Visual Tracking of Three-Axis Computer Numerical Control Machines for Intelligent Controllers

Olivier St-Martin Cormier
Centre for Intelligent Machines
McGill University, Montreal

December 2012

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of
Master of Engineering

Contents

Contents i

Front Matter **iv**

Abstract iv

Résumé v

Acknowledgments vi

1 Introduction **1**

1.1 Motivation 1

1.2 Background 1

1.3 Goals 2

1.4 System Requirements 2

1.5 Assumptions 3

1.6 Potential Problems 3

1.7 Potential Applications 4

1.8 General Goals 4

1.9 Thesis Overview 4

2 Literature Review **6**

2.1 Position-Based Tracking 6

2.2 Image-Based Tracking 7

3 Testing Platform **8**

3.1 Hardware Platform 8

3.2 Software Platform 8

3.3 Machine Simulator 9

3.4 Real Machine 10

4 Camera Calibration **11**

5 Marker-Based Detector **14**

5.1 Background 14

5.2 Overview 14

5.3 The Markers 15

5.4 The Examples 16

5.5 Ideal Bayes Classifier 18

5.6 Nearest-Neighbour Classifier 20

5.6.1 Reducing the Number of Examples 20
5.7 Sub-Pixel Sampling 21
5.7.1 Maximum Intensity Pixel (MIP) 22
5.7.2 Non Maxima Supression (NMS) 22
5.7.3 Intensity Barycentric Weighting (IBW) 22
5.7.4 Intensity Linear Interpolation (ILI) 22
5.7.5 Intensity Quadratic Fitting (IQF) 23
5.7.6 1D Intensity Gaussian Fitting (1DIGF) 23
5.7.7 2D Intensity Gaussian Fitting (2DIGF) 24
5.7.8 Comparison of the Methods 24
5.8 System Evaluation 25
5.8.1 Sensitivity to Noise 26
5.8.2 Sensitivity to Motion 28
5.9 Strengths and Weaknesses 28

6 Template Matching **30**
6.1 Motion Vector 30
6.2 Training Stage 32
6.3 Tracking Stage 34
6.4 Determining Optimal Parameters 34
6.4.1 Number of Training Examples 34
6.4.2 Number of Tracking Points 35
6.5 Oscillations 36
6.6 System Evaluation 37
6.6.1 Sensitivity to Noise 37
6.6.2 Sensitivity to Motion 38
6.7 Strengths and Weaknesses 39
6.8 Improving the Template Tracker 39

7 Three-Dimensional Data From Two-Dimensional Tracking **41**
7.1 Implementation Notes 43

8 Model-Based Tracking **45**
8.1 Camera Model 45
8.2 Model Rendering 46
8.3 Locating Edges 47
8.4 Computing Motion 50

8.5 Stabilization . . . 52
8.6 Tracking Performance . . . 53
8.6.1 Small Displacement Assumption . . . 53
8.6.2 Lag . . . 53
8.6.3 Sensitivity to Noise . . . 54
8.7 Strengths and Weaknesses . . . 57

9 Simulator Tests **59**
9.1 Setup Procedure . . . 59
9.1.1 Camera Calibration . . . 60
9.1.2 Sensor Planning . . . 61
9.2 Expected Tracking Accuracy . . . 65
9.2.1 Fiducial Detector . . . 66
9.2.2 Conversion between 2D and 3D coordinates . . . 69
9.3 Tracking Tests . . . 71
9.4 Concluding Remarks . . . 71

10 Results **73**
10.1 Experimental Setup . . . 73
10.2 Tests . . . 73
10.2.1 Qualitative Test . . . 74
10.2.2 Quantitative Test . . . 75

11 Conclusion **80**

References **82**

12 Appendix **86**
12.1 Linear Kalman Filter Implementation . . . 86
12.2 Table of Measurements . . . 89

ABSTRACT

The purpose of this thesis is to determine the applicability of computer vision tracking as a source of feed-back in the development of an intelligent computer numerical (CNC) controller potentially capable of detecting problems and eventually correcting them. For this task, three types of visual tracking methods are quantitatively evaluated to determine which approach is more suited to the development of the visual tracker. Emphasis will also be placed on camera calibration.

The three classes of visual methods chosen are a marker-based detector, a template matching algorithm, and a model-based tracker. From these, it is found that the marker-based detector is the most accurate for the CNC tracking task by providing sub-pixel accuracy and robustness to visual contaminants such as noise.

A visual simulator is developed to provide a fully controllable testing environment to determine optimal system parameters. The simulator also provides precise ground-truth used to quantify the tracking error and obtain an accuracy baseline before applying the tracker on real data. The tracking algorithm is then applied to image sequences of a physical machine to evaluate the real performance of the system. The accuracy of the system is found to be limited mostly by image resolution.

RÉSUMÉ

Le but du présent mémoire est de déterminer si la vision par ordinateur peut être utilisée comme source d'information, afin de réaliser un contrôleur intelligent pour une machine contrôlée numériquement. Un tel contrôleur pourrait être capable de détecter les problèmes et ultérieurement être capable de les corriger. Pour cette tâche, trois types de traqueurs visuels sont évalués quantitativement afin de déterminer quelle approche est la plus adéquate. L'accent sera également mis sur la calibration de la caméra.

Les trois classes de méthodes visuelles choisies sont: un détecteur de marqueurs, un algorithme de correspondance de gabarit et un traqueur de modèle. À partir de ces méthodes, il est constaté que le détecteur basé sur les marqueurs est le plus précis pour la tâche qui est de suivre une machine opérée par commande numérique, car il fournit une précision sous-pixel et est robuste aux contaminants visuels tel que le bruit.

