Infering the 3D spin of a moving object using the Microsoft Kinect camera

Daniel Kestell
B.A.I Computer and Electronic Engineering
Final Year Project, 2014
Supervisor: Dr. Rozenn Dahyot
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

_________________________ March 31, 2014
student’s name
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

_____________________________ March 31, 2014
student’s name
Acknowledgements

Many thanks to my Supervisor Dr. Rozenn Dahyot for all her help and guidance throughout this project, I greatly appreciate it. I would also like to thank my parents for their support throughout this project and my time in University. Thank you to the School of Computer Science for providing the Kinect for this project.
## Contents

1 Introduction ........................................... 1

2 State of the Art ................................. 3
  2.1 Object Tracking ................................. 3
  2.2 Segmentation with Depth ................. 4
  2.3 Least Squares and M-estimators ........ 4
    2.3.1 Least Squares .......................... 4
    2.3.2 M-estimators ........................... 5
    2.3.3 Least Squares and Motion Estimation in Literature 8
  2.4 Feature Based Techniques .................. 9
  2.5 Pose Estimation .............................. 9
  2.6 Kinect ........................................ 10
  2.7 Conclusion .................................. 11

3 Methodology ...................................... 12
  3.1 Pipeline ..................................... 12
  3.2 Reading from the Kinect .................... 12
  3.3 Ball Segmentation ......................... 15
    3.3.1 Colour Segmentation ................. 15
    3.3.2 Depth to Colour Mapping .......... 15
    3.3.3 Depth Segmentation ............... 17
  3.4 Features ................................... 22
    3.4.1 Feature Detection .................. 22
    3.4.2 Feature Matching ................... 23
  3.5 Converting Image Features to Real World Coordinates 24
  3.6 Computing the Rotation and Translations 25
  3.7 3D Graphics ................................ 28

4 Experimental Results ......................... 30
  4.1 Ideal Case ................................ 30
    4.1.1 2D case ................................ 30
    4.1.2 3D case ................................ 35
  4.2 Real Data - Very Slow Movement ........ 39
  4.3 Real Data - Full Movement ............... 40

5 Conclusion ...................................... 47
## List of Figures

1.1 Lane ................................................................. 2

2.1 Least Squares Line approximation ................................. 5
2.2 Least Squares approximation with a Polynomial .................. 5
2.3 Least Squares is a Quadratic ..................................... 6
2.4 Least Squares Polynomial approximation when there is an outlier . 6
2.5 Comparison of different criteria for optimisation .................. 7
2.6 M-estimators used on the data with an outlier .................... 7

3.1 The Microsoft Kinect ............................................... 13
3.2 Depth (left) and RGB (right) data from the Kinect in Kinect Studio . 13
3.3 Two Frames one after the other at a Higher Frame rate ............ 14
3.4 Two Frames one after the other at a Lower Frame rate ............ 14
3.5 Ball Segmented By Colour Subtraction .......................... 15
3.6 Binary Mask when there is shadow ................................ 16
3.7 Depth Image (left), Depth Image with mask(right) ............... 16
3.8 Depth and RGB image mapped for image with ball (left) and background (right) 17
3.9 The difference between the Depths for the Background Image and the Foreground Image ........................................ 18
3.10 Difference Image zoomed ......................................... 18
3.11 Binary mask created from depth difference ....................... 19
3.12 RGB image masked based on depth ................................ 19
3.13 Current Frame and Background Masked With Depth ............... 20
3.14 Difference Image, using Colour ................................... 20
3.15 Mask created using Colour segmentation ........................ 21
3.16 RGB image masked with Colour segmentation mask ............... 21
3.17 Circular mask for the ball ....................................... 22
3.18 RGB image with Circular mask applied .......................... 22
3.19 SIFT features .................................................... 23
3.20 SIFT matches between frames at time t (left) and time t+1 (right) 23
3.21 Converting Image Coordinates to Real World Coordinates ........ 24
3.22 Correspondences in Image and Real World Space ................ 25
3.23 Points at time t in mm (first three columns) and points at time t+1 in mm (last three columns) .................................. 27
3.24 Example variable cells with 3 angles and 3 translations ........ 28
3.25 Estimated Values of $x_{t+1}$, $y_{t+1}$ and $z_{t+1}$ ...................... 28
3.26 The error between the point at time t and t+1 (three left most columns), M-estimators for x,y and z (column 4,5 and 6), Sum of M-Estimators (column 7) ................................................................. 29
3.27 3D Graphics Implementation ................................................................. 29

4.1 Ideal 2D input data ..................................................................................... 31
4.2 Ideal 2D with no outliers ........................................................................... 31
4.3 Ideal 2D with outlier .................................................................................. 32
4.4 Ideal 2D with one close outlier using Least Squares ................................. 32
4.5 Ideal 2D with one far outlier using Least Squares .................................... 33
4.6 Ideal 2D with one outlier using M-estimators ........................................... 34
4.7 Ideal 2D Comparison of Least Squares and M-estimators for one close outlier 34
4.8 Ideal 2D Comparison of Least Squares and M-estimators for one far outlier 35
4.9 Ideal 2D Multiple Far Outliers Comparison ............................................ 35
4.10 Ideal Rotation about the X-axis. ............................................................... 36
4.11 Ideal Rotation about the Y-axis. ............................................................... 37
4.12 Ideal Rotation about the Z-axis. ............................................................... 37
4.13 Outlier Effect 3D Least Squares .............................................................. 38
4.14 Outlier Effect 3D M-estimators ............................................................... 38
4.15 Outlier Effect 3D M-estimator, different angle ......................................... 39
4.16 Real Data points ....................................................................................... 40
4.17 RGB image from Kinect .......................................................................... 41
4.18 Slower moving ball ................................................................................... 41
4.19 Angle of Rotation $\alpha$ between frames about the X-axis ....................... 42
4.20 Angle of Rotation $\beta$ between frames about the Y-axis ......................... 42
4.21 Angle of Rotation $\gamma$ between frames about the Z-axis ....................... 43
4.22 Translation $t_x$ between frames in the X-direction ................................ 43
4.23 Translation $t_y$ between frames in the Y-direction ................................ 44
4.24 Translation $t_z$ between frames in the Z-direction ................................ 44
4.25 Sum of the translations in the X-direction .............................................. 45
4.26 Sum of the translations in the Z-direction .............................................. 45
4.27 Norm Vector of $t_x$ and $t_z$ .................................................................. 46

B.1 Angle of Rotation $\alpha$ between frames about the X-axis ....................... 53
B.2 Angle of Rotation $\beta$ between frames about the Y-axis ......................... 54
B.3 Angle of Rotation $\gamma$ between frames about the Z-axis ....................... 54
B.4 Translation $t_x$ between frames in the X-direction ................................ 55
B.5 Translation $t_y$ between frames in the Y-direction ................................ 55
B.6 Translation $t_z$ between frames in the Z-direction ................................ 56
B.7 Sum of the translations in the X-direction .............................................. 56
B.8 Sum of the translations in the Z-direction .............................................. 57

C.1 Example of the Excel Spreadsheet used ................................................... 59

D.1 example of Kegal Oil pattern ................................................................... 61
Abstract

The aim of this project is to infer the spin of a 3D moving object, specifically a sphere, using a 3D depth camera. The camera that has been used is the Microsoft Kinect. This report comprises of a review of other work done in the areas of Object tracking, Pose Detection, Depth Segmentation and Optimisation in chapter 2. Chapter 3 contains the methodology and what was done. This section shows how the Depth segmentation system and the feature tracking works and explains how the Rotations and Translations, the movement parameters, are found. Chapter 4 contains experimental results for ideal and real data. A method for computing the spin of a spherical object was created for this project as a proof of concept. This concept has been proven in this report and is a promising start to an interesting idea.
Chapter 1

Introduction

Ten pin bowling is a sport enjoyed by millions of people around the world, some for leisure and some for sport. Those that bowl for leisure often don’t understand the complexities that are actually in a sport bowling game and will dismiss bowling as a game where the average person can throw a ball, knock down a few pins and drink alcoholic beverages while doing so. That is fine, if that is what one plays bowling for, just as one can have fun playing soccer with their friends in the local park. Being good at beating your friends in a casual game of football does not mean that you would be even close to the capability of players in the Premier League. Just as this is true, it is also true that professional bowlers are at a completely different level that in some ways they may as well be playing a different game.

Sport bowling, whether it be at the top level of professional or even just a bowling alley league requires more than just throwing the ball down the lane and hoping for the best. It requires the ability to be accurate and consistent but also to understand the lane conditions. Lane conditions are the oil pattern that has been put down onto the lane and an example of a Kegel lane pattern is shown in appendix D from [13]. Some areas will have different amounts of oil and this will affect how the ball moves. It is therefore essential for a bowler to understand the lane conditions if they are to achieve top scores. A bowler understands these conditions by observing how the ball rolls on the lane. This is done purely by eye as it is.

The aim of this project is to use the Microsoft Kinect to analyse the movement of the ball and find how the ball is spinning. This would be very useful not only to the bowler who is trying to understand the lane conditions but also those who are attempting to refine their shot and would like to know exactly how their ball is moving so that they can change elements of their shot to improve their performance.

This analysis has been done using the Kinect in order to use its Depth camera as well as its RGB camera. A vision based technique using a single camera was desirable over multiple cameras or laser scanners as it would allow the setup to be compact and easily portable, which is desirable for the average user. The Kinect is also a relatively inexpensive apparatus at only €100 to €200 and within the reach of many consumers. In fact, if one already had a Kinect for an Xbox they could use it.

