, $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is the translation vector.

The camera calibration matrix is given by

$$\begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

where

- α_x is the focal length in the x-direction, in pixels
- α_y is the focal length in the y-direction, in pixels
- s is the skew factor
- x_0 and y_0 is the principal point

R is given in

$$\begin{bmatrix} r_{1,1} & r_{2,1} & r_{3,1} \\ r_{1,2} & r_{2,2} & r_{3,2} \\ r_{1,3} & r_{2,3} & r_{3,3} \end{bmatrix} \tag{2.7}$$

where

- $r_{1,1} = \cos(\Theta)\cos(K)$
- $r_{1,2} = -\cos(\Theta)\sin(K)$
- $r_{1,3} = \sin(\Theta)$
- $r_{2,1} = \cos(\Omega)\sin(K) + \sin(\Omega)\sin(\Theta)\cos(K)$
- $r_{2,2} = \cos(\Omega)\cos(K) - \sin(\Omega)\sin(\Theta)\sin(K)$
- $r_{2,3} = -\sin(\Omega)\cos(\Theta)$
- $r_{3,1} = \sin(\Omega)\sin(K) - \cos(\Omega)\sin(\Theta)\cos(K)$
- $r_{3,2} = \sin(\Omega)\cos(K) + \cos(\Omega)\sin(\Theta)\sin(K)$
- $r_{3,3} = \cos(\Omega)\cos(\Theta)$

The parameters Ω, Θ and K are the rotation around the $x-$, $y-$ and z direction respectively. These are given in world coordinates. The camera is not rotated at all, so all those parameters are zero. Inserting zero in the equations, the identity matrix **I** is reached. This simplifies (2.5) to

$$\mathbf{P} = [\mathbf{K}| - \mathbf{K}\mathbf{t}] \tag{2.8}$$

The translation vector is given as

$$\mathbf{t} = \begin{bmatrix} \delta x \\ \delta y \\ \delta z \end{bmatrix} \tag{2.9}$$

where the element represent the translation in each direction.

2.9.2 Finding the Camera Centers

In order to find a plane in space one needs three points in space. In the current setup, three points are known, the camera center and two points on the edge-image.

The camera center can be computed from the camera matrix **P**, given by (2.8) and a translation vector **t**, given in (2.9). The world coordinate system is set to have the origo just above the center of the object. The xy plane is parallel to the image plane, and the z axis is set along the optical axis. The system is scaled so the distance from the camera center to the image plane is one unit. A sketch to demonstrate this is shown in Figure 2.19. The center of the camera is found as **c** where $\mathbf{Pc} = \mathbf{0}$.

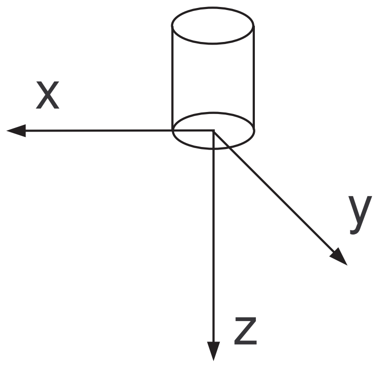


Figure 2.19: Definition of the world coordinate system

2.9.3 Finding the Planes

The two endpoints of each edge are used as the two points left to define the plane. In order to retrieve the world coordinates for these points, the camera matrix is used. This time the translation vector is the zero vector, due to the fact that we are now working in the image-coordinate system. The equation for mapping from world coordinates to image coordinates is given by (2.4). This equation must be solved with respect to **X** in order to map from image coordinates to world coordinates. The solution to (2.4) can not be found directly as **P** is not a square matrix, so the following actions are taken.

1. The equation can be written as $\mathbf{x} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{X}$
2. K is a square matrix, so it can be inverted $\mathbf{K^{-1}x} = [\mathbf{R}|\mathbf{t}]\mathbf{X}$
3. Given that the rotation matrix is the identity matrix and the translation vector is the zero vector the equation looks like
$\mathbf{K^{-1}x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$
4. Multiplying yields $\mathbf{K^{-1}x} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

This maps the 2D points to 3D, with the distance to the z plain set to 1 unit. The equation of the plane is on the following form

$$\mathbf{nx} = -\alpha \tag{2.10}$$

where $\mathbf{n} = [n_x, n_y, n_z]$ is the normal to the plain. α is the distance from every point on the plane to the origo, which is set as the position of the camera. This plane is found for each edge, in each image.

2.9.4 Finding the Intersection of the Planes

As said before, the line in space is found where the planes are closest to intersect. Two points are found in space where

$$\mathbf{Px} = \mathbf{0} \tag{2.11}$$

where $\mathbf{P}$ is the plane for each line, and $\mathbf{x}$ is a point in space. These two points span a line written on the form $l = ax_1 + bx_2$ where a and b are any number.

2.10 Implementation of the Geometry

The reconstruction algorithm takes as input the coordinates of matching lines. Finding the lines in 3D, the following steps are taken

- Find the camera matrix
- Find the center of the camera
- Find, in world coordinates, the edge endpoints
- Determine plane
- Find 3D points

2.10.1 Finding the Center of the Camera

As said before, the center of the camera is found where $\mathbf{Px} = 0$.
The following call is used for this

```
center=camcent(P)
```

where

- `center` is the center of the camera in 3D space.
- $\mathbf{P}$ is the camera matrix. The translation vector $\mathbf{t}$, is computed as the distance from origo.

Using singular value decomposition, the x is computed.

2.10.2 Finding the Edge Points

The edge points found on the edge-image are transformed into world coordinates using the camera matrix, with translation vector equal to the zero vector. The following call is used to do this

```
[p1,p2]=findedgepos(index,seglist)
```

where

- `p1` and `p2` are the endpoints in world coordinates
- `index` is a list of matching edges
- `seglist` contains the positions of the edges in image coordinates

2.10.3 Making the Plane

Given the three points in space from the previously mentioned methods, one can now make a plane. The following call is used

```
[n alpha] = makeplane(s1, s2, s3, ccenter)
```

where

- `n` is the normal vector to the plane
- `alpha` is the distance from a point on the plane to the origin. The point on the plane here is chosen to be the camera center.
- `s1`, `s2` and `s3` are the points that make the plane.
- `ccenter` is the camera center, used to compute alpha.