Un simulateur visuel est conçu pour fournir un environnement de test entièrement contrôlable afin de déterminer les paramètres optimaux du système. Le simulateur fournit aussi des données précises utilisées pour quantifier l'erreur du traqueur et obtenir un niveau de référence de précision avant d'appliquer le traqueur sur des données réelles. Le traqueur est ensuite appliqué à des séquences d'images d'une machine physique pour évaluer la performance réelle du système. La précision du système se trouve être limitée principalement par la résolution d'image.

ACKNOWLEDGMENTS

I would first like to thank Professor Ferrie, who has given me the opportunity to work on this project and who has inspired me to further my knowledge of the computer vision field. This master's thesis would not have been possible without Professors Arbel, Levine, and Siddiqi, who have shared their passion of computer vision with me through their courses.

The support of my family has been invaluable to the completion of this thesis and for that, I am infinitely grateful.

I kindly thank all my friends in the Artificial Perception Lab who made working on this thesis an enjoyable experience. I would also like to thank Frida Ceja-Gomez for checking my math in times of uncertainty.

I finally have to express my gratitude to everyone involved in the open-source community, especially those working on the following projects: GCC, LaTeX, and gnuplot.

1 Introduction

1.1 Motivation

The motivation behind the current thesis can be separated in two distinct parts. The first concerns how low-precision CNC machining can be improved through visual feedback using off-the-shelf sensors, and the second concerns how to choose visual tracking methods for a specific case application.

Fully automated manufacturing machines have been an important topic of science fiction for many years. A good example of this is the Star Trek replicators capable of manufacturing many different types of objects ranging from ship parts to prepared meals. The current state of automated manufacturing is not even close to the futuristic visions of science fiction writers, but the potential applications of fabrication machines are almost limitless.

Over the past few years, many consumer-grade CNC machines and 3D printers have become available. These machines are obviously not on par with their more expensive commercial and industrial counterparts in terms of quality and precision. This presents new problems such as finding ways to keep the price of machines low while obtaining the best accuracy possible.

The current thesis addresses this problem by investigating the possibility of using computer vision as a source of feedback for low-end CNC machines.

1.2 Background

Fairly recent papers by Xu and Newman [1],[2] provide a good overview of the modern state of automated manufacturing processes. According to them, most problems with the currently widely-used CNC programming language known as G-code arise from the lack of high-level information and the machine-specific properties of the language. An on-going effort to solve most of the problems of G-code is the development of a new format known as STEP-NC. This format allows "smart" CNC controllers to use high-level information to control the machines. For this reason, STEP-NC does not define how controllers should be implemented, but only gives a description of the final expected result. Figure 1 depicts the block diagram of a "smart" controller that would use visual information as a source of feedback.

Obviously, this controller relies heavily on feedback from the vision system, whose main task is to track the current physical state of the machine. Recent

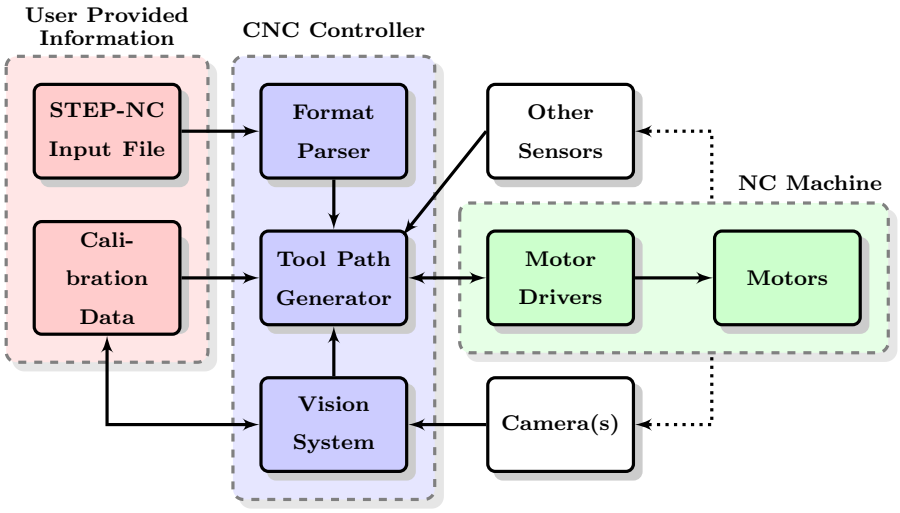


Figure 1: Block diagram of a smart CNC machine controller

work on CNC machine tracking has been done ([3]) with the goal of implementing an augmented reality system capable of displaying real-time information about the machine to a human operator. This research showed promise but did not aim to find the best tracking method for the tracking task.

1.3 Goals

The purpose of this thesis is to assess the viability of computer vision in the field of automated machining. This will be done by exploring various three-dimensional object tracking approaches to determine which is more suited to the development of an "intelligent" CNC controller. Emphasis will be placed on the tracking accuracy, the ability to run in real-time, and how well the methods cope with noise and rapid motion.

1.4 System Requirements

- **Real-time**
  For any practical application, the tracking system needs to run at the same time as the machine and thus the selected methods need to be efficient enough to process data in real-time.

- **Precision**

As with any machining tasks, the required precision is dependent on the type of part being machined. For example, surface-mount printed circuit board routing requires more precision than through-hole printed circuit board routing.

1.5 Assumptions

- **Full Environment Control**
  For the purpose of this thesis, it will be assumed that the environment in which the machine resides can be fully controlled. Some of the critical aspects that may be controlled include the ability to augment the machine with markers, control the position of the light sources and cameras in the scene. This means that the tracking method does not necessarily need to be robust to different scene configurations such as changing shadows.

- **Machine Configuration**
  The current thesis will concentrate on machines with a three-axis configuration, commonly referred to as 2.5-dimension machines. While the method selected needs only be able to track such machines, thought will be given to tracking machines with a greater number of degrees of freedom.

- **Rigid Objects**
  It will be assumed that all of the tracked objects are rigid, there will be no visible object deformation.