As it is, the project is a proof of concept and is not in a product ready state. In this project different techniques were investigated and a set of Offline techniques were chosen. It was considered wise to prove the concept using offline techniques as it was decided that the adding the extra difficulties of an online setup would make things more complicated than they need to be at this stage.
CHAPTER 1. INTRODUCTION

To test the idea a small scale lane was set up as shown in figure 1.1

![Lane](image)

Figure 1.1: Lane

The ball is rolled down this lane and the Colour and Depth images are captured by the Kinect. Using this data the ball is segmented from the image and features are tracked between frames. These features are then compared between one frame and the next and used to compute the rotation between frames. Investigated for this project were methods such as SIFT and SURF for feature matching and Least Squares and M-estimators for optimisation as well as different methods for tracking, pose estimation, depth segmentation and optimisation.
Chapter 2

State of the Art

This chapter contains a literary review and a discussion of the state of the art aspects of the project.

2.1 Object Tracking

For this project it was necessary to be able to track the ball in the scene. There are many examples in literature of object tracking techniques. Many of these techniques are used to track humans for video surveillance systems. Video surveillance is a common application of Computer Vision. Wang et al [19] used Boosted Colour Distribution Method to track objects by their colour. They proposed three contributions to this, the first was to create a novel Online Gentle Boost algorithm (OGB) for online learning, they designed a new weak classifier called the Log Colour Feature Distribution Ratio (LCFDR) to provide a simple but effective way to find colour features and they combined OGB with the new LCFDR classifier to create a new colour based object tracking technique.

Another technique which is used to track objects, in this case human faces, was proposed by Zhang et al [23]. It is called Online Discriminative Feature Selection and is a classifier based method for robustly and quickly tracking faces. Their proposed algorithm takes a frame at time t+1, crops some patches of the image based on the tracking location at time t, applies the classifier and find the area of maximum confidence, which is the new tracking result. Once this result has been calculated new cropped patches of the image are taken and based on their distance from the tracking location they are classified as either positive or negative, whether they contain the face or not. These results are used to update the classifier and the next frame is brought in. Their algorithm optimises the objective function of the classifier using steepest ascent for positive samples and steepest descent for negative samples.

Yoon et al [22] used a fuzzy particle filter for tracking cars on a road. According to [22] particle filters are commonly used to track objects in real time. They proposed using an adaptive autoregressive model as a model for state transitions and the adaptive appearance mixture model as an observational model. They were using adaptive models in order to gain robustness for their tracking but they found that adaptive models are affected greatly by noise. In order to overcome the noise problem they proposed a fuzzy particle filter which used fuzzy variables using probability and possibility theory.
2.2 Segmentation with Depth

The Kinect provides the functionality to find the depth of points in an image. This allows it to be used to segment the foreground and the background of an image more effectively than using the RGB camera alone. Hernandez-Lopez et al. [9] proposed using the depth camera on the Kinect to segment objects in a scene even if there are many objects of the same colour present. An RGB camera would be ineffective at segmenting just one of these objects from the scene, however using the depth information provided by the Kinect the objects can be segmented more effectively. Their algorithm segments the image by colour and by depth and then fuses them together. From the RGB images, an Homography matrix is used to relate the colour and depth information and obtain 'geometrical adequation', i.e. match the colour and depth images together. The colour space of the image is then changed into the CIE-Lab colour space using a matrix which when multiplied by the RGB vector, for each pixel, converts the image into the CIE-Lab colour space. The Euclidean distance for the colour components of the image is used as an initial mask for the segmentation. Once this procedure is done a probabilistic image allows the system to identify all objects of the same colour. Finally the initial mask and the depth information are combined in order to segment a particular object that is at a desired depth from the camera and discarding those that appear at different depths. Statistical analysis is done to determine which points in the depth image belong to a certain object.

Fernandez-Sanchez et al. [7] proposed using a Depth Extended Version of the Codebook scheme proposed by Kim et al for Background subtraction. The Codebook algorithm constructs a background model based on a quantization or clustering method. The background model is made up of a codebook which contains one or more code words contain information about colour, brightness, frequency of use etc. The Codebook algorithm is a learning algorithm.

It was decided that in the case of this project a less complex method could be used for depth segmentation because the scene, apart from the ball, can be assumed to be static. This means that simple subtraction of the depth image containing the ball from the image without the ball can be used to segment the ball with depth.

2.3 Least Squares and M-estimators

2.3.1 Least Squares

Least squares is a common technique for computing optimal values in a system and is often mentioned in literature. It is commonly used as a method of fitting a line to a series of data and an example is show in figure 2.1.

It can be seen that Least Squares has been used to fit a line to the blue dataset. Least squares can be used to fit potentially any equation to a dataset, for example a cubic polynomial could be fitted to this data by changing four parameters, i.e. its coefficients. In this case the cubic polynomial looks like a line as a line is the best fit for this data so the $x^3$ and $x^2$ terms have near 0 coefficients, so it is effectively the equation of a line. It is however necessary to have at least four points in order to compute the optimum solution for a cubic polynomial as there are four coefficients in a cubic polynomial. The data approximated by a least squares line is shown in figure 2.2.

One of the weaknesses of the Least Squares method is its sensitivity to outliers. It is
CHAPTER 2. STATE OF THE ART

Figure 2.1: Least Squares Line approximation

Figure 2.2: Least Squares approximation with a Polynomial

sensitive to these outliers because the criteria to minimise is a quadratic which as its input increases, its output increases to the square of that value, i.e. $y = x^2$. The figure 2.3 shows that as the error between the points and their estimation by the chosen equation increases, the error squared, which is the criteria to minimise in Least Squares, continues to increase. This means that any outliers skew the result and the further the outlier the more it skews the result.

The effect of an outlier on the cubic polynomial case can be seen in figure 2.4. This is the only major outlier and yet it has greatly changed the polynomial which is fit to the data.

This method clearly has a flaw in the presence of outliers. As the nature of this project suggests that there will be outliers in the data it is clear that another method is needed to improve the resistance to outliers.

2.3.2 M-estimators

M-estimators is another method of finding the optimum solution to a set of equations and is considered better than Least Squares according to [3]. Collet et al [1] mention that when they have computed correspondences that there will inevitably be mismatches and say that they need to use robust techniques such as M-estimators in order to reduce the effect of these
mismatches. In their paper, on the MOPED framework, they use an algorithm called ICE (Iterative Clustering Estimation) in order to both solve the correspondence and pose estimation problems. They also use RANSAC which is another algorithm for robust estimation. M-estimators is very similar to Least Squares except that instead of computing the sum of the squared error the following formula is used instead:

$$\sum \frac{\epsilon^2}{1 + \epsilon^2}$$

(2.1)

Where $\epsilon$ is the error between the actual point and the estimated point using the formula that is approximating the data, i.e. the polynomial shown earlier. Using the formula shown above (Geman and McClure) instead of the least squares formula can be seen to reduce the effect of outliers. Shown in figure 2.5 is a comparison of 4 different functions that could be used to find an optimum solution, including Least squares, which was seen in [3]. It can be seen that Geman and McClure is the most effective at limiting the effect of points with large error, i.e. outliers.
CHAPTER 2. STATE OF THE ART

Figure 2.5: Comparison of different criteria for optimisation

Using GM caps the effect of an outlier, i.e. the result of the formula is capped to 1 for a large values of error. This means that the greater the outlier the less effect it has on the inlier data, as shown in figure 2.6

Figure 2.6: M-estimators used on the data with an outlier

It is clear that the outlier is now having little effect on the estimation of the data. If this data set were modelling a real system it is clear that the point whose y-value is 100 is an outlier and doesn’t match the rest of the pattern. In a Electrical Engineering system, for instance, this could signify some noise in the measurement at that point. It is not desired that such an obvious error have a large effect on the estimation of the data by a model so M-estimators is a useful technique to reduce the effect of these types of outliers.
2.3.3 Least Squares and Motion Estimation in Literature

Eggart et al compared four different major algorithms for estimating 3-D rigid body transformations in their 1997 paper. The algorithms that they compared were the Arun et al. (1987) algorithm which is based on computing the Singular value decomposition of a matrix derived from the standard Rotation and Translation Representation, the Horn et al. (1988) algorithm which was based on the exploitation of the orthonormal properties of the rotation matrix and computes the eigensystem of a different derived matrix, the second algorithm by Horn (1987) which is based on computing the eigensystem of a matrix and representing the rotational component as a unit quaternion and Walker et al. presented a fourth technique in 1991 which analysed an eigensystem yet again but this time the rotation and translation matrices were represented by a dual quaternion.

In all of the algorithms the data is a set of corresponding point sets \( \{m_i\} \) and \( \{d_i\} \) with \( i = 1..N \), in the case of this project they would be the returned SIFT feature matches. The relationship between the two data sets is as follows:

\[
d_i = Rm_i + T + V_i
\]

Where \( R \) is a 3×3 Rotation Matrix, \( T \) is a 3×1 translation vector and \( V_i \) is a noise vector. This is the same equation as those shown above, \( d_i \) is a point at time \( t+1 \) and \( m_i \) is a point at time \( t \).

Each of these algorithms attempts to minimise the Least Squares Error Criterion which is shown below.

\[
\epsilon^2 = \sum_{i=1}^{n} \left\| d_i - (\hat{R}m_i + \hat{T}) \right\|^2
\]

Eggart et al [5] reported that there was very little difference in the performance of these four algorithms other than a few speed enhancements and minor stability differences. They noted that an earlier paper by Walker et al claimed that their algorithm (DQ) provided superior accuracy but that this claim was not substantiated in their experimental data.

The algorithms that were investigated by Eggart et al are closed form algorithms. Another type of algorithm that can be used for estimating a solution is called an iterative estimation process. According to Hersch et al [10], iterative algorithms can be more effective than closed-form solutions for applications such as object tracking. In their algorithm, parameters \( b \) and \( t \), which represent the rotation and translation vector respectively are iteratively updated. The iterative estimation process begins with an initial estimate of \( T \), the transform, and picks a random pair of points \((x_i, y_i)\) and performs a gradient descent step on the corresponding residual \( \left\| y_i - T(x_i) \right\|^2 \). This means that if the transformation \( T(x) \) which can be represented as \( T(x) = R_b(x) + t \) can be parameterised by \( t \) and \( b \) then it can be updated iteratively by computing its gradient and using a learning step. In this paper they mentioned that iterative solutions are good for situations that require the estimation of many similar transformations, such as those in this project. They also mention that iterative algorithms can take advantage of the estimations of the previous steps as the initial estimations for the current step.