For each edge the $\mathbf{n}$ and α from the equation of the plane (2.11) is inserted in a matrix P. This matrix is used later to determine the line in 3D.

2.10.4 Finding the 3D Points

The intersection of all the planes, is found using the equation $\mathbf{Px}$=0. Where $\mathbf{P}$ is a matrix of planes, discussed in the previous section. This is done using singular value decomposition (svd). The *svd* returns points that are the closest to represent the line where the planes intersect.

2.11 Calibration

When taking a picture from a lens camera, some distortion will appear on the image. Due to this, these images are not suitable for measurements. The distance between elements in the center of the image, does not represent the distance between elements at the edges of the image. This can be mended by calibrating the camera. By camera calibration, the distortion in the image is measured. This is used later in the 3D computations.

The purpose of camera calibration is to compute the parameters of the inner orientation, the focal length, principal point, distortion and skew.

A complete Camera Calibration Toolbox for `Matlab` may be downloaded[3]. This toolbox is used to calibrate the camera.

First a calibration sheet is made. This sheet is a chessboard with equally sized squares, black and white. The sheet is glued on a rigid metal plate, due to the risk of the sheet bending under the procedure.

Twenty images of different angle and position is taken from the camera. A calibration sheet of 24×38 squares, with side-length of 10×10mm is tried out but found to be too small. A 16×24 squares, with side-length 16mm gives a better result. If the squares are too small, the corners of the squares are difficult to detect, even visually as the case was with the 10mm squares. If the squares are too big, the number of corners are too few for a good calibration.

Figures 2.20 and 2.21 show the original image, and the undistorted image.

The parameters computed by the calibration are shown in Table 2.1.

	x	y
Focal Length	380.60892	378.42038
Principal Point	153.27998	119.84683

Table 2.1: Parameters for the camera

Skew is 0.000141. The distortion parameters are $[-0.25888, 0.36657, 0.00110, 0.00005]$. The parameters are described in more detail below

- Focal Length denotes the length from the Projection Center to the Image Plane in pixels. The focal length is divided into x- and y direction because the pixels are not quadratical.

[3]`http://www.vision.caltech.edu/bouguetj/calib_doc/`

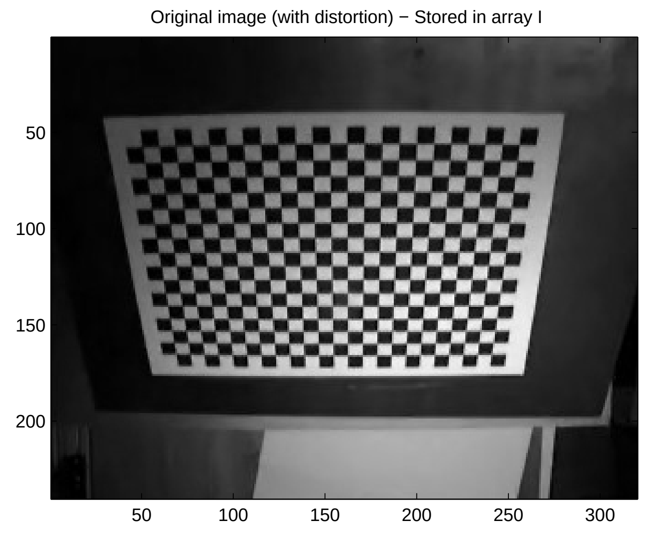


Figure 2.20: The calibration sheet used to calibrate the camera

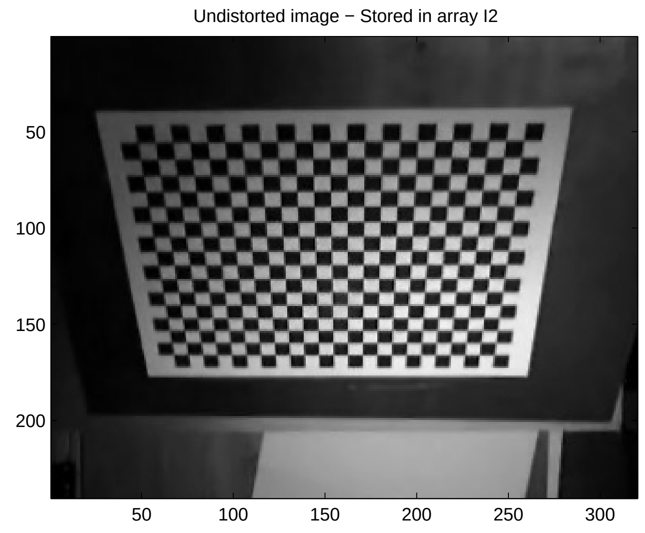


Figure 2.21: The undistorted image

- Principal Point is the coordinates where the perpendicular from the Projection Center intersects with the Image Plane

- Skew is a coefficient defining the angle between the x and the y pixel axis

- Distortion are the parameters of the distortion function

It is assumed here that the camera used does not have the "Auto-zoom" turned on. Auto-zoom, changes the focal length as objects in front of the lens is moved. This would have an erroneous effect on the computation of the focal length.

2.12 Constructing the scene

When constructing the scene physically, that is placing the object, choosing background, light and so on, many things have to be considered. Those are things that influence the end result. One can make a great 3D reconstruction algorithm, but lose some information to bad lighting or wrong background. One has to take into consideration what features in the object one wants to amplify. Some of these factors are discussed below.

2.12.1 Lighting

When detecting features in an object, an essential part is the lighting. Many aspects of lighting have to be taken into consideration. Some are

- Intensity
- Shadows

The intensity of the light has to fit the surface of the object. Too strong light can result in mirror effect in the case of metal surfaces. This could disturb the gradient calculations. Too weak light can result in bad illumination of the object, and describing features would not be detected.