- **Approximate knowledge of the object position**
  As the goal of this project is to track a CNC machine while it is running, the controller of the machine may be able to provide an approximation of the tool position. This may be useful when initializing some tracking algorithms.

1.6 Potential Problems

As mentioned in [3], practical applications will often include visual "contaminants" like dust and coolant which may occlude or change the appearance of some features of the scene.

1.7 Potential Applications

At the most basic level, a system capable of accurately tracking a CNC machine can be used for simple tasks such as initializing a machine to a given home position after a system reset or between two programs. Faster initialization and re-initialization can greatly reduce production downtime and minimize the need for human operator intervention. An efficient tracking system is also an integral part of the development of an "intelligent" controller capable of detecting and correcting errors.

1.8 General Goals

By the end of this thesis, the following questions will be answered.

- Which type of method provides the most accurate tracking results?
- Which type of method is the most computationally efficient?
- How does camera resolution influence tracking accuracy?
- What accuracy should be expected from such a system?

1.9 Thesis Overview

The thesis will start with a short literature review in Section 2. Section 3 will then give an overview of the testing platform that will be used throughout the thesis, while Section 4 will present the camera calibration algorithm implemented. The next sections will then be used to demonstrate the implementation of the three candidate trackers, in the following order: Section 5 will introduce a marker-based 2D tracker, Section 6 will present a tracker based on 2D template matching, and Section 8 will show a model-based 3D tracker. Each of these sections will characterize the different trackers by applying controlled tests to determine how they react to noise and motion.

In Section 9, the tracker that has been found to be more suited to the task of tracking a CNC machine will be tested on a simulated machine. The simulator will first be used as a sensor planning tool to determine the optimal position of the camera. The simulator will then be used to predict the maximum theoretical accuracy of the overall system. Finally, an estimate of the expected practical accuracy of the tracker will be determined.

The last section of the thesis will apply the tracking system to an image sequence from a real machine to determine the applicability of the system to real world implementation.

2 Literature Review

The basic goal being the tracking of a structure moving through three dimensional space, most of the related literature will come from the general field of object tracking. As there is a wealth of literature discussing this topic, it is important to find information pertinent to the current topic by referring to the requirements and assumptions detailed earlier and by finding papers dealing with similar problems. The two closest areas of research are visual servoing and augmented reality.

Visual servoing aims to create a feedback loop based on visual information to control robot actuators. This is very similar to the idea of the smart CNC machine controller discussed earlier. Visual servoing methods are separated into two main parts: visual tracking, and motion control. In the context of the current thesis, only the tracking aspect will be investigated.

Augmented reality systems usually involve tracking the state of the camera relative to the real world in order to overlay information over camera images. Tracking the position of the camera is the same operation as tracking an object's position with respect to the camera.

Short literature reviews dealing with each of the trackers implemented in the context of this thesis will be presented at the begining of each of their respective sections.

For both visual servoing and augmented reality tracking systems, the tracking can usually be categorized in one of two categories [4], position-based, or image-based. The first type of method involves extracting the pose of the tracked object by using a three dimensional geometric model of the object, while the latter only uses features from the two dimensional camera image. The next two subsections will summarize a few existing methods from both types of approach.

2.1 Position-Based Tracking

Position-based tracking involves the use of three-dimensional models. Existing methods in this field present various degrees of complexity depending on the type of object to be tracked.

One approach is to track polyhedral objects that have a known model. This model can be obtained directly from the Computer Aided Design (CAD) file of the part being tracked. Such an application is found in [5].

Models composed of different parts can also be tracked by using a method such as what is presented in [6]. This is closer to the problem being investigated by this thesis as a CNC machine can be defined as a collection of four models that have known relations. The underlying tracking method is very similar to [5] and [7]. This type of tracker will be explored in more details in Section 8 by implementing the algorithm presented in [7].

More complex model-based trackers can use deformable models to track non-rigid objects such as human body parts. This sub-area of research is not considered in the current thesis as no deformable objects are to be tracked.

2.2 Image-Based Tracking

The papers mentioned here are chosen to show both the types of image features that can be used and the variety of situations in which image-based visual servoing can be applied.

A paper by a fellow center for intelligent machine researcher [8] details a servoing method applied to an aquatic robot following a colored ball detected using a region of interest detector. No formal study of the tracker's accuracy is provided in this particular paper, but examples provided show that the region of interest only provides a rough estimate of the center of the ball.

A similar application of servoing is detailed in [9], where a sum of square difference (SSD) template matching is used to locate a known marker in a camera image. The position of this marker is then used to compute the position of the robot.

Another interesting application of visual servoing is the control of a welding robot based on the tracking of straight lines [10]. As stated in the paper, lines are robust image features that can be extracted in many different ways.

As can be seen from these few papers, any image feature which can be reliably extracted is a good candidate for an image-based tracking system.

3 Testing Platform

This section will describe the tools used to evaluate the different methods.

3.1 Hardware Platform

The specific characteristics of the system are not important other than to provide a general idea of the context in which measurements were made. The test system is a six-core AMD Phenom II processor clocked at 3.30 Ghz. The other noteworthy part in the test system is an NVidia GTX 560 video card, which supports CUDA parallel computing. Execution times shown in this thesis are used as a metric to compare different algorithms; they are not meant as an absolute measure of an algorithm's performance, as execution time would be different on other sytems.