Microsoft Excel can be used to run the iterations necessary for Least Squares. It optimises a “trial and error” method, which would take a significant amount of time as every single value is checked, by analysing the observed outputs and their rates of change and uses these as a guide to selecting the trial values for the next iteration. The derivatives and gradients
provide indications on how the “adjustable cells”, i.e. the coefficients in the polynomial example shown above. The Solver decides which parameters need to be changed more by computing the partial derivative of the optimum/criterion cell, in the case of least squares this cell is the sum of the squared errors, with respect to the adjustable cells. If this partial derivative is a large positive it will be adjusted in one direction, if it is negative it will be adjusted in the other. If the partial derivative is close to zero then it will be altered less. This means that each iteration converges on a solution.

2.4 Feature Based Techniques

In order to track correspondences between frames it was decided that a feature based technique would be used to track different portions of the ball between frames. A Feature based technique was chosen instead of a boosted classifier technique because it does not require a learning algorithm. In bowling many different types of bowling ball are used, with different colours and patterns abound and in order for this system to be able to work without requiring a large database of images of the ball it was decided that feature matching was the best solution. The next step is to choose the best feature matching technique. For this project the Scale Invariant Feature Transform (SIFT) was chosen to provide this functionality.

Carvalho et al [2] proposed a new paradigm for assessing appearence models and their corresponding description techniques. Their technique was used to analyse these techniques, including SIFT, SURF (Speeded Up Robust Features) and FIRST (Fast Invariant to Rotation and Scale Feature Transform), in relevance to a tracking environment. They mentioned that independent testing of FIRST suggested that it was the fastest but that using their technique they found that the results were not as good due to the interdependecies of the modules that were not taken into account in other tests. They said that the interdependencies meant that more iterations in the matching step were necessary as there was increased uncertainty.

They noted that SURF was considerably faster than SIFT but that it was more sensitive to noise and that SIFT had competitive accuracy but had a high computational weight. Another method that was discussed which was considered faster than SIFT was HOG(Histogram of Oriented Gradients). It was mentioned in their literature review that HOG has a high dimensionality, so requires a large amount of memory. It was also mentioned that HOG is weak when it comes to representaions of objects with large smooth regions and that it is highly sensitive to rotation transformations, which are the transformations that are desired in this project.

SIFT and SURF were decided to be the main candidates for the feature matching algorithm that would be used as they both have implementations in OpenCV, however it was found that SIFT provided better correspondences. Despite the fact that SURF is a faster algorithm than SIFT, it has a poor implementation in OpenCV, according to Dawson-Howe [4], and is much slower than it should be.

2.5 Pose Estimation

Pose Estimation was investigated as it may have been useful for finding the rotations between frames, if the pose was known for each frame, given that the camera remains fixed. There are many papers on pose estimation but most of them were found to be based on the pose
estimation of planar objects such as boxes and the pose estimation was not used for moving objects.

For instance, Yang et al [21] proposed a method for pose estimation which used SIFT features to match correspondences between images and a database and they verified the result using the homography constraint. They used RANSAC to find the best Homography Matrix and used it to estimate the Pose. They computed a $\alpha$, $\beta$ and $\gamma$ rotation angles. Yang et al confirmed that SIFT features performed better for them than SURF features.

Ekvall et al [6] recognised and estimated the pose of objects using colour co-occurrence histograms. A colour histogram of an object is a compact representation of its appearance. They use these histograms to decide firstly if and where the desired object is in the scene and create a vote matrix indicating the likelihood that particular windows, within the image, contains the object in question. The rotation is also estimated by comparing the histograms of the object in known training images of different poses and the histogram of the object in the current image, once it has been found. The hypotheses are weighted by a Gaussian to find the most likely pose. This idea may not work for a bowling ball, as bowling balls come in many different colours and patterns which would need to be trained into the system.

Li-Juan et al [14] used Line Correspondences to estimate the Pose of an object. They mentioned that the accuracy of Line Detection is high and that Lines are better at dealing with occluding and ambiguous situations. They also mentioned that lines are easily expressible in mathematics and work efficiently and that their technique did not require the optimisation of complex non-linear equations. The main problem with this method, for this project, is that a spherical object such as a ball does not have straight lines in it and a line that seems straight in a 2D image is actually curved.

Penman et al [17] proposed a method for estimating the pose of 3D symmetrical objects of unknown shape. They proposed that, knowing that an object was symmetrical allowed them to find its pose from a single image of the object, as long as there was symmetry in the object in the current view. The method can only deal with Rotation in one direction and fails when there is rotation in both the X and Y axes as the symmetry is not preserved. This method is not useful for the problem of this project as a spherical object is symmetrical in all axes and the symmetry is therefore not a good method for finding the pose of a spherical object.

Another technique for Pose Estimation was used by Shang et al [18]. They used 3D range finders to help them identify the object and its pose. They align 3D point clouds with known 3D models that they have stored in a database. They compute error surfaces, a process which involves a formula very similar to Least squares. They observed that each unique view of an object will result in a different error surface with respect to a model in a fixed pose. This is then used to classify the object and can be used to compute the pose. The problem with this method for the current problem of the moving ball is that the ball will appear nearly identical in every view in terms of the depth image alone.

2.6 Kinect

The Kinect is a useful enhancement to the situation as its depth camera provides data about 3D coordinates instead of just 2D coordinates. As already stated, this can be used to make the segmentation problem easier, as in [9] and [7].

Gonzalez-Jorge et al [11] stated that the Kinect was a useful sensor for many engineering applications. They did state that the precision decreases with range but for the range of about
1-2 m the range precision is quite good. They stated that the values of precision for a 1m range were 1 to 6mm and those for a range of 2m were 4 to 14mm. They found that the Kinect is only useful in an indoor environment as the high radiance of sunlight can saturate the sensors. They found that both the Microsoft Kinect and Asus Xtion camera have similar accuracies which they state is probably due to the fact that they both use the same PrimaSense infrared measurement unit.

The range is adequate for the small scale application but may not be adequate for the full scale bowling alley. The Kinect has been used to prove that the 3D spin can be inferred using a 3D camera and would be upgraded for the full scale application. The problem of the high radiance of sunlight is not important in this project as its application would be an indoor one.

2.7 Conclusion

There are clearly many methods to perform pose estimation and object tracking already investigated. However, the problem of computing the spin of a moving object is not one that has been widely researched.

Some of these methods were applied to this problem. It was decided that a feature based technique would be used in order to find the Rotations of the spherical object because a learning technique requires a database of images of the object and, given that there are many different types of bowling balls on the market, it was decided that a method that does not require this database was preferable. It was decided that features would be tracked between frames and their 3D positions, using the Kinect, could be found.

SIFT was decided as the feature type of choice as it was considered by papers such as Yang et al [21] and Carvalho et al [2] to be one of the most robust of the feature techniques. It is considered quite a slow technique but given that this project could be undertaken in an offline capacity and that the OpenCV implementation of SURF is a slow one it was decided that SIFT was the best choice.

M-estimators was chosen in order to estimate the Rotations because outliers were expected in the data due to mismatched correspondences, which would largely affect the estimation of the Rotations, if Least Squares were used. M-estimators is considered a robust estimation technique which is easy to implement once Least Squares has been implemented.

It was decided that for this project the tracking element is easier than it was for Wang et al [19] and Zhang et al [23] as the background is fixed with no extra feature rich areas. This means that background subtraction can be used to segment the ball and therefore be used for tracking easily. The background in a bowling alley is feature rich but is also static with very little lighting change. This means that these assumptions for the scaled down lane are valid.
Chapter 3

Methodology

The Project is split up into 3 parts. The C++ code to read in the RGB and Depth data from the Kinect, the C++ code to Segment the ball from the image and match the features between frames and the Excel spreadsheet in which the Rotations and Translations are computed from the correspondences.

It was decided that the project would be undertaken in an offline manner as the project is a proof of concept. This means that each of these sections are kept separate and are joined manually. Excel was used because of a lack of time as it has a built in Solver Add-in for optimisation. If the full online version were implemented it would have to be replaced by a C++ equivalent.

3.1 Pipeline

In order to compute the rotations and translations of the ball between frames it was decided to use the following series of steps:

1. Ball Segmentation (OpenCV, C++)
2. Feature Detection (OpenCV, C++)
3. Feature Matching (OpenCV, C++)
4. Conversion from Pixel to Real world coordinates (C++)
5. Rotation and Translation Estimation using Matched Features (Excel)

3.2 Reading from the Kinect

The first thing that needs to be done before these steps can be implemented is to read data from the Kinect. The Kinect provides Colour and Depth information using an RGB camera and an IR emitter and sensor, as shown in figure 3.1.

Figure 3.2 shows the Depth and RGB images that the Kinect records. These images are shown here in the Kinect Studio application provided with the Kinect SDK. The Kinect Studio allows one to record the RGB and Depth information with any program that uses the Kinect and can store this data. This data can then be played back as if it is live data from the Kinect.
CHAPTER 3. METHODOLOGY

This is useful because a system can be tested on data that won’t change so the improvements will be clear.

As the project was undertaken offline it was decided to take snapshots of the images and depth data. This was necessary for sending the RGB and Depth data into the offline program for segmenting the ball and matching the features as writing the Depth data to a file, for $320 \times 240$ pixels, took a large amount of time and taking manual snapshots of the RGB and Depth data made sure that there was no lag between the RGB and depth image so that both would be seeing the exact same scene. It also allowed for the data to be captured frame by frame at the full frame rate as can be seen in figure 3.3.