Shadows in the scene can be detected as edges, which leads to a bad reconstruction model. This can be mended using either ambient light, or more easy, mounting the light on the same arm as the camera, so the parts that are being photographed are illuminated at the same time.

2.12.2 Background

An essential part of choosing background is that there should not be any detectable features in the background. A clean sheet where the junctions do not show, will result in a good feature detection. Other thing to be taken into consideration is the color of the background. It is of great importance that the color is opposite to the color of the object. This way the edges of the object are enhanced, leading to a better solution. This is particularly important when working with an object like the one in Dataset 3 where there is a hole in the object.

2.13 Ground Truth

For testing how well the reconstruction algorithm performs, a ground truth is necessary.

The object in Dataset 2 is plotted in `Matlab`. In the case of a more complex object, this would be done in `AutoCAD`, and imported to `Matlab` via the import-function `cad2mat`[4]. This plot is used for comparison of the output of the application. The plot is shown in Figure 2.22.

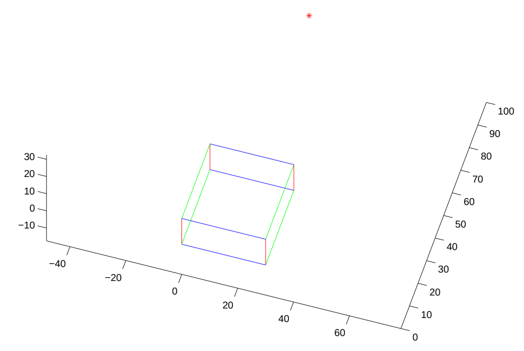


Figure 2.22: The box

[4]`http://www.mathworks.com/matlabcentral/fileexchange/`

Chapter 3

Tests

How well does the system work? How does the different parameters affect the outcome? Answer to these questions and similar is the subject of this chapter.

In order to tell how well the system works, an extensive test scheme is made and carried out. An overview of the tests is given in the following

- **Basic functionality** - some basic tests to see if parts of the algorithm work.
  - The points in 3D that form the line on the object should project on, or close to the line found on the edge image.
  - The epipolar line is used to see if the 3D point projects correctly to the next image.
  - A simple synthetic data, from Dataset 1, is used to see if the algorithm works with simple data.
- **Translation** Determine the value of the camera translation t in pixels and millimeters.
- **Tuning the Edge Detection parameters** - What effect does the constraint parameters in the edge detection have.
- **Baseline** - what effect does it have to change the length of the baseline?
- **Reconstructing the Box** - data from Dataset 2 is used to reconstruct the box.
- **Two directions** - Reconstructing edges where the camera moves perpendicular to the edge and where it moves parallel to the edge.

- **Noise in data** - translation of an edge is simulated with no noise and reconstructed.

3.1 Epipolar Test

Using epipolar geometry [7], a 3D point should be seen somewhere along the epipolarline from the other image. Knowing the translation of the robot one can predict the epipolarline. In the dataset used the robot arm moves only in the x direction. This means that the epipolarline lies at $y = k$, where k is the y coordinate for the 3D point projection in the first image. This is illustrated in Figure 3.1.

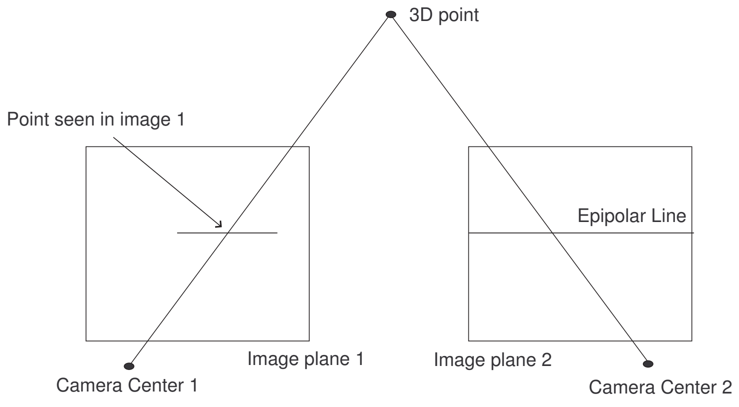


Figure 3.1: The 3D point should be seen along the epipolarline

The intersection of the second image-plane was computed and compared to the intersection of the first image-plane. The y-coordinates proved to be the same all the time.

3.2 Projection

When the points in 3D are found, they should project near the edge-line in every view when projected to the camera center.

This is tested using the distance between two lines. The equation for the shortest distance between two lines can be written in the following way

$$D = \frac{|\mathbf{c} \cdot (\mathbf{a} \times \mathbf{b})|}{|\mathbf{a} \times \mathbf{b}|} \tag{3.1}$$

where $\mathbf{a} = \mathbf{x_2} - \mathbf{x_1}$, $\mathbf{b} = \mathbf{x_4} - \mathbf{x_3}$ and $\mathbf{c} = \mathbf{x_3} - \mathbf{x_1}$.
$\mathbf{x_1}$, $\mathbf{x_2}$, $\mathbf{x_3}$ and $\mathbf{x_4}$ are explained in Figure 3.2. This was tested on various

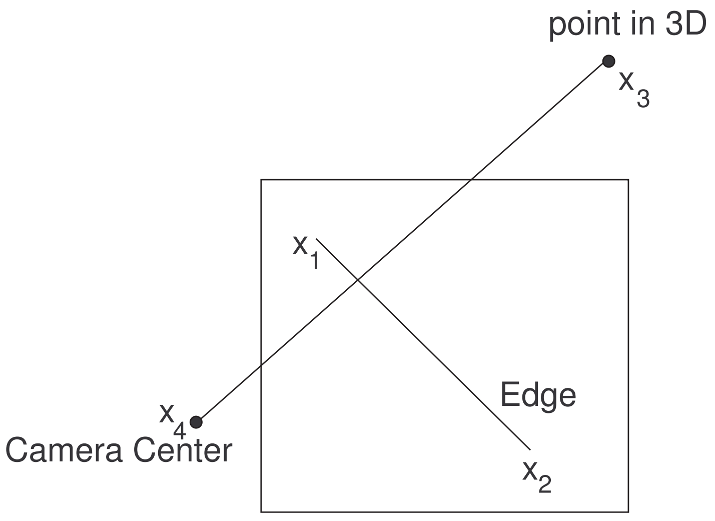


Figure 3.2: The distance between the lines should be minimal

datasets where the distance is found to be very small. This shows that the points project correctly back to the image plane, which implies that the position of the point in 3D space is correct.