3.2 Software Platform

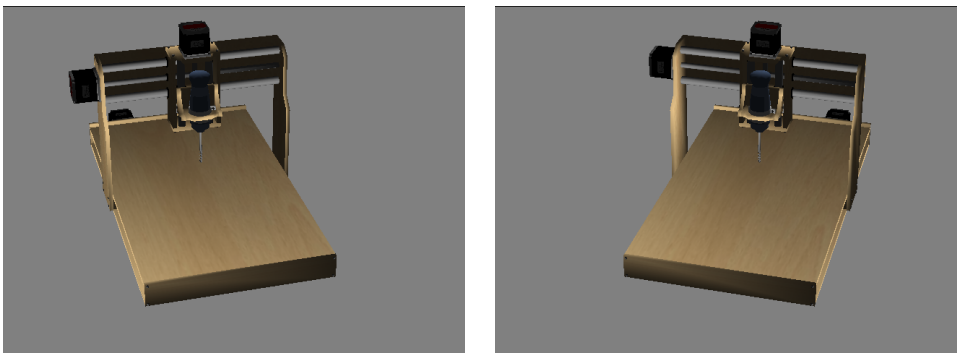
All algorithms implemented for this thesis are written in C/C++. Many computer vision libraries such as the widely used OpenCV exist, but it was decided that these would not be used. The reason for this is more than purely educational, full transparency of the underlying structure is required to properly compare the different algorithms. The goal of this thesis is not only to evaluate tracking algorithms, but also to learn about the inner workings of computer vision. Using high level libraries and tools such as Octave or MATLAB is convenient, but it shields the developers from the implementation details, which are often important to understanding the intricacies of the algorithm being implemented. Writing the low level functions also allows for better optimization based on the specific situations being encountered. This gives a greater understanding of the characteristics and limitations associated with each component of the system.

Even if no computer vision library is used, there are obviously many software libraries used to implement large scale algorithms. Video Capture is done through the Video for Linux version 2 (V4L2) application programming interface (API). The portable network graphics library (libpng) is used to read and write image files. Most of the more complex linear algebra problems are handled through the Armadillo library, chosen because of its reliance on the linear algebra package (LAPACK), which in turn relies on the Basic Linear

Algebra Subprograms (BLAS) to achieve very high performance. Finally, the Simple DirectMedia Layer (SDL) and the OpenGL libraries are used to display images, render three dimensional scenes and handle user inputs.

3.3 Machine Simulator

To be able to compare the different methods in a fair manner, they need to be tested on the same datasets. As the methods studied may require different scene configurations, it would not be technically feasible to capture videos from a physical machine for the testing stage. For this reason, a machine "simulator" program was devised. This program uses OpenGL to render a virtual machine. Figure 2 shows a stereo pair of images of the machine rendered by the "simulator". None of the tracking methods implemented for this thesis rely on stereo imagery, but it is interesting to note that the simulator is flexible. Strictly speaking, this program is a visual simulator and not a physical simulator, as no physics simulation is performed.



(a) Left Image (b) Right Image

Figure 2: Stereo pair of images generated by the simulator

The second reason for using a simulator is that to properly evaluate the tracking methods, a set of ground truth measurements is required. This ground truth is very hard to obtain accurately in real life and would probably require human annotation of an image sequence. The simulator is capable of outputting very precise data regarding the state of the machine for each frame of the sequence.

3.4 Real Machine

For the final testing stage, a real CNC machine will be used to measure the performance of the algorithm. The physical setup used will be described in more detail in Section 10.

4 Camera Calibration

Camera calibration is an important step in most object tracking methods as the accuracy of the tracking is largely dependent on the precision of the camera calibration. Most of the information for this section of the thesis comes from [11], which presents a camera calibration method requiring a single pattern placed on a planar surface. This section will detail the calibration method and its implementation. More recent calibration methods such as [3] were considered, but the approach of [11] was determined to be adequate for the present thesis.

The first calibration pattern chosen was very similar to the one used in the original paper. A scaled version of the pattern is shown in Figure 4.

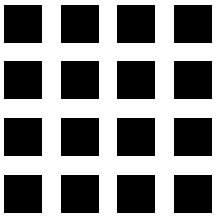
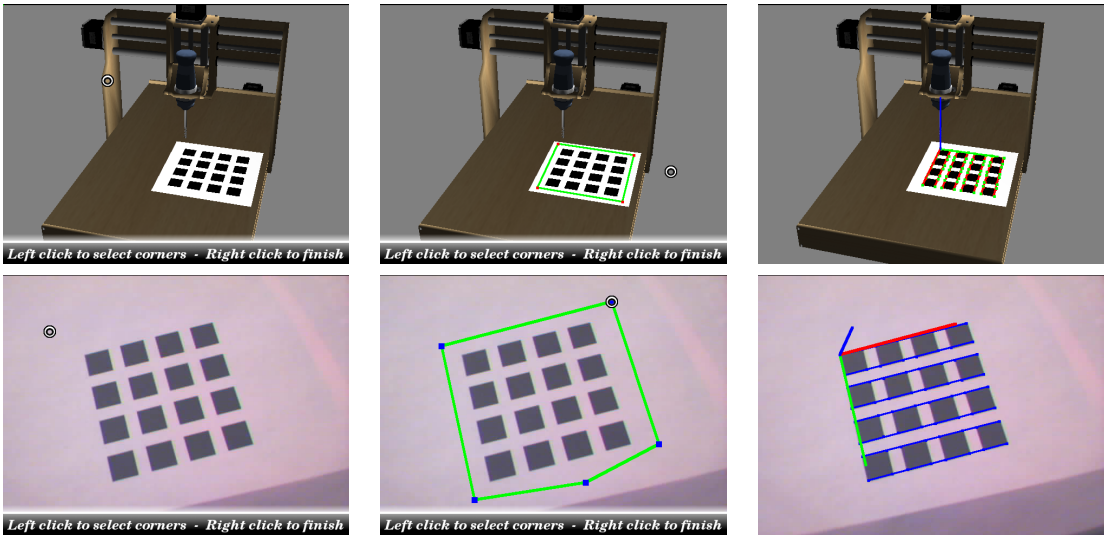


Figure 3: Planar pattern used for the camera calibration (1:2 scale)

The calibration procedure starts by prompting the user to input a polygonal crop region to restrict the search area of the detector. This first step is not strictly necessary, but provides a considerable improvement in calibration speed. To simplify the corner detection further, the image region is binarized. The threshold value is determined using Otsu's method [12]. This method assumes that the image is composed of two types of pixels, foreground and background. This is exactly the situation we are presented with as the calibration pattern is made of black squares over a white background. Once the corners are extracted, the program applies the calibration algorithm presented in [11]. The optimization stage relies on a Levenberg-Marquardt least-squares minimization algorithm. Figure 4 shows the three step calibration procedure for both a real camera and a simulated OpenGL camera.