Figure 3.3 shows an example of the images taken using the snapshot method. Using this offline method the frame rate possible is quite high, i.e. the change in the images is quite small.

However, as in figure 3.4 when the Depth and RGB images are stored as fast as they can be stored continuously there is significant change between frames, so much so that the features that will be matched will be mostly incorrect.

The Depth data is stored in a text file in terms of the X and Y coordinate, in RGB space, and its depth value. There are $320 \times 240$ depth pixels so these will be stored in the text file. The code for reading from the Kinect is based on an example found on [8]. This was code
to read the RGB and depth information from the Kinect and display them. This code had to be modified to take snapshots of the video in order to allow for the full frame rate to be read from the Kinect, i.e. each frame can be taken without having to worry about how fast it reads. The code was also modified to map the depth image to colour space using the function described in the next section. A system for writing out the colour images and depth data was also added. It was felt that the important part of this project was not in the reading of the Data but in its processing so it was decided that using this code as a starting base was acceptable.
3.3 Ball Segmentation

Once the RGB and Depth data are stored in their .jpg and .txt files, respectively, and they are imported into the FYPOffline project, they can then be processed. The information is first used to segment the ball from the background using OpenCV. It was decided that background subtraction would be used as the scene was constrained to the same background and the lighting conditions were not changed during each video of the scene. The first frame captured for each of the datasets was the frame where there was no ball present.

3.3.1 Colour Segmentation

The background used for subtraction was shown in figure 1.1. Colour background subtraction was tested first and it was found that there were some problems with just using colour background subtraction.

![Figure 3.5: Ball Segmented By Colour Subtraction](image)

It can be seen, in figure 3.5 that the shadow of the arm in this image has affected the depth segmentation. This is because, in order to get at good circular mask it was decided to choose the row and column which had the largest width and height respectively to figure out the location of the centre of the circle. If there is shadow, then the subtraction image, with thresholding and morphological operations, has large regions of white space in the mask that are caused by the shadow, as shown in figure 3.6.

From figure 3.6 it can be seen that the widest area is not at the centre of the ball, but occurs below it in the image, where the shadow occurred in the RGB image.

3.3.2 Depth to Colour Mapping

In order to solve this problem it was decided to use the Depth data that the Kinect provides in order to do Depth subtraction. However there is a problem, the depth data is recorded in 320×240 and the RGB data is recorded in 640×480 resolution. This means that creating a depth mask will not work to mask the RGB image as the images are of different sizes. The first assumption that one would make would be to simply quadruple the size of the image.
CHAPTER 3. METHODOLOGY

Figure 3.6: Binary Mask when there is shadow

and make four pixels, in the image of size $640 \times 480$, have the depth value of one pixel in the $320 \times 240$ depth image. So if one wanted to find the depth value of a particular point in the RGB image they could just halve its location in the $x$- and $y$-axes and round down. Figure 3.7 shows a depth image, at a resolution of $320 \times 240$. It can be seen that the centre point of the ball, as found by the colour segmentation, where there were no shadows, is not at the centre of the ball as it should be. This is even more clear in the right image in figure 3.7 where the circular mask that was on the RGB image is converted to $320 \times 240$. It can be seen that the mask does not match the location of the ball in the image.

Figure 3.7: Depth Image (left), Depth Image with mask(right)

The reason that one cannot just scale up or down the image to match the Depth to the RGB image is due to the fact that the Kinect’s IR camera for detecting depth and its RGB camera are not coincident as was seen in figure 3.1. This means that the views they see of
the same scene are slightly different.

Luckily the Kinect SDK has a function `NuiImageGetColorPixelCoordinatesFromDepthPixel` which can be used to convert the pixels in the depth image into their corresponding values in the RGB image.

```c
NuiImageGetColorPixelCoordinatesFromDepthPixel(NUI_IMAGE_RESOLUTION_640x480, NULL, depthX, depthY, realDepth<<3, &ix, &iy);
```

This function takes in the X-value of the current pixel in the depth image (`depthX`), the Y-value of the current pixel in the depth image (`depthY`) and the depth value (`realDepth`). It returns the X and Y coordinates in the RGB image that correspond to that depth pixel (`ix` and `iy`).

Figure 3.8 shows the depth pixels superimposed over the RGB pixels. To show the change in depth in this image, different colours have been used for different depth ranges. Pixels which are black represent a depth of less than 800 mm. This is the smallest depth that the standard Xbox 360 Kinect can get reliable values for. The furthest that the standard Kinect can measure reliably is 4m. Depths from 800 mm to 1000 mm are shown in red. 1000 mm to 1200 mm is shown in yellow. 1200 mm to 1400 mm is shown in green. 1400 mm to 1600 mm is shown in cyan. 1600 mm to 1800 mm is shown in blue. 1800 mm to 2000 mm is shown in purple and anything further is shown in white.

![Figure 3.8: Depth and RGB image mapped for image with ball (left) and background (right)](image)

It can be seen in figure 3.8 that the RGB image can still be seen through this depth overlay. This because the 320×240 image has been stretched up. This means that a certain pixel in the RGB image may still not have any depth value associated with it. It was decided to compute the distance between the pixel for which the depth value is required and the surrounding pixels that have depth values associated with them using the standard distance between two points formula: 

\[ d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \]

The pixel which is the smallest distance from the chosen pixel is the one whose depth value is used.

### 3.3.3 Depth Segmentation

Now that the depth has been matched to the RGB image it can be used to segment the ball. Shown in figure 3.9 is an image of the depth segmentation mask for a particular frame.

Figure 3.9 has been scaled down to fit on the page. If one zooms into the individual depth points can be seen as before. This is shown in figure 3.10.
CHAPTER 3. METHODOLOGY

From figure 3.9 it is clear that it is the ball that has caused the change in depth, as expected. It can be seen that there is some noise where the walls meet the floor and where the walls running perpendicular to one another meet. This noise is common with the Kinect as the reflections of the IR rays would be different from the floor than the wall and at the area right where they meet there may be conflicting data. This type of noise is small and can be removed by performing an Opening operation which erodes a binary image and then dilates it. Firstly, however, a closing needs to be done in order to fill any gaps in the image, as there will be gaps within the area of the ball after the subtraction. If these gaps are not closed first it would mean that the opening would remove all of the pixels within the ball and the resulting image would be blank. After performing these operations the depth mask is ready and is shown in figure 3.11.

This can then be used to mask the RGB image containing the ball, as shown in figure 3.12.

It can be seen that there is an area to the right of the ball in this image in which some of the background has been segmented also. This is because there is an area around the ball, as originally seen in figure 3.2 which is found to have zero depth. This is a common occurrence with the Kinect when the object is close to the camera. The effect can be seen to be reduced as the ball moves further away.
In order to remove this effect, Colour segmentation is used at this point. Colour can now be used as the areas that are being subtracted are smaller and the effect of the shadows such as those thrown by the arm, as shown earlier in figure 3.5, are significantly reduced.

The colour images that are to be subtracted are shown in figure 3.13. Subfigure 3.13a shows the current frame that is being analysed masked with the depth mask. Subfigure 3.13b shows the background image when the depth mask is applied.

When these two images are subtracted in RGB space, using the OpenCV \texttt{absdiff()} function, the resulting image is shown in figure 3.14. It can be seen from this image that there is no effect from the shadow that was present in the original RGB image.

Thresholding the image to make the image binary, as it is a colour image, and then applying the same morphological operations as were applied to the depth difference image, results in the image shown in figure 3.15.
It can be seen that this is a much better mask than was seen in figure 3.11. This is even clearer in figure 3.16, when the mask is applied to the RGB image.
This mask is then approximated with a circle in order to create a better mask, as a sphere, when viewed in two dimensions, appears as a circle. The circular mask is shown in figure 3.17. The centre and radius of the circular mask are found by computing the widest row and tallest column as these are most likely the row and column at which the centre occurs.

Once this mask has been computed, it is applied to the RGB image. This is the image used for matching features. The RGB image with the circular mask is shown in figure 3.18. It can be seen that for this case the mask is very good and is considered good enough to run the SIFT matching on.
3.4 Features

As stated earlier, it was decided to use SIFT features as the technique to find correspondences between the frames at time $t$ and time $t+1$.

3.4.1 Feature Detection

Once the ball has been isolated from each of the two frames being compared (frame at time $t$ and time $t+1$), features are located in each image using SIFT. An example of the Features extracted for one frame is shown in figure 3.19.

It can be seen that there have been many features computed for this particular image. Not all of features will be matched in the next frame as some features will be out of view as the ball rotates, some motion blur may blur some of the features or insufficient resolution.
CHAPTER 3. METHODOLOGY

3.4.2 Feature Matching

These features are then matched between frames using the SIFT feature matching functions in OpenCV. An example of feature mapping between two frames, in which the ball has already been segmented from the image is shown in figure 3.20

Figure 3.20: SIFT matches between frames at time t (left) and time t+1 (right)

If one looks closely at the segmented balls in figure 3.20 it can be seen that there are indeed less features which are compared between frames. It can also be seen that there are some spurious features which are found, such as the feature in the left image which is found in the black area. This is obviously an incorrect feature. However it is known that between frames, at 30 frames per second, the ball will not have moved a great deal. This means that a
viability threshold can be set for the matches. In this case the viability threshold was set to be ±10 pixels in the x- and y-direction. This ensures that points which are definite outliers are not used for the next stage.

3.5 Converting Image Features to Real World Coordinates

The SIFT matching algorithm returns the points in each image whose SIFT features have matched. These points have an x- and a y-coordinate in pixel space, i.e. the pixels that those features occur on in the image. From the Depth Data, which is now mapped onto the RGB data, the z-coordinate of the features is known by passing the x- and y- coordinates into the function to find the depth at this point, measured by the Kinect, by finding the nearest matching depth pixel to the current pixel. The depth is known in millimetres. By Similar triangles, given that the focal length of the Kinect is provided in the SDK, the X- and Y- real world coordinates can be found in millimetres.