3.3 Test on synthetic dataset

In order to see if the system works on a simple synthetic data, Dataset 1 was made. The data is constructed of 6 parallel lines, perpendicular to the movement of the camera. 20 images are made, where each line moves different amount in each image. This can be compared to when a person looks at a staircase from above, and moves to the side. The topmost steps will move the most, when the bottommost, will move the least. Figure 3.3 illustrates the setup.

The outcome is as predicted, a stair. The line that moves the least is the one most far from the camera, then step by step, closer to the camera. A screen-shot of this is shown in Figures 3.4 and 3.5. As seen the lines form a step in both the xy and yz planes, which shows that the 3D reconstruction works properly.

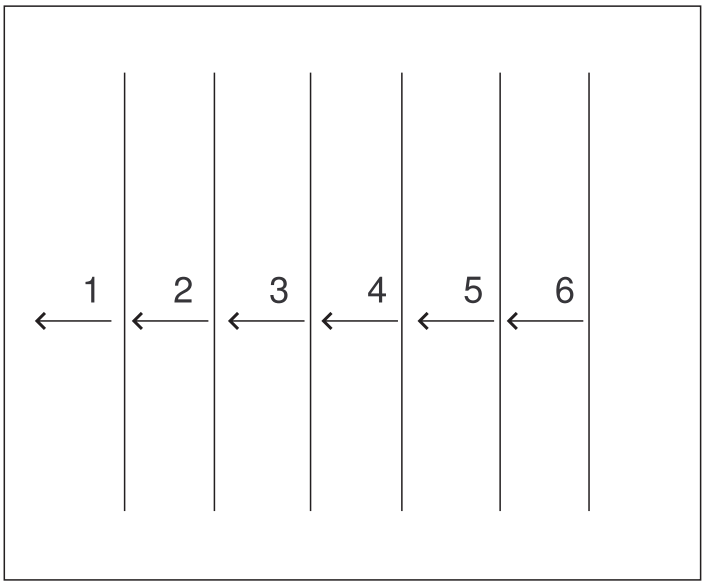


Figure 3.3: The numbers represent the number of pixels the lines are moved between images

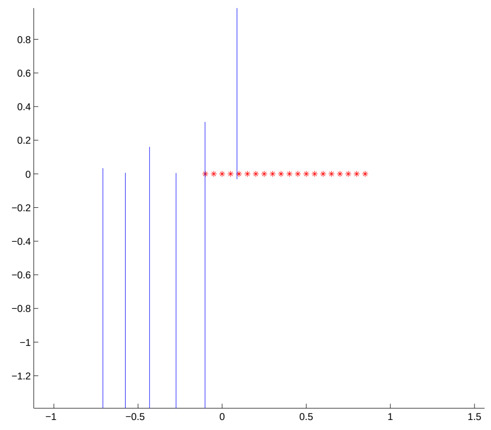


Figure 3.4: xy view of the steps. The red asterix's represent the cameras

3.4 Tuning Edge Detection

An essential part of the system is edge detection. Detecting the interesting edges is of great importance. Interesting edges are edges that describe the object well. The edge detection has some constraints that can be adjusted. In

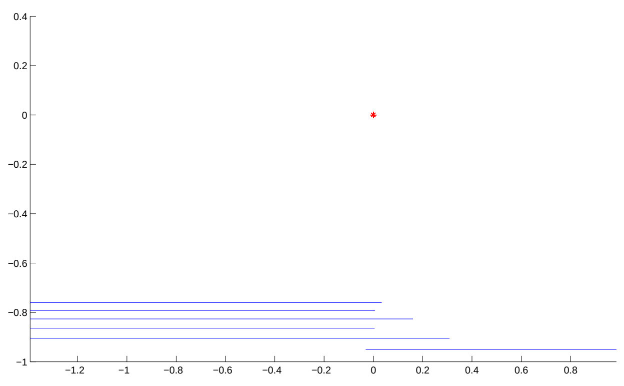


Figure 3.5: yzview of the steps. The red asterix's represent the cameras

this project it is crucial to detect long edges, that are easily tracked through a set of images.

The constraints are the following

- `tol`, an upper threshold on the maximum deviation from original edge.
- `angtol`, a threshold on difference in angle.
- `linkrad`, a measurement where endpoints is merged if they are closer than this threshold.

Two sets of values are tried out on data from Dataset 2. The values are listed in Table 3.1. The outcome of the different parameters are shown in

	Testset 1	Testset 2
`tol`	2	6
`angtol`	0.05	0.2
`linkrad`	2	6

Table 3.1: Values for edge detection

Figures 3.6 and 3.7. Notice the right-top edge on the images. The one with the smaller values is detected as two edges, while the one with the bigger values is detected as one edge. This is the preferred situation in this project, so the values in Testset 2 are chosen.