The use of this pattern causes two problems. First, the fact that the pattern is symmetrical makes it impossible to acquire information about its orientation, and thus, the extrinsic parameters of the camera are difficult to compute. The second problem is caused by the use of a FAST corner detector [13],[14]. This detector, as its name implies, is fast but not accurate enough



(a) (b) (c)

Figure 4: The three steps required to calibrate a camera. An image generated by the simulator and an image captured by a webcam are shown in (a). The user-defined crop region is shown in (b). In (c), the calibration results are used to draw the three axes of an X-Y-Z coordinate system in red, green, and blue, respectively.

for a good camera calibration. Figure 5 shows the problems related to the use of this detector. As can be seen in 5(b), the corners are detected inside of the dark region. Some pruning can be applied to the output of the FAST detector to get a result similar to what is shown in 5(c). It can be seen that the detected corner is still located half a pixel from the real corner, shown in 5(d).

Because of these problems, the calibration pattern was changed to what is shown in Figure 6. With this new pattern, the circle detector that will be presented in Section 5 can be applied to find the center of the circles to sub-pixel accuracy. Because of the missing circle in the top right corner of the pattern, the origin of the coordinate system can always be positioned on the circle in the opposite corner. User intervention is also not required as the detector can process the entire image with little to no misclassification.

By estimating the camera parameters over a few frames, it was noticed that the non-linear optimization stage returns slightly different results for each frame. This is probably due to measurement noise. Figure 7 shows the estimated focal length in red over one thousand frames. The estimated value fluctuates between 10.5 and 12.5. Only the estimated focal length is shown,

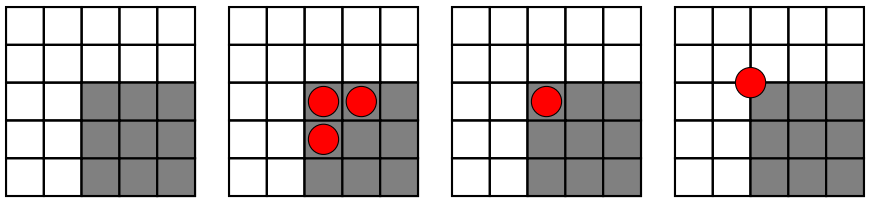


Figure 5: Illustration of the problem related to the use of FAST corner detectors. In (a), a corner is shown. The predicted output of a FAST detector is shown in (b). The results of (b) can be pruned to get a better idea of the location of the corner (c). Finally, (c) shows where the corner should be detected.

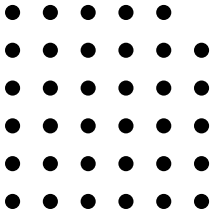


Figure 6: Second calibration pattern used. (1:4 scale)

but similar fluctuations are found with the other camera parameters as well. For this reason, the output of the estimation was fed into a Kalman filter to try to reduce the amount of noise. For the implementation details of the Kalman filter used, please refer to Section 12.1. The filters were initialized with $Q_{i,i} = 0.0001$ and $R_{i,i} = 0.5$, meaning a very small amount of process noise and a large amount of measurement noise. The output of the Kalman filter for the focal length is superimposed over the estimation values in Figure 7 (in blue). By looking at the plot, it is clear that the filtered values seem to be closer to a constant value than the measurements. The final value of the focal length is taken to be the average of the filtered values.

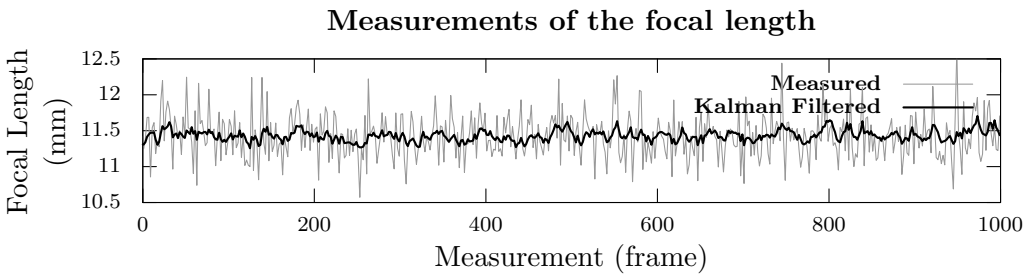


Figure 7: Plot of the measured focal length of a camera over a period of 1000 frames (red) and the output of a Kalman filter (blue).

5 Marker-Based Detector

5.1 Background

Marker-based algorithms require one or more visual markers to be added to the tracked object. Marker detection has been extensively used by augmented reality, medical vision systems and by point of sale systems to identify products in a way similar to bar-codes. The marker designs vary between the various methods that have been proposed over time. Some of these methods use quadrilateral markers [15],[16] providing strong edges that can easily be extracted from images using standard image processing methods. Some use single circular markers composed of concentric rings [17],[18]. Other methods, like the one that will be detailed in this section, use a set of easily detectable points with a known geometric relation.

According to Lepetit and Fua [19], circular markers can be detected to an accuracy of 1/10 pixel in most cases and an accuracy of 1/100 pixel is possible when controlling scene conditions. The fiducial detection method that was implemented for this thesis comes from [20] because it uses circular markers and is reported to be very robust.