Shown in figure 3.21 is an example of how to convert the Image Coordinates to Real World Coordinates. \( x_i \) is the x-coordinate in the image, \( y_i \) is the y coordinate in the image, \( f \) is the focal length of the camera, \( X_w \) is the real world x-coordinate, \( Y_w \) is the real world y-coordinate and \( Z_w \) is the real world z-coordinate as measured by the Kinect.

![Figure 3.21: Converting Image Coordinates to Real World Coordinates](image)

This leads to the formula: \( X_w = Z_w \times \frac{x_i}{f} \) and the corresponding formula for the Y coordinate: \( Y_w = Z_w \times \frac{y_i}{f} \)

Shown in subfigure 3.22a are the points in pixel space and shown in subfigure 3.22b are the same points in real world space, both shown in Excel.

It should be noted that the values in pixel space have the origin at the bottom left corner of the image (once converted from OpenCV to standard representation), but in order for the conversion to work the origin was transferred into the centre of the image, i.e. the camera is at the centre of the image. This is the reason that the pixel coordinates have no negative values but the real world coordinates do.
CHAPTER 3. METHODOLOGY

3.6 Computing the Rotation and Translations

Once the coordinates of the correspondences have been computed the rotations and translations are then to be computed. Excel was the tool used to compute these rotations and translations. The main reason that Excel was chosen, despite the fact that it can be slow, is due to its built-in Solver Add-in. This Add-in allows Excel to use generalised reduced gradient to approximate and optimise criteria based on a set of changeable values that will affect the criteria.

Equation 3.1 defines the affine transformation between the point position at time \( t \) and at \( t+1 \).

\[
P_{t+1} = R \times P_t + T
\]  

(3.1)

Where \( R \) is the Rotation Matrix, \( T \) is the Translation Vector, \( P_t \) is a 3D point at time \( t \) and \( P_{t+1} \) is the same point at time \( t+1 \). \( R \) is a combination of the following matrices:

\[
R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix}
\]

\[
R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}
\]

\( \beta \) is the angle of rotation about the Y-axis.
$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$\gamma$ is the angle of rotation about the Z-axis.

In order to get a matrix which combines all of the Rotations these are multiplied together as follows:

$$R_{tot} = R_xR_yR_z \quad (3.2)$$

The calculations for this are shown in Appendix A.

Once the overall rotation matrix has been computed it can be split into the following equations. $x_{t+1}$, for instance, is the individual elements of the first row of the overall rotation matrix multiplied by the column vector containing the 3D point at time $t$. The centre point of the ball, at time $t$, is subtracted from each of the points at time $t$ before they are rotated in order that the rotations are about the centre of the ball and not the origin of the space. The translation component is added to these equations in order to estimate the translation as well as the rotation. The last term on each equation is the centre of the ball at time $t$, which is added so that when the translations are calculated by M-estimators that they will be computed as translations from the location of the ball and not the origin.

The full equations for the 3D points at time $t+1$ are shown below:

$$x_{t+1} = \cos(\beta) \cos(\gamma)(x_t - x_c) - \cos(\beta) \sin(\gamma)(y_t - y_c) + \sin(\beta)(z_t - z_c) + t_x + x_c \quad (3.3)$$

$$y_{t+1} = \cos(\alpha) \sin(\gamma)(x_t - x_c) + \sin(\alpha) \sin(\beta) \cos(\gamma)(x_t - x_c) + \cos(\alpha) \cos(\beta)(y_t - y_c) - \sin(\alpha) \sin(\beta) \sin(\gamma)(y_t - y_c) - \sin(\alpha) \cos(\beta)(z_t - z_c) + t_y + y_c \quad (3.4)$$

$$z_{t+1} = \sin(\alpha) \sin(\gamma)(x_t - x_c) - \cos(\alpha) \sin(\beta) \cos(\gamma)(x_t - x_c) + \sin(\alpha) \cos(\gamma)(y_t - y_c) + \cos(\alpha) \sin(\beta) \sin(\gamma)(y_t - y_c) + \cos(\alpha) \cos(\beta)(z_t - z_c) + t_z + z_c \quad (3.5)$$

Once the equations above, which describe a model for 3D rotations and translations, have been formulated then the next step is to find an optimum solution to these equations, where the variables are the angles $\alpha$, $\beta$, $\gamma$, $t_x$, $t_y$ and $t_z$. M-estimators was used as the optimisation technique of choice as explained earlier. These equations were confirmed to be correct using [16]. The Excel spreadsheet was set up to take the real world coordinates for both time $t$ and time $t+1$ and compute the estimated values of the points at time $t+1$ by minimising the M-estimators criterion using Excel’s Solver. The solver works as follows:

1. Inputs: Found Correspondences($x_t$, $y_t$, $z_t$ and $x_{t+1}$, $y_{t+1}$, $z_{t+1}$) and Initial values for desired variables ($\alpha$, $\beta$, $\gamma$, $t_x$, $t_y$ and $t_z$), figure 3.23 and figure 3.24
2. Estimate $x_{t+1}$, $y_{t+1}$ and $z_{t+1}$ using equations 3.3, 3.4, 3.5 from figure 3.25
3. Find the difference between the estimated value of $x_{t+1}$, $y_{t+1}$ and $z_{t+1}$ and the known values of $x_{t+1}$, $y_{t+1}$ and $z_{t+1}$ provided as input. This is the error, shown in figure 3.26
4. Find the square of the error and use it in the M-estimators formula, shown in figure 3.26.
CHAPTER 3. METHODOLOGY

5. Add the M-estimator errors for the x, y and z components for each point correspondence, shown in figure 3.26

6. Objective Cell, in Excel, or Criteria cell is the sum of the M-estimator errors, column 7 in figure 3.26

7. The solver is run with the Objective cell as the Criteria and the Variable cells as the values for $\alpha$, $\beta$, $\gamma$, $t_x$, $t_y$ and $t_z$ set to minimise setting.

8. If it is the first set of correspondences analysed then the Multistart option is set in the Options menu for the Solver and a set of constraints, which limit the variable values to a certain range are added. For later sets of correspondences, the computed values for the Rotation and Translation variables of the previous set of correspondences can be used as the initial values for the $\alpha$, $\beta$, $\gamma$, $t_x$, $t_y$ and $t_z$ values and a local solution is then found without Multistart.

It was found that using the previous result of the Rotations and Translations as the initial estimate for the variable values for the next set of images is a good way to improve the results, as it is known that the rotation of a ball will not change greatly between frames. This idea was mentioned by Hersch et al [10] in their paper as a method commonly used in tracking scenarios when it is known that the object being tracked cannot change position greatly between frames. An example of an Excel sheet for this project is shown in Appendix C

Figure 3.23 shows the set of correspondences between the frame at time t and the next frame at time t+1. These are the actual values that were computed using SIFT.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$y_1$</th>
<th>$z_1$</th>
<th>$x_{t+1}$</th>
<th>$y_{t+1}$</th>
<th>$z_{t+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-37.8462</td>
<td>72.40199</td>
<td>874</td>
<td>-30.3982</td>
<td>72.6179</td>
<td>897</td>
</tr>
<tr>
<td>-58.9325</td>
<td>68.48913</td>
<td>846</td>
<td>-20.6559</td>
<td>78.44112</td>
<td>868</td>
</tr>
<tr>
<td>-72.9813</td>
<td>16.38005</td>
<td>881</td>
<td>-90.7904</td>
<td>20.26546</td>
<td>897</td>
</tr>
<tr>
<td>-48.6303</td>
<td>98.88167</td>
<td>861</td>
<td>-38.4524</td>
<td>101.9825</td>
<td>888</td>
</tr>
<tr>
<td>-71.1663</td>
<td>64.84044</td>
<td>840</td>
<td>-61.4553</td>
<td>82.47953</td>
<td>859</td>
</tr>
<tr>
<td>-83.8181</td>
<td>50.60717</td>
<td>840</td>
<td>-77.447</td>
<td>64.53921</td>
<td>857</td>
</tr>
<tr>
<td>-39.4917</td>
<td>87.21077</td>
<td>874</td>
<td>-30.3982</td>
<td>89.50579</td>
<td>897</td>
</tr>
<tr>
<td>-106.241</td>
<td>135.216</td>
<td>855</td>
<td>-94.5546</td>
<td>130.434</td>
<td>866</td>
</tr>
<tr>
<td>-57.0409</td>
<td>120.6514</td>
<td>856</td>
<td>-66.7662</td>
<td>120.1017</td>
<td>886</td>
</tr>
<tr>
<td>-113.798</td>
<td>51.97402</td>
<td>828</td>
<td>-102.54</td>
<td>92.92567</td>
<td>851</td>
</tr>
<tr>
<td>-68.1053</td>
<td>85.60294</td>
<td>842</td>
<td>-58.2208</td>
<td>90.56575</td>
<td>859</td>
</tr>
<tr>
<td>-111.852</td>
<td>75.71496</td>
<td>914</td>
<td>-96.404</td>
<td>75.37042</td>
<td>931</td>
</tr>
</tbody>
</table>

Figure 3.23: Points at time t in mm (first three columns) and points at time t+1 in mm (last three columns)

Figure 3.24 shows the variable cells which the solver changes. These contain the Rotation and Translation variables. It should be noted that A is angle $\alpha$, B is angle $\beta$ and G is angle $\gamma$. The solver changes these values in order to change the value of the criteria.

Figure 3.25 shows the estimated values of $x_{t+1}$, $y_{t+1}$ and $z_{t+1}$ when the angles and translation values are put into equations 3.3, 3.4, 3.5. Column 1 uses equation 3.3 and so on. This is done for every value of $x_t$, $y_t$ and $z_t$. 
CHAPTER 3. METHODOLOGY

3.7 3D Graphics

For this project a simple 3D graphics system was created to show the rotations and translations in 3D. An image from this model is shown in figure 3.27.