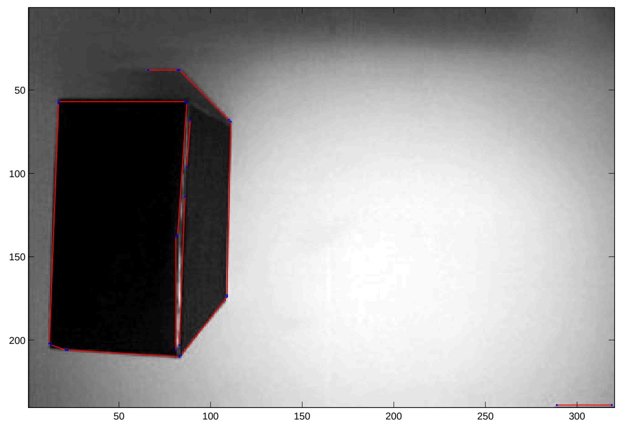


Figure 3.6: Edges found using values from Testset 1

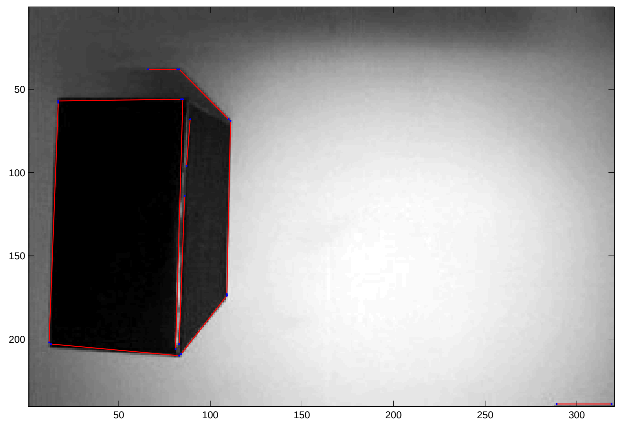


Figure 3.7: Edges found using values from Testset 2

3.5 Determine the translation

The translation of the camera is an important parameter when computing 3D. The value of t is determined in this section, both in pixels and millimeters.

An image sequence from Dataset 2 is used. A single edge is tracked through 71 images and the total distance in pixels is taken. This way the distance for each translation is calculated. The edges are shown in Figure 3.8.

To measure the translation in millimeters the robot arm was moved a

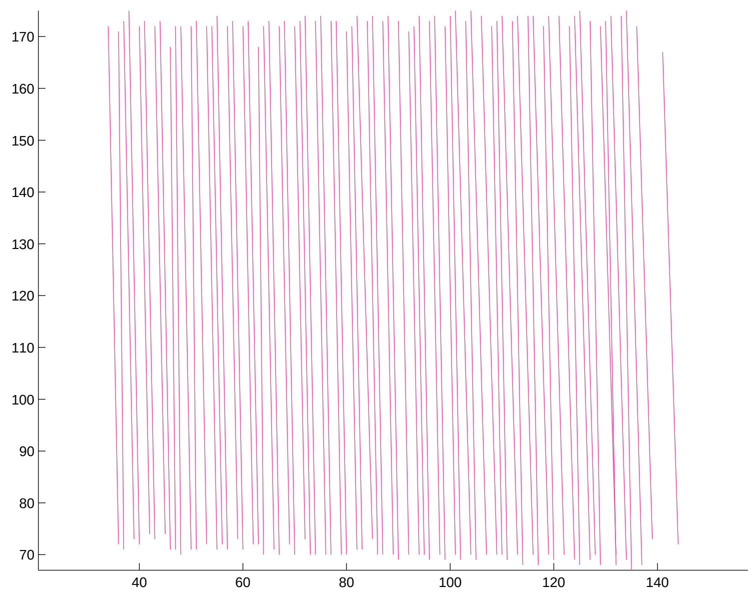


Figure 3.8: Edges found

certain distance, while taking a video sequence. Counting the number of frames the translation in millimeters is calculated. The images are taken with an interval of 5 frames.

The results are listed in Table 3.8.

	t
Pixels	1.44
Millimeters between images	4.1

Table 3.2: Translation of the camera

Looking at Figure 3.8 one observes that the lines “jump” in intervals of 2 or 3 edges. This is due to the uneven movement of the robot arm. This can be a source of error.

3.6 Baseline Test

According to [4] the optimal relation between the baseline and the distance between the object and camera should be

$$\frac{B}{D} = \frac{1}{3} \ \ to \ \ 3. \tag{3.2}$$

This is sketched in Figure 3.9.

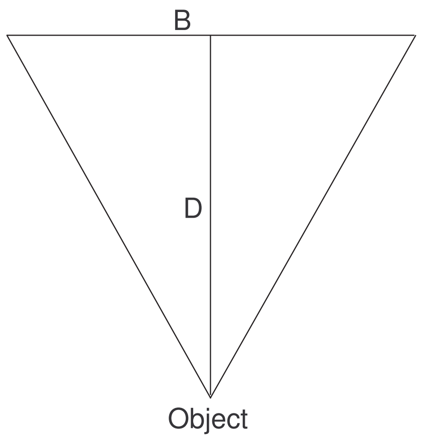


Figure 3.9: Relation between baseline and distance to object

A test on an edge on the box from Dataset 2 is made. The test is made the following way. The edge is detected and tracked through 135 images and reconstructed. Then an image is iteratively removed from the sequence until only few is left. For each removal, the 3D is reconstructed from the remaining edges. This way the baseline is shortened each time. The error is then calculated for each iteration.

As the edges can be of various lengths, it is a bit tricky to compute the error in a reliable way. The solution is to make a plane in $y = 0$, compute the intersection of the lines with this plane. The distance in the z-direction is computed between the intersections. Only the z-direction is used here as it is the important part of the reconstruction. The distance is plotted as a function of images removed. The plot is shown in Figure 3.10.

A translation of 135 images corresponds to 55.3 cm. The distance from the object to the camera is 100 cm, so the relation (3.2) is fulfilled. The theoretical limit is plotted in the figure as a red line. When the baseline passes the limits, the error increases drastically. This supports the relation.