5.2 Overview

The fiducial marker detection algorithm proposed in [20] can be summarized by the block diagram shown in Figure 8. The input image is the image where fiducial markers are to be found. The list of markers is a possibly empty list containing information about the detected markers. Each of the other blocks will be described in the following subsections.

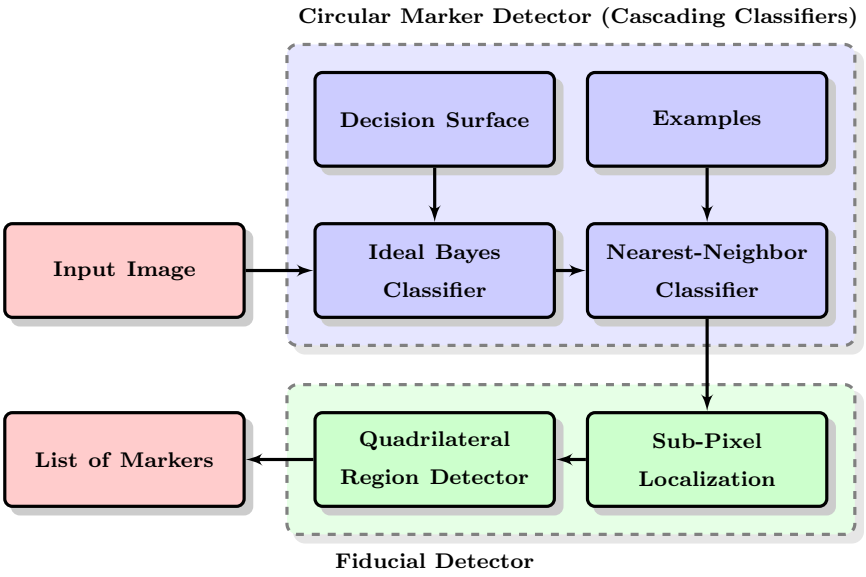


Figure 8: Block diagram of the methodology behind the fiducial marker detector

5.3 The Markers

The markers used are fairly simple in design, they are created by placing four circles on the vertices of a square. For the purpose of this thesis, the side of the square was set to 10 centimetres and the radius of the circles was chosen to be 8 millimetres. These choices yield markers that look similar to those presented in the original paper [20]. To be able to distinguish between multiple markers, a three by three matrix capable of storing identification information is added within the space enclosed by the four circles. This method of adding data to the marker is presented in [21]. Figure 9 shows the position of the four circles and the position of the identification data.

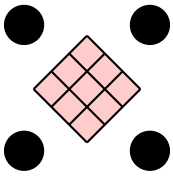


Figure 9: Fiducial marker with pink region representing the area available to store information (1:3 scale)

Even if the matrix used to store information is three by three, it is not possible to use all 512 possible combinations with this marker. This is due to

the fact that there is no way to compute the orientation of a marker. Thus, it is impossible to distinguish between two markers if it is possible to generate the second by rotating the first. This situation can be seen in Figure 10 (a and b). There are many ways to deal with this problem, the simplest one being to restrict the set of markers to those that are not affected by the problem. A more elegant way to handle the orientation problem is to add a fifth circle on one of the sides of the square to denote the top of the data matrix, as shown in Figure 10 (c). While this allows the use of all 512 combinations, there are still potential problems. If the reflection of a marker on a mirror is observed, two different markers would be confused. This last problem can be solved by moving the orientation circle to a side of the marker, as seen in Figure 10 (d).

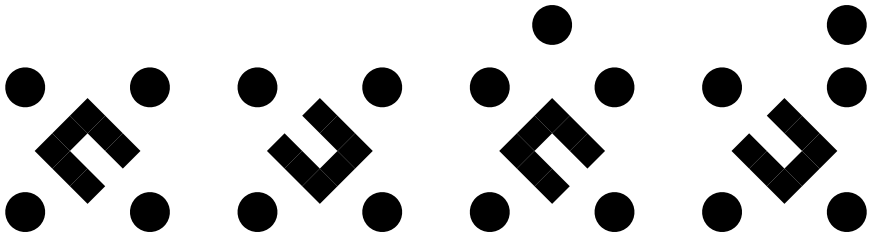


Figure 10: Two markers that are indistinguishable (a and b) and markers augmented with orientation circles (c and d) (1:3 scale)

For the purpose of tracking a CNC machine, a minimum of one marker is required, and it might be interesting to place a different marker on each of the different parts of the machine to track them separately. For this reason, simply choosing markers that cannot be confused is appropriate and thus, no orientation circle will be necessary for the current project. Once the black circles are found in the image, fiducial markers can be detected by a simple geometric check, as done in [20].

5.4 The Examples

The two classifiers that determine whether an image patch is a marker or not require a set of positive and negative examples. Obviously, the classification accuracy is highly dependent on the initial examples and thus, care must be taken when gathering examples. A few of these examples are shown in Figure 11. All examples used are 12 pixels by 12 pixels.

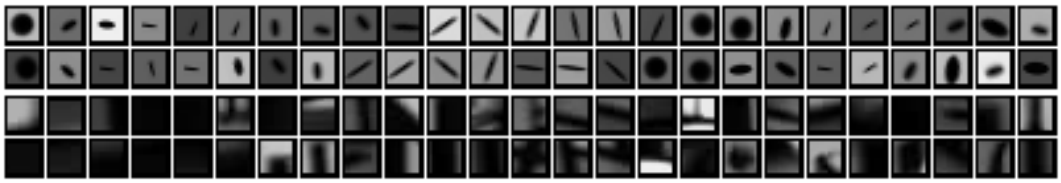


Figure 11: A subset of the examples used for the two classifiers. The first two rows are positive examples and the last two rows are negative

The negative examples are generated by extracting all 12 by 12 sub-windows of a few images not containing any markers. The positive examples are created by manually choosing image patches containing valid circular markers from a set of training images. As can be seen in Figure 11, examples representing extreme distortion and lighting conditions are included to make the classifiers more robust to changes in perspective and illumination of the scene. Robustness, in the current context, refers to the ability of the system to successfully etect the marker when presented with different scene configurations. To increase the robustness further, a larger set of examples is generated from those manually chosen to simulate more varied conditions. The images are smoothed by convolving them with a Gaussian kernel of varying standard deviation to simulate slightly out of focus circles. Different brightness and contrast settings are also applied to the examples to simulate a wide range of lighting conditions. Finally, the images are rotated to simulate camera rotation.