It is simply a 3D model of a sphere generated in OpenGL with a texture placed on it.
CHAPTER 3. METHODOLOGY

<table>
<thead>
<tr>
<th>$e_x$</th>
<th>$e_y$</th>
<th>$e_z$</th>
<th>$e_x^2/(1+e_x^2)$</th>
<th>$e_y^2/(1+e_y^2)$</th>
<th>$e_z^2/(1+e_z^2)$</th>
<th>$M_x + M_y + M_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.1638</td>
<td>1.702319</td>
<td>3.773905268</td>
<td>0.575268654</td>
<td>0.743450999</td>
<td>0.934393496</td>
<td>2.25311315</td>
</tr>
<tr>
<td>-1.20577</td>
<td>9.826826</td>
<td>3.580256907</td>
<td>0.552483536</td>
<td>0.98975058</td>
<td>0.927631381</td>
<td>2.509866</td>
</tr>
<tr>
<td>5.621596</td>
<td>4.142435</td>
<td>-0.206575726</td>
<td>0.969327354</td>
<td>0.944488426</td>
<td>0.04092703</td>
<td>1.95474281</td>
</tr>
<tr>
<td>0.226775</td>
<td>3.835413</td>
<td>-7.25367913</td>
<td>0.048911372</td>
<td>0.936347879</td>
<td>0.981348803</td>
<td>1.96560805</td>
</tr>
<tr>
<td>0.130991</td>
<td>16.90199</td>
<td>1.104610797</td>
<td>0.016869241</td>
<td>0.996511757</td>
<td>0.549583029</td>
<td>1.56296403</td>
</tr>
<tr>
<td>-2.75274</td>
<td>12.74712</td>
<td>-0.00322625</td>
<td>0.863417009</td>
<td>0.993883384</td>
<td>1.04685E-05</td>
<td>1.8773108</td>
</tr>
<tr>
<td>-0.01</td>
<td>3.740667</td>
<td>3.343538858</td>
<td>9.99326E-05</td>
<td>0.933003371</td>
<td>0.917893152</td>
<td>1.85129345</td>
</tr>
<tr>
<td>0.249248</td>
<td>-6.16677</td>
<td>-8.007650152</td>
<td>0.058490808</td>
<td>0.974378036</td>
<td>0.984644315</td>
<td>2.01751315</td>
</tr>
<tr>
<td>-0.14855</td>
<td>0.078456</td>
<td>-0.171194411</td>
<td>0.021619192</td>
<td>0.006117639</td>
<td>0.028473052</td>
<td>0.05620988</td>
</tr>
<tr>
<td>0.315705</td>
<td>-1.5917</td>
<td>5.634811338</td>
<td>0.050636013</td>
<td>0.716994279</td>
<td>0.969466673</td>
<td>1.77709696</td>
</tr>
<tr>
<td>-0.24937</td>
<td>4.414109</td>
<td>-1.575012935</td>
<td>0.058345648</td>
<td>0.951182269</td>
<td>0.737234064</td>
<td>1.7456198</td>
</tr>
<tr>
<td>8.02935</td>
<td>-0.11839</td>
<td>0.197507037</td>
<td>0.964725935</td>
<td>0.013821447</td>
<td>0.037544457</td>
<td>1.03609184</td>
</tr>
</tbody>
</table>

Figure 3.26: The error between the the point at time $t$ and $t+1$ (three left most columns), M-estimators for $x, y$ and $z$ (column 4, 5 and 6), Sum of M-Estimators (column 7)

This texture allows the Rotations to be seen easily. If the videos, attached with the project, are viewed one can see that the movement of the ball is indeed similar to the real movement of the ball in the video taken by the Kinect. The code for this was based on the sample code provided for the CS4052 Computer Graphics Course [15].
Chapter 4

Experimental Results

The main aim of this Project was to infer the Rotation and Translations of a ball as it moves down the lane. As already stated, the experiment was done on a small scale setup using a football, instead of using a bowling ball in a bowling alley.

4.1 Ideal Case

For the initial experiments ideal data was used to test the M-estimators and Least Squares techniques for finding an optimum solution from the set of data. By choosing points carefully so that the correct output was known it was possible to test if the system would compute the correct Rotations and Translations.

4.1.1 2D case

Least Squares

The first case tested was the ideal 2D scenario. In this test there were a set of 2D points chosen as the 'Initial Data', which, at the beginning, consisted of a square with a point at its centre as shown in figure 4.1.

These points in the X- and Y-direction are as follows: \((x, y) = \{(1, 1), (2, 1), (1, 2), (2, 2), (1.5, 1.5)\}\). These are considered the X and Y values at time \(t\) so from now on they will be referred to as \(x_t\) and \(y_t\). At time \(t+1\) the X-coordinate of the points are referred to as \(x_{t+1}\) and the Y-coordinates are referred to as \(y_{t+1}\). When the system is using the real data computed using SIFT features, as explained earlier, time \(t\) refers to the current frame and time \(t+1\) refers to the next frame.

For testing the ideal case, a known 60° rotation is set up and then the model chosen, which was described in the State of the Art section previously, was tested to see did it transform the points at time \(t\) to the points at time \(t+1\) accurately. The points used in order to set up a positive rotation of 60°, a translation in the X-direction of 2.5 units and a translation in the Y-direction of 2.5 units, are as follows:

\((x_{t+1}, y_{t+1}) = \{(4.2, 3.3), (4.7, 4.2), (3.3, 3.8), (3.8, 4.7), (4, 4)\}\) Each of these points are the location of the original points at time \(t+1\). Using Excel’s Solver Add-on the optimum angle of Rotation and the X and Y Translations were computed. It was seen that the Rotation was indeed 60° and the translations were 2.5 and 2.5 respectively. This is exactly what was expected. The result of this is figure 4.2. The blue points are the points at time \(t\), the red
CHAPTER 4. EXPERIMENTAL RESULTS

Figure 4.1: Ideal 2D input data

points are those at time t+1 and the grey points are the estimated values of the points at time t+1 when rotated and translated from their positions at time t.

Figure 4.2: Ideal 2D with no outliers

The model was fit using the Least Squares algorithm in this case and it can be seen that the least squares algorithm has effectively found the correct rotation and translations. In this case the data is ideal and there is no outliers but, as stated previously, the Least Squares algorithm is not robust in the presence of outliers. Shown in figure 4.3 is the ideal data with a fifth point added.

It can be seen that this data has an extra point at \((x_t, y_t) = (1.5, 2.5)\). In the “Transformed
Data" this point was set to not lie where it should if the transformation was done correctly on that point. Instead it lies further away, which makes it an outlier. Figure 4.4 shows this.

![Figure 4.3: Ideal 2D with outlier](image)

It can be seen that the outlier has caused significant effect on the estimation of the Rotation and Translations, as shown in grey and the red which are in the correct positions are not coincident with their respective grey points. Figure 4.5 shows the effect of an outlier which is even more erroneous. It can be seen that its effect is very significant and the Rotation and Translation are quite obviously very incorrect.

In fact, the rotation was computed as $\approx 110^\circ$, the translation $t_x$ was $\approx 2$ and the trans-
CHAPTER 4. EXPERIMENTAL RESULTS

Figure 4.5: Ideal 2D with one far outlier using Least Squares

lation $t_y$ was $\approx 2$ also. These values are very wrong and it can be seen that they are much worse than the values computed with the closer outlier. It is clear that the further the outlier and the more outliers, the worse the approximation of the inliers.

**M-estimators**

It was clear that when there are outliers the Least Squares method fails to accurately obtain the best solution for the Rotation and theTranslations as previously expected, and that as the outliers get more prominent and numerous the error gets worse. As explained earlier, in order to reduce the effect of outliers on the calculations the M-estimators method is used as a substitute for the Least Squares method. The improvements delivered by the M-estimators are clear from the figures below, which have the same Data points provided at time $t$ and time $t+1$ but now use M-estimators as the criteria to minimise. Figure 4.6 shows the estimation of the data, with the closer outlier, but using M-estimators instead of Least Squares.

It is clear, when this graph is compared to the graph containing the Least Squares solution in figure 4.7, that M-estimators provides a better result that is less effected by the outlier and the rotations now better approximate the rest of the data, which are considered inliers.

This effect is even more prevalent in the figure 4.8, which compares the Least Squares against the M-estimators when the outlier is much further away from the inliers.

It can be seen, as discussed earlier, that the outlier causes a large amount of error in the case of Least Squares (right), larger than that of the closer outlier, but it can be seen that with M-estimators, the far outlier actually has the opposite effect. The M-estimators has estimated the rotations and translations well for the rest of the data and has nearly totally removed the effect of the outlier at (0, 0). This is a very useful property of M-estimators, i.e. the worse the outlier the less it effects the solution.
CHAPTER 4. EXPERIMENTAL RESULTS

Figure 4.6: Ideal 2D with one outlier using M-estimators

Figure 4.7: Ideal 2D Comparison of Least Squares and M-estimators for one close outlier

It can also be seen that the the same can be said for multiple outliers, M-estimators improves the estimate of the Rotations and Translations to better fit the inliers just as it did for one outlier. It can be seen that the more outliers the greater the effect they have but when all of the outliers are far from the inliers they have little effect on the computed Rotations and Translations. Figure 4.9 shows a comparison of the Least Squares and M-estimators techniques for multiple outliers.

It can be seen that the four distant outliers have had a very large effect on the estimation, making it completely incorrect, in the Least Squares example (left). However the M-estimators (right) has computed the Rotations and Translations for the inliers very well. These examples
have confirmed the merits of M-estimators as a method for finding the optimum solution to the problem of computing the Rotations and Translations.