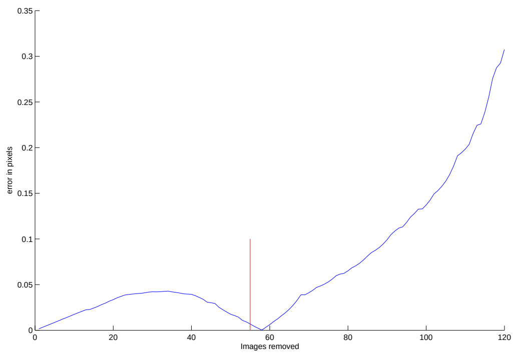
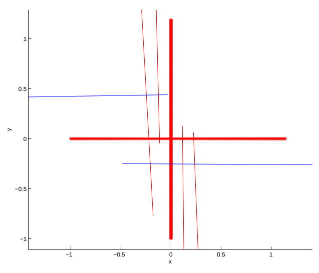


Figure 3.10: y axis is the error in pixels. x axis is the number of images removed

3.7 Reconstructing the box

Dataset 2 is used for reconstructing the box and checking the functionality, and limitations of the system with real data. First edges perpendicular to the x-direction are tracked and reconstructed using part of the dataset where the camera moves in the x-direction. Then edges parallel to the x-direction are tracked and reconstructed, using sequences where the camera is moved in the y-direction.

The xy-plane is shown in Figure 3.11 (a). The ground truth is shown in Figure 3.11 (b). Edges tracked in the x-direction is plotted in red, edges


(a) Reconstructed box

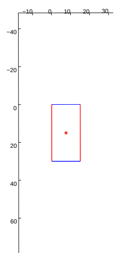
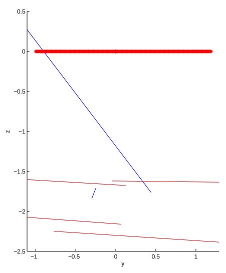

(b) Ground truth

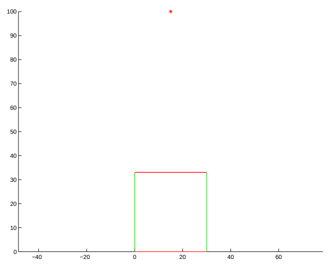
Figure 3.11: xy-plane

tracked in the y-direction is plotted in blue. The red asterix's are the cameras.

Figure 3.12 (a) shows the box in the yz-plane. This view is perpendicular to the edges tracked in the x-direction, or those plotted in red. These are fairly well reconstructed, showing two edges as the bottom edges, and two edges as the top edges. The reconstruction is right up to a scale factor.



(a) Reconstructed box



(b) Ground truth

Figure 3.12: yz-plane

Considering the z-position of the red lines in space, the box is measured and found to be 33 cm. The distance from the floor to the camera is 100 cm. From the image it can be seen that the two top lines are close to being $\frac{1}{3}$ of the distance from the bottom lines to the cameras. This shows that the system does a good job estimating the z-position.

Looking at the blue lines in Figure 3.12 (a) they look way off. One should expect a dot, as the view is along the line. The explanation to this is shown in Figure 3.13. The system finds the lines where planes meet, and planes are

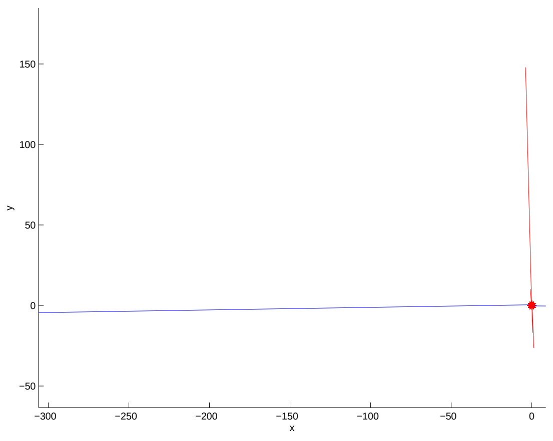


Figure 3.13: Entire view of the box

a infinite size. This means that the reconstructed edges can be of various lengths. The edges can get quite long compared to the edges on the box.

Looking along those lines, only a small error can give the wrong impression. If the lines are trimmed this effect would be mended.

At last the xz-plane is shown in Figure 3.14. This figure shows the two top edges close to the same z-coordinates as the other top lines in Figure 3.12. Now the red lines are far off.

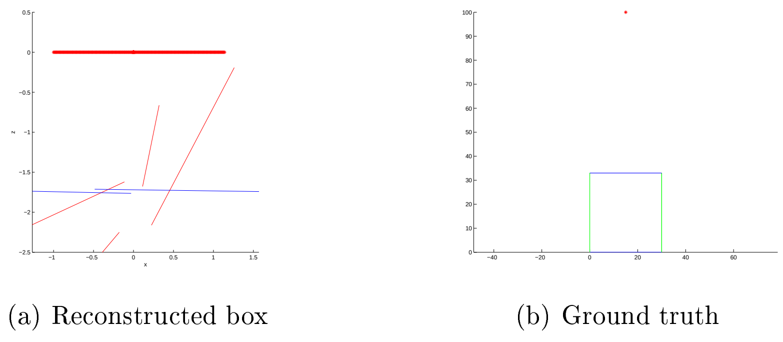

(a) Reconstructed box (b) Ground truth

Figure 3.14: zx-plane

3.8 Reconstruction using parallel lines to the camera movement

Until now, edges that are perpendicular to the movement of the camera has been used for 3D reconstruction. Is there a reason for this? The answer is yes.

Going back to Figure 2.18 in Section 2.8 one can see that the translation of the camera is perpendicular to the edge. If the translation is parallel the planes would all lie together, giving no information about the position of the edge. This can be compared to looking at a line, and moving perpendicular to the line, one can see and register the movement well. Moving parallel to the line, one can not see the movement (giving a line without any texture).

Reconstructing two of the edges from the experiment in the previous section is tried out. The results are shown in Figure 3.15.

The reconstruction of the two lines parallel to the camera movement fails miserably. Obvious that the system detects no translation of the edge.