The detector proposed in the original paper [20] is not directly robust to scale variation. The detection of varying circle sizes is achieved by successive application of the detector on scaled versions of the input image. Scale variation could also be handled by using a larger window size and presenting the training algorithm with examples containing black circles of different sizes. Using a large window size would slow the algorithm down in the nearest neighbor detection stage that will be detailed in Section 5.6. Thus, both methods of handling scale variation reduce the performance of the system. For the tests in this section, the diameter of the circles will be restricted to approximately 10 pixels to eliminate the scale variation problem. For the final implementation described in Sections 9 and 10, a larger sub-window size will be used and the detector will be trained with example circles of different sizes.

5.5 Ideal Bayes Classifier

The goal of the ideal Bayes classifier is to quickly find image patches that may contain a circular marker. It is used mostly as a means of reducing the number of sub-windows that will need to be evaluated by the nearest-neighbour classifier.

This classifier is based on the heuristic that if an image patch contains a circular marker, the pixels on the outer edge should be lighter than those in the center. By considering pairs of pixels containing one of the center points and one of the 8 pixels along the borders, two distributions D^- and D^+ can be built from the negative and positive examples, respectively. The two distributions are shown in Figure 12. It must be noted that the two distributions have been smoothed by convolving them with a Gaussian kernel of small standard deviation to remove potential abnormalities and create more uniform distributions.

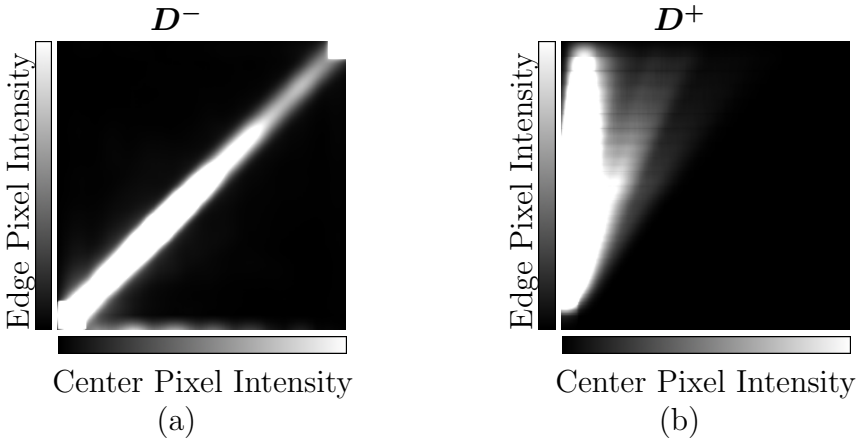


Figure 12: Distribution of pixel pairs (a) from the negative examples and (b) from the positive examples. The horizontal axis represents the intensity of the center pixel and the vertical axis represents the intensity of the edge pixel.

$$\text{classification}(I_c, I_e) = \begin{cases} +1 & \text{if } \alpha \cdot D^+(I_c, I_e) > D^-(I_c, I_e) \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

Given a pair of pixels (I_c, I_e), where I_c is the intensity of the center point and I_e is the intensity of the edge point, Equation (1) can be used to determine whether the pair comes from a window containing a circular marker $(+1)$ or not (-1). The α parameter represents the "relative cost of a false negative

over a false positive"[20].

By varying the value of α, it is possible to generate a receiver operating characteristic (ROC) curve as shown in Figure 13.

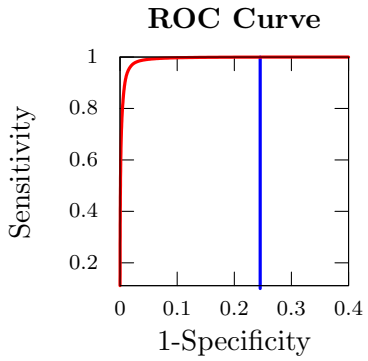


Figure 13: ROC curve generated by varying α.

As stated previously, the goal of the Bayes classifier is to determine which image patches should be considered by the next classifier. For this reason, false negatives may cause severe problems in later stages, but false positive should not be a concern. This means that α needs to be chosen where the sensitivity is very high. It is obvious that a high specificity is preferable, but not necessarily required at this point. By using the ROC curve, it was found that $\alpha = 0.274373$, associated with a sensitivity of 0.967120 and a specificity of 0.980106, generates a good decision surface, shown in Figure 14. Again, the decision surface is smoothed and thresholded to make the boundary smoother.

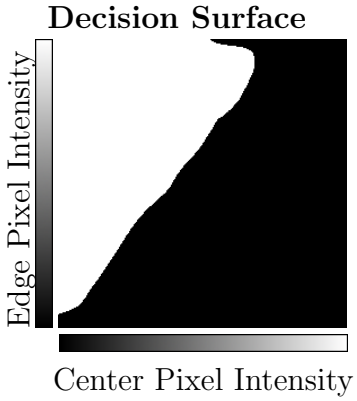


Figure 14: Resulting Decision Surface.

$$\text{classification}(I_c, I_e) = \begin{cases} +1 & \text{if } (I_c, I_e) \text{ lies on a white region of} \\ & \text{the decision surface} \\ -1 & \text{otherwise} \end{cases} \quad (2)$$

The beauty of this method is that once the decision surface is obtained, it can be saved and reloaded whenever necessary. The decision surface can then be used to quickly classify new pixel pairs (I_c, I_e) using the classification rule in Equation (2). The decision surface is effectively used as a look-up table.