4.1.2 3D case

The 2D case is good for confirming the idea of M-estimators as the method of choice for the solution to the problem of minimising the error but it is limited as the real world is three dimensional and therefore 2D information is not very effective for computing the real world Rotations and Translations. As was stated earlier, for the 3D case there are 6 parameters...
that need to be evaluated in order to obtain the real world Rotations and Translations, the Rotations about the X, Y and Z axes and the Translations in the X, Y and Z axes. This means that it is necessary to have 6 equations to solve in order to get a single solution, i.e. 6 simultaneous equations in 6 variables. If there is less than six equations then information about the Rotations and Translations can be gained but not a full solution as there will be variables that are still quite undefined.

In the case of this project it is necessary to find 6 correspondences between each image in order to gain a single solution.

The first test done for the 3D case was to ensure that the formulae for computing the X, Y and Z coordinates after a particular Rotation and Translation was applied, discussed in chapter 3, worked correctly. This was done by choosing the angles of rotation and the distances for translation. Firstly a 45 degree rotation about the X-axis was applied to the 3D ideal data. This was done for all the axes and it was found that the formulae worked to apply the Rotations correctly.

Shown in figure 4.10 below is a rotation about the X-axis. Then in figure 4.11, a rotation about the Y-axis. Figure 4.12 shows a rotation about the Z-axis.

From examining these diagrams it is clear that the model is working correctly for the supplied angles of 45°.

Figure 4.13 shows the effect of outliers on the 3D data using Least Squares as the optimisation technique.

It is clear that the outlier’s effect is significant in this case. As stated previously, this is because the further the outlier the greater the error and, using least squares, the error squared term causes outliers to have an even larger effect when they are further from the inliers.
CHAPTER 4. EXPERIMENTAL RESULTS

Figure 4.11: Ideal Rotation about the Y-axis.

Figure 4.12: Ideal Rotation about the Z-axis.

M-estimators

However it can be seen that the outlier’s effect on the estimation in figure 4.14 (green points) is very small. Using M-estimators has made the effect of the outlier insignificant and the estimated data matches the expected data once the Rotations and Translations have been applied.

It can be seen that some of the green points are visible and some are not, the same is
CHAPTER 4. EXPERIMENTAL RESULTS

Figure 4.13: Outlier Effect 3D Least Squares

Figure 4.14: Outlier Effect 3D M-estimators
true for the red points. In this diagram they are coincident and Matlab (which was used to
draw 3D point scatter plots, which are not available in Excel) can only draw one or the other
on the diagram.

Figure 4.15 is the same 3D diagram shown from a different angle. This diagram confirms
that the points mentioned earlier are in fact coincident.

![Figure 4.15: Outlier Effect 3D M-estimator, different angle](image)

4.2 Real Data - Very Slow Movement

The next stage was to use data generated from the real scene using a controlled slow moving
ball. The ball was segmented and its features computed. The current frame and the next
frame were examined frame by frame. Using M-estimators the 3D Rotations and Translations
were computed, as explained earlier, for this data. Only sets which contained 6 or more
correspondences were used, as 6 is the minimum number of correspondences necessary to
properly solve for 6 variables \((\alpha, \beta, \gamma, t_x, t_y, t_z)\). Shown below is the result of the Solving
operation. As before, the blue points are the “Initial Data” (The matched points for time
t), the red points are the “Transformed Data” (The matched points at time \(t+1\)) and the
green points are the “Estimated Data” (The points at time \(t\) transformed by the computed
rotations and translations).

It can be seen that the green “Estimated Points” don’t precisely match up with the red
points which were found in the image. This may be because the SIFT features matched
between the images are not all correct. However even though the points don’t match up
precisely, as in the Ideal case, it is clear from the diagram that the rotations and translations
do give a decent representation of the rotation and translation for the two frames used. For these two frames it was found using the Excel spreadsheet that $\alpha = 8.44^\circ$, $\beta = 2.31 \times 10^{-5}^\circ$, $\gamma = -5.25^\circ$, $t_x = 8.196mm$, $t_y = -14.485mm$, $t_z = 7.16mm$.

It was found that it was difficult to move the ball consistently, as if it was rolling, when the ball was moved using the manipulators and not actually rolling the ball. It was discovered that instead of improving the results, they were made worse, as the assumption that the ball moves very little and consistently between frames is not accurate in this case.

### 4.3 Real Data - Full Movement

It was decided that in order to get a consistent rotation it was necessary to actually roll the ball instead of moving it with manipulators. It should be noted that the movement shown here is by no means at the same velocity as a bowling ball would travel at down a real bowling lane.

It can be seen that there is some significant motion blur evident in the image in figure 4.17. This causes problems when attempting to compute features on the ball. However, when the ball is rolled slower there are much better results such as in figure 4.18.

It can be seen that in this image there is less motion blur and therefore it is more useful for testing.

The Rotation angles and Translations, for the slower data set are summarised in figures 4.19, 4.20, 4.21, 4.22, 4.23 and 4.24. It was found that they produced a quite consistent rotation, with respect to the actual rotation that can be seen in the RGB video.

It can be seen that the Rotation about the X-axis has changed considerably, despite the fact that the rotations, when viewed in the 3D model, look consistent. This is because there are different ways to get the same overall rotation from 3 rotation angles. If figure 4.21 is viewed it can be seen that the angle about the Z-axis is changing also. The overall rotation created by these angles together is similar, despite the individual angle difference.
It can be seen that for this particular roll of the ball that the angle $\beta$ remains at about $0^\circ$ (as $-360^\circ = 0^\circ$). It does not have much effect, in this case.

It can be seen that the translations are all in the positive direction, as expected, as the ball is moving from left to right in the video sequence, which implies a positive translation in the X-direction.

The translation in the Y-direction is caused by mismatches in the data and can be seen to get worse as the ball moves further from the camera. There should be no translation in the Y-direction as the ball remains on the ground and this is a mistake caused by mismatches.
Given that it is known that the ball is not moving in the Y-direction the movement in the Y direction is considered to be 0.

The Z translation can be seen to be in the positive direction also, i.e. the ball is moving away from the camera, as was expected. When the overall trend for both the X and Z direction is viewed, by summing the Translations in the X direction and summing the translations in the Z-direction, it can be seen that both show the trend of moving in the positive direction.
CHAPTER 4. EXPERIMENTAL RESULTS

Figure 4.21: Angle of Rotation $\gamma$ between frames about the Z-axis

Figure 4.22: Translation $t_x$ between frames in the X-direction

(figure 4.25, figure 4.26). This means that the ball is moving right and forward (away from the camera). If the trend had been in the negative direction then the ball would have been moving left and towards the camera.

Taking a Norm vector of the X and Z translations results in data in figure 4.27. $t_y$ was not included as the Y-translation was considered to be 0 given that the ball is rolling on a flat plane. The Norm Vector was computed using equation 4.1.
\[ \|t\| = \sqrt{x^2 + y^2 + z^2} \] (4.1)

The norm Vector shows the overall Translation magnitude. It can be seen that the translation increases at the beginning, with an initial acceleration and is beginning to decrease near the end as there is some deceleration of the ball due to friction.

It can be seen in the videos of the 3D model, attached with this report, that the 3D model
is moving in the correct direction and with similar rotations to those seen in the real RGB image. This shows that the system is estimating the Rotations and Translations quite well.

However, this ball was rolling quite slowly and there was very little motion blur in the images. This allowed for better correspondences. When another video was used, with the ball moving faster it was found that the Rotations were not as well estimated. It is believed that this is caused by a mixture of motion blur and low resolution. It was found that for a
few of the image comparisons that there were few good feature matches, which means that the Rotations and Translations cannot be estimated well for these frames. This is clearest in frames 3-5 in the GL video attached on the CD. In appendix B the graphs, similar to those that were shown above, are included. From these it can be seen that the frames 3-4 and 4-5 have a large change in angle that was not seen in the video. Each of these frames only had 2-3 good correspondences and this shows that it is necessary to compute at least 6 good correspondences to estimate the rotation correctly.

From these graphs it can be seen that the translations are mostly correct but $t_x$ has been miscalculated for the case of frames 3-5, which can be clearly seen in figure B.7 in the appendix. It can also be seen that frames 3-5 in the other graphs also have some strange values. Yet again, for the 3D graphics it was assumed that Y was 0 for all cases. In the graph for the depth translation it can be seen that the depth changed between frames is greater than those for the previous case. This suggests that the ball is indeed travelling faster than the previous case. It can also be seen that the change between frames in the Z-direction is quite consistent, except for frames 3-5 where there were less correspondences.
Chapter 5

Conclusion

The method proposed has been shown to provide an estimate of the spin of the ball in terms of rotations about each axis. The overall movement of the ball has been estimated by observing the rotations and the translations. The videos attached with this report show the systems working, including the OpenGL implementation.

The system showed promising results and computed rotations and translations which were close to those expected. It has been proven that it is possible to infer the spin of a 3D moving object, in this case a ball, using a 3D camera Vision system. It has also been proven that it is possible to do so without a learning type algorithm.

It was clear however that there are some flaws in the estimation, as the rotations and translations computed, though similar to the video, have visible flaws when compared to the real video. It was found that the resolution of the Kinect was quite low when this application is considered. Though the Kinect is a fine tool for tracking objects using depth, its resolution, being quite low, makes it unsuitable for tasks that require high resolution, such as this task. The SIFT feature matching may have been flawed because of this resolution and some features at further ranges may be missed. Other features which would normally be distinct may not be found at this resolution.

The Kinect’s frame rate was found to be quite slow when reading the Colour and Depth data at the same time. This required a more manual method for acquiring the RGB and Depth images which is undesirable for a consumer application. Also, the general frame rate can cause some motion blur in the frame when the ball is moving at more significant speed. Feature detection is unreliable on an image with significant motion blur.