3.9 Noise in data

In this section the influence of noise in the data is checked. An edge from Dataset 2 is tracked through 73 images and is plotted in Figure 3.16. Looking

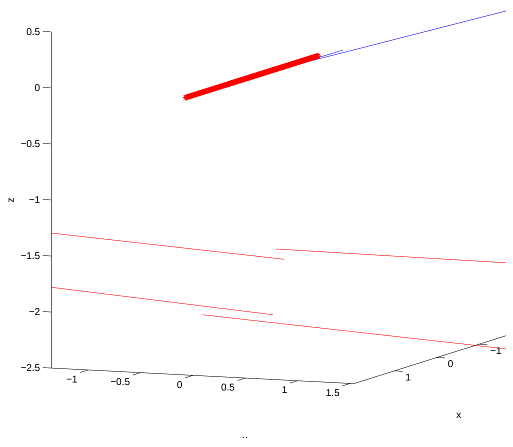


Figure 3.15: Reconstructing edges using parallel camera movement

at the figure the noise in the data is obvious.

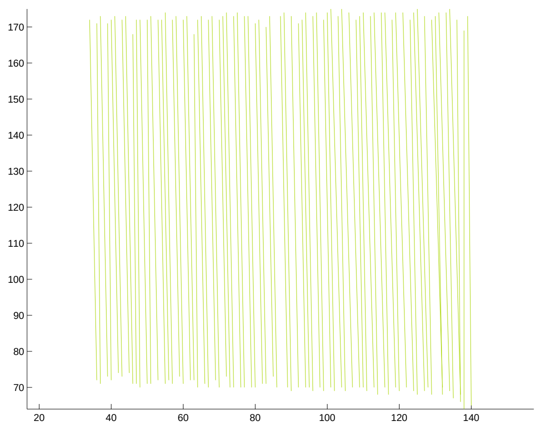


Figure 3.16: An edge tracked through a part of an image sequence. The noise is obvious

Another data is created to simulate the one in Figure 3.16. Edges are created at the same start- and endpoints, but linearly distributed in between. This is shown in Figure 3.17.

As in the previous section, the edges are reconstructed and plotted in Figure 3.18.

One can clearly see which edge the data was simulated from. The simulated line is vertical as oppose to the one reconstructed from real data.

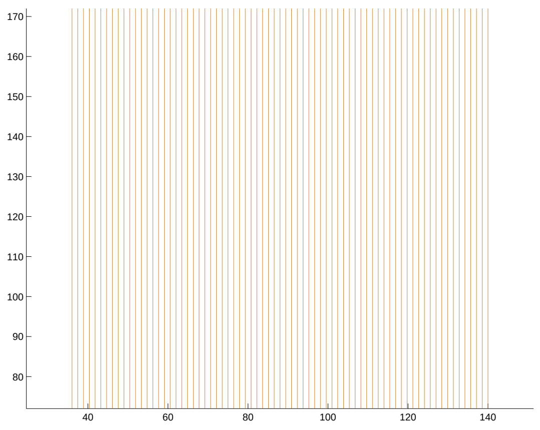


Figure 3.17: Synthetic data to simulate the one shown in Figure 3.16

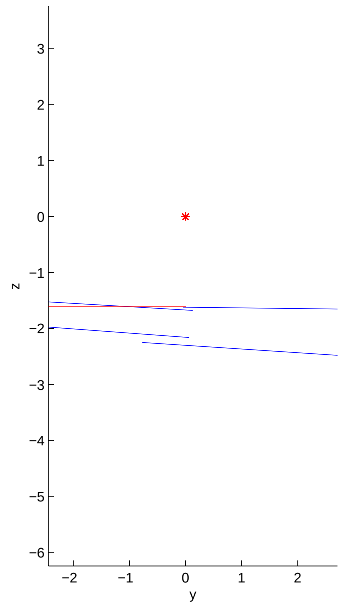


Figure 3.18: Edges reconstructed. Blue lines are edges tracked from the image sequence. Red line is synthetic data made to simulate one of the edges

The xz view is plotted in Figure 3.19. The red line in this figure appears as a dot. This shows that the line is perfectly straight in the y-direction. This shows that the application works good with noiseless data.

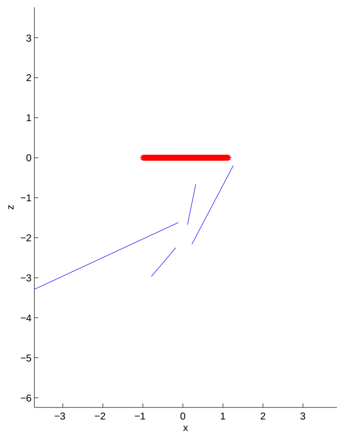


Figure 3.19: The red line appears as a dot in this view

Chapter 4

Discussion and Future Work

This chapter is a general discussion about the system and problems met during implementation. These problems can be dealt with in future project, to further develop the system.

One possible source of error is the uneven movement of the robot arm. This problem could be mended physically by fixing the arm, or by software implementation. The positions of the cameras are just a guess in the current system, but could be used as a start-guess in a structure and motion problem, where the position of the cameras are optimized by iteration.

The camera used in this project is a `340x240` pixel camera. A translation between images is small in millimeters, but it is almost 1.5 pixels. This can result in error when determining the difference in small movements.

The execution time is always an issue in a system like this. In the current system the execution time is the highest in the tracking part. A conversion to `C` and linking it to `Matlab` via `Mex` could be a solution to this problem. Converting the system completely to `C` would make it faster, but as a developing tool `Matlab` is much better choice and therefor used in this project.

More automation of the robot could be done. That is the robot takes one sweep over the object and the resulting sequence is processed. If the system needs some more information about a part of the object the arm moves there and another sequence is made. This could be repeated several times.

One way to improve the system could be to remove outlier plane(s) in the 3D reconstruction part. This would improve the determination of the 3D position of the reconstructed line.

As information about the translation of a line is best captured perpendicularly the robot arm has to move both in the x- and y-directions. To be able to retrieve information about the vertical edges, the robot should be able to rotate the camera, and move it perpendicular to those lines. This would mean that the equations in Section 2.9 containing the rotation matrix

R would have to be recalculated.

To be able to visualize the reconstruction better, trimming the reconstructed lines is necessary. In the reconstruction planes are used, which have infinite size, so reconstructed lines can get very long.