M-estimators in this project has improved the rotation and translation estimation but many of the mismatched points may be part of the inlier data which means that M-estimators struggles to remove their effect. This means that better correspondences have to be found, which may be possible if the images were of better quality.

Speed is one of the main issues. As it was decided that this project would be undertaken offline, it was considered acceptable to split up the code and run everything offline. However, for a consumer based product it would be necessary for this to be more online oriented.

If this research were to be continued there are some areas of interest that would be investigated, especially if the project was to transition into a product for release. This project consists of many sections, i.e. the code to read the colour and depth data from the Kinect, the code to segment the ball from the image and track features, the Excel spreadsheets for computing the Rotations and Translations and the OpenGL code to display the computed
rotations, and therefore these sections would need to be combined together into one large program in C++ to run everything and provide the user with the desired Rotations and Translations without them having to do anything other than start it up.

Excel was a useful way to analyse the data for this project and use its built-in optimising Generalised Reduced Gradient Solver, but some of the techniques could be accomplished in C++ and be more easily repeatable as well as faster. As it is, in order to analyse new data the points need to be brought into Excel manually and the solver's parameters need to be changed for each set of cells.

A further investigation of other feature-based techniques, with a focus on speed as well as accuracy, would be done in order to improve the speed of the matching algorithm, especially if it was to be used for an online version of the application.

The OpenGL implementation would be improved by adding a more visually pleasing aesthetic and showing the actual bowling lane as well as some GUI based options for the user.

There is also still testing to be done on the system for the ball in different conditions, such as hitting of a wall or coming to a sudden stop. These are some testing areas that would be investigated to further improve the system, even though on a bowling lane the ball will not collide with a wall or come to a sudden stop.

Possibly the most important thing that would be changed about the design, as it currently is, would be to improve the 3D camera used, in order to increase the resolution and improve the matches. It has been seen that the depth camera is useful for segmenting the ball and finding its real world coordinates but that its RGB camera is not the best camera for this application. Inferring the spin of a spherical object based on depth alone is considered not viable as the depth image provides no information about the orientation of a sphere. This means that the RGB camera is of great importance and it would be necessary to get a better RGB camera if the research was to be continued. The range of the depth camera used would also need to be increased. The Kinect can only register depth values well if they are less than 4m from the camera, but a full size bowling lane is 19m long. This means that a more sophisticated 3D camera would be necessary. It may even be a good idea to change from a "Time Of Flight" camera (The Kinect) to a stereo setup. A high quality stereo setup may be cheaper to create than a very accurate "Time of Flight" camera, which would also need 19m of accurate range. There is a problem with using a stereo setup though, as it will be affected by shadow changes, which were dealt with effectively by the Infrared "Time of Flight" camera.

As a proof of concept, this project has been quite successful. It has been found that computing the spin of a ball using a 3D camera is possible and less complex than using a learning based method and if there was more time it is believed that this method could be further refined to produce better results. It is clear though that for the real world application of bowling, in which a much larger range and much faster movement is present, that the Microsoft Kinect is unsuitable for the task and that a better camera is needed.
Bibliography


Appendix A

3D Rotations

\[ R_{tot} = R_x R_y R_z \]

\[
R_x R_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \times \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix}
\]

\[
= \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ \sin(\alpha) \sin(\beta) & \cos(\alpha) & -\sin(\alpha) \cos(\beta) \\ -\cos(\alpha) \sin(\beta) & \sin(\alpha) & \cos(\alpha) \cos(\beta) \end{bmatrix} \times \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

\[
= \begin{bmatrix} \cos(\beta) \cos(\gamma) & -\cos(\beta) \sin(\gamma) & \sin(\beta) \\ \cos(\alpha) \sin(\gamma) + \sin(\alpha) \sin(\beta) \cos(\gamma) & \cos(\alpha) \cos(\gamma) - \sin(\alpha) \sin(\beta) \sin(\gamma) & -\sin(\alpha) \cos(\beta) \\ \sin(\alpha) \sin(\gamma) - \cos(\alpha) \sin(\beta) \cos(\gamma) & \sin(\alpha) \cos(\gamma) + \cos(\alpha) \sin(\beta) \sin(\gamma) & \cos(\alpha) \cos(\beta) \end{bmatrix}
\]

\[
\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \end{bmatrix} =
\begin{bmatrix} \cos(\beta) \cos(\gamma) & -\cos(\beta) \sin(\gamma) & \sin(\beta) \\ \cos(\alpha) \sin(\gamma) & \cos(\gamma) & -\sin(\alpha) \cos(\beta) \\ \sin(\alpha) \sin(\gamma) & \cos(\gamma) & \cos(\alpha) \cos(\beta) \end{bmatrix} \times \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}
\]

In order for this matrix equation to be useful in Excel it needs to be changed into standard equation form, i.e.:

\[
x_{t+1} = \cos(\beta) \cos(\gamma) x_t - \cos(\beta) \sin(\gamma) y_t + \sin(\beta) z_t \quad (A.1)
\]

\[
y_{t+1} = \cos(\alpha) \sin(\gamma) x_t + \sin(\alpha) \sin(\beta) \cos(\gamma) x_t + \cos(\alpha) \cos(\gamma) y_t - \sin(\alpha) \sin(\beta) \sin(\gamma) y_t - \sin(\alpha) \cos(\beta) z_t \quad (A.2)
\]

\[
z_{t+1} = \sin(\alpha) \sin(\gamma) x_t - \cos(\alpha) \sin(\beta) \cos(\gamma) x_t + \sin(\alpha) \cos(\gamma) y_t + \cos(\alpha) \sin(\beta) \sin(\gamma) y_t + \cos(\alpha) \cos(\beta) z_t \quad (A.3)
\]

In order to take the translations into account the translation term is added to the equations as shown below:

\[
x_{t+1} = \cos(\beta) \cos(\gamma) x_t - \cos(\beta) \sin(\gamma) y_t + \sin(\beta) z_t + t_x \quad (A.4)
\]
yi+1 = \cos(\alpha) \sin(\gamma) x_i + \sin(\alpha) \sin(\beta) \cos(\gamma) x_i + \cos(\alpha) \cos(\gamma) y_i - \sin(\alpha) \sin(\beta) \sin(\gamma) y_i - \sin(\alpha) \cos(\beta) z_i + t_y \\
zi+1 = \sin(\alpha) \sin(\gamma) x_i - \cos(\alpha) \sin(\beta) \cos(\gamma) x_i + \sin(\alpha) \cos(\gamma) y_i + \cos(\alpha) \sin(\beta) \sin(\gamma) y_i + \cos(\alpha) \cos(\beta) z_i + t_z 
\text{(A.5)} \\
\text{(A.6)}
Appendix B

Second Dataset

This appendix contains the graphs of the Rotations and Translations for the second dataset, mentioned at the end of chapter 4.

Figure B.1: Angle of Rotation $\alpha$ between frames about the X-axis
APPENDIX B. SECOND DATASET

Figure B.2: Angle of Rotation $\beta$ between frames about the Y-axis

Figure B.3: Angle of Rotation $\gamma$ between frames about the Z-axis
Figure B.4: Translation $t_x$ between frames in the X-direction

Figure B.5: Translation $t_y$ between frames in the Y-direction
APPENDIX B. SECOND DATASET

Figure B.6: Translation $t_z$ between frames in the Z-direction

Figure B.7: Sum of the translations in the X-direction
Figure B.8: Sum of the translations in the Z-direction
Appendix C

Excel Spreadsheet
### Figure C.1: Example of the Excel Spreadsheet used

<table>
<thead>
<tr>
<th>Local Length</th>
<th>Estimated $x_{\text{nu}}$</th>
<th>Estimated $y_{\text{nu}}$</th>
<th>Estimated $x_{\text{nu}}$</th>
<th>$x_{\text{nu}}$</th>
<th>$y_{\text{nu}}$</th>
<th>$e_\theta$</th>
<th>$e\theta$</th>
<th>COS($)</th>
<th>SINA($)</th>
<th>COS($)</th>
<th>SINA($)</th>
<th>COS($)</th>
<th>SINA($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>331.15</td>
<td>94.66</td>
<td>-217.73</td>
<td>835.13</td>
<td>32.2737</td>
<td>384.509</td>
<td>885.019</td>
<td>0.9993</td>
<td>3.6509</td>
<td>0.0000</td>
<td>-0.009</td>
<td>0.9911</td>
<td>0.1332</td>
<td></td>
</tr>
<tr>
<td>Estimated A</td>
<td>A1</td>
<td>-6.38</td>
<td>-213.57</td>
<td>887.73</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated B</td>
<td>B1</td>
<td>-5.00</td>
<td>-177.23</td>
<td>845.92</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated C</td>
<td>C1</td>
<td>1.40</td>
<td>213.86</td>
<td>828.87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated D</td>
<td>D1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated E</td>
<td>E1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated F</td>
<td>F1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated G</td>
<td>G1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated H</td>
<td>H1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated I</td>
<td>I1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated J</td>
<td>J1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated K</td>
<td>K1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated L</td>
<td>L1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated M</td>
<td>M1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated N</td>
<td>N1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated O</td>
<td>O1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated P</td>
<td>P1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated Q</td>
<td>Q1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated R</td>
<td>R1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated S</td>
<td>S1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated T</td>
<td>T1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated U</td>
<td>U1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated V</td>
<td>V1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated W</td>
<td>W1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated X</td>
<td>X1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated Y</td>
<td>Y1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated Z</td>
<td>Z1</td>
<td>2.18</td>
<td>203.90</td>
<td>835.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Notes:

- $e\theta$ values are approximate.
- All values are rounded to two decimal places.
Appendix D

Oil Pattern Example

Full image on next page
## APPENDIX D. OIL PATTERN EXAMPLE

Figure D.1: example of Kegal Oil pattern
Appendix E

CD contains Code, Videos and Excel Spreadsheets, see README.txt