Chapter 5

Conclusion

The objective of this thesis were

- To propose and select a suitable feature detection
- To implement a feature tracker
- To implement a 3D reconstruction algorithm
- To test the implementation

These have been met to a large extent.

Three methods for detecting features in an image are proposed, tested and compared. These are corner detection, edge detection and optical flow. These methods were tried out on actual data taken from the robot. Edge detection was found to be the most suitable method. The problem with corner detection is that corner can be detected on mis-colorations on surfaces and does not necessary describe the structure of the object. Optical flow worked fine when the translation between images was small, but as the translation increased the method failed to detect the movement. Edges are the feature that describe an object well, and are easy to detect and track.

An edge tracker is implemented. This edge tracker is able to track edges through a whole sequence of images.

A 3D reconstruction algorithm is implemented. The edges detected by the edge detector and tracked by the edge tracker are used to reconstruct the object in 3D. Here it becomes essential that the features used describe the object well.

Tests are done to test the functionality and limitation of the system. Most of the tests are made on the 3D reconstruction part. One is made on the edge detection part. Here it is found that three constraint parameters have great influence of how detailed the detection is. In this system it is essential that the system finds long edges, as the detection is able to detect and segment the smallest edges.

The translation of the camera is measured in pixels and millimeters. Using those information actual measurements can be made on objects.

The error is measured for different lengths of the baseline, and found to be in good coherence with the rule in the literature that the ratio between baseline and distance to object should be between $\frac{1}{3}$ and 3. After the baseline is shorter that $\frac{1}{3}$ the error increases.

The implementation was used to reconstruct a box. The reconstruction proved to be a bit erroneous. The edges do not lie vertically in the z direction. But the distance between the lines and camera proved to be right, which is important.

A test was done to show the difference between tracking lines which are perpendicular to the movement of the camera versus lines parallel to the camera. The test showed that lines parallel to the camera can not be reconstructed. This means that if vertical lines on the object is to be reconstructed the robot has to be able to move the camera perpendicular to those.

Noise in data proved to have a huge effect of the quality of the reconstruction. An edge was tracked through number of images where data was obviously noisy. The data was simulated, with no noise and reconstructed. The reconstruction proved to be better.

The results for the system implemented in this thesis show that the reconstruction algorithm and implementation work, but there is still work to do on the input data. This can be thought of as a step toward a fully automatic quality control system.

Bibliography

[1] Berthold, W. & Bisiach, B. (1998). An innovative fully automatic facility in welding process: the erection state in the railway construction industry, where auxiliary operations are carried out automatically and a relevant improvement is assured to the working environment.

[2] Wolf, K. & Roller, D. & Schäfer, D. (2000). An approach to computer-aided quality control based on 3D coordinate metrology. Journal of Materials Processing Technology 107, 96-110.

[3] Shafiq, M. S. & Tümer, T. & Güler, H. C. (2001). Marker detection and trajectory generation algorithms for a multicamera based gait analysis system. Mechatronics 11, 409-437.

[4] Jens Michael Carstensen (Ed.) (2002). Image analysis, vision and computer graphics. IMM, Technical University of Denmark.

[5] Harris, C. & Stephens, M. (1988). A combined corner and edge detector. Plessey Research Roke Manor, United Kingdom.

[6] Hartley, R. & Zisserman, A. (2000). *Multiple View Geometry*. CAMBRIDGE: Cambridge University Press.

[7] Aanæs, H. (2003). An Introduction to Multiple view Geometry.

[8] Black, M. J. and Anandan, P., The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields, Computer Vision and Image Understanding, CVIU, 63(1), pp. 75-104, Jan. 1996.

[9] Hardarson, Gunnar (2005). XYZ-Table.

Appendix A

Code

Figure A.1 shows a sketch of the structure of the code. All files and functions

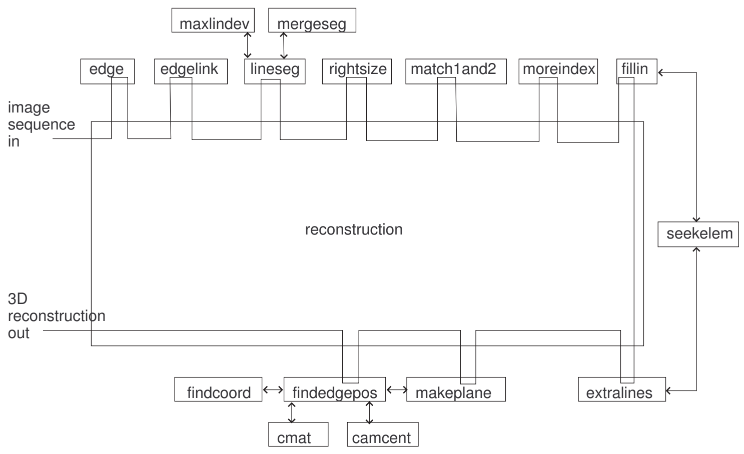


Figure A.1: Illustration of the code

are listed in Table A.1. All code is written in `Matlab`. The code and datasets can be found on the accompanying CD.

File name	Function
edge	Takes an image as input and finds its edges
edgelink	Takes edges as input and links them together
lineseg	Takes edges as input and returns the endpoints
maxlindev	Used by lineseg. Finds the point of maximum deviation from a line
mergeseg	Used by lineseg. Seeks to merge edges together given constraints
rightsize	Takes two lists of edges and makes them the same size so
	they can be put together
match1and2	Matches two first images in a sequence
moreindex	Matches the lines in the first image to the rest of the images
fillin	Fills in edges where they have occluded
seekelem	Used by fillin and extralines. Seeks an edge in a list of edges
extralines	Matches lines that appear after the first image in a sequence
findedgepos	Finds position of an edge in space
cmat	Used by findedgepos. Computes the camera matrix
camcent	Used by findedgepos. Finds the center of the camera
findcoord	Used by findedgepos. Finds in world coordinates the
	position of edge endpoints
makeplane	Used by findedgepos. Makes a plane given three points in space

Table A.1: Files used in this project and function