5.6 Nearest-Neighbour Classifier

This classifier is conceptually quite simple and robust, but is not computationally efficient. That is why it is only applied to regions detected by the Bayes classifier.

In order to find a nearest neighbour, a distance metric must be defined. If the 12×12 image region being evaluated and the examples are reshaped to a 144-dimensional vector, then the distance between an image patch and an example is induced by the Euclidean vector norm $|| \cdot ||$.

$$\text{classification}(x) = -\text{sign}(\min_i ||p_i - x||^2 - \min_j ||n_j - x||^2) \quad (3)$$

Given an image patch, applying the classifier involves finding the closest example from each of the positive and negative example set. From that, the binary classification of a 12×12 image region (x) can be found by using Equation (3), reproduced from [20], where p and n are the positive and negative example sets, respectively.

5.6.1 Reducing the Number of Examples

The execution time of the nearest neighbour classifier is directly proportional to the number and dimension of examples that have to be compared with the image patch being classified. Thus, the best way to decrease the execution time required to run the classifier is to either decreasing the number of examples or reduce the dimentionality of the examples. Dimensionality reduction refers to either using smaller subwindows, thus making the vectors representing the subwindows smaller or using methods to encode the information of the

subwindows in smaller vectors. Methods such as principal component analysis could be used for dimensionality reduction. To implement the same method as presented in [20], the number of examples will be reduced. To determine which of the examples to remove and which to keep, the condensed nearest neighbour rule presented in [22] is used.

The logic behind the condensed nearest neighbour rule is to find a subset of the examples that can properly classify the remaining examples using the nearest neighbour classification discussed previously. This subset is called a consistent subset of the example set.

The condensed nearest neighbour algorithm uses two bins to separate the redundant examples from the ones part of the consistent subset. The first bin is called the store and will contain the consistent subset once the condensation is complete. The second is called the grab-bag and contains the other examples. The algorithm begins by placing all examples in the grab-bag. Then, the first positive and negative examples are transferred to the store. At each iteration of the algorithm, the examples of the store are used to classify the examples in the grab-bag. If an example from the grab-bag cannot be properly classified as positive or negative, it is added to the store. The algorithm continues until all the examples in the grab-bag can be classified correctly by the examples in the store. At this point, the examples in the grab-bag can be discarded.

As stated in [20], manually choosing a positive and negative initial example yields smaller final subsets. The number of negative training examples was reduced from 34916 to 60, and the number of positive examples reduced from 990 to 54. This smaller number of examples makes it possible to run the nearest neighbour in a shorter amount of time.

5.7 Sub-Pixel Sampling

$$\text{likelihood}(x) = \frac{\min_j ||n_j - x||^2}{\min_i ||p_i - x||^2} \tag{4}$$

In order to find the coordinates of the circular markers to sub-pixel precision, the nearest neighbour classifier is modified to return the likelihood of a marker being inside the considered window instead of a binary classification. This likelihood is computed by using Equation (4).

The results from the nearest neighbour classifier are labelled using a sequential labelling algorithm [23] to find connected regions representing individual circles. This labelling can be used to segment the image into "windows", where

sub-pixel sampling strategies can be applied. The original marker detection method makes use of a non-maxima suppression to determine the center of the circles, but other methods presented in [24] will now be implemented in Sections 5.7.1 through 5.7.7 in order to determine which is more accurate.

5.7.1 Maximum Intensity Pixel (MIP)

This is the simplest method; it involves finding the pixel with the highest likelihood and selecting it as the center of the circle. In most cases, MIP does not find the center of the circle to sub-pixel accuracy. The only situation in which such precision can be obtained is when more than one pixel have the same likelihood and that this likelihood is the maximum. In that case, MIP returns the average of the positions of the maximum points.

5.7.2 Non Maxima Supression (NMS)

The non maxima suppression strategy is similar to MIP, but returns more than one point if multiple local maxima exist in the image. In the same way as MIP, NMS does not usually offer sub-pixel accuracy.

5.7.3 Intensity Barycentric Weighting (IBW)

$$C_x = \frac{\sum_{i=0}^{n} m_i \, x_i}{\sum_{i=0}^{n} m_i} \quad , \quad C_y = \frac{\sum_{i=0}^{n} m_i \, y_i}{\sum_{i=0}^{n} m_i} \tag{5}$$

This algorithm is based on the physical concept of center of mass. Given an object defined by a set of n particles, the physical center of mass of this object is determined with Equation (5), where C is the center of mass, m_i is the mass of the ith particle, and $\mathbf{p}_i = (x_i, y_i)$ is the position of the ith particle.

The same equation can be used by considering the circle as an object composed of n pixels and replacing the mass by the likelihood of each pixel.

5.7.4 Intensity Linear Interpolation (ILI)

This algorithm starts by finding the location of the MIP of the image. Once found, the row containing the MIP is extracted. This row can be viewed as a set of measurements taken from a 2D curve. Figure 15 shows an example of such a row with black square markers. As stated in the name, linear interpolation (green lines) is used to connect the points. The method continues by finding the intersections (black lines) between the interpolated curve and an imaginary

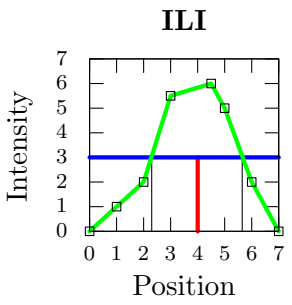


Figure 15: Reproduction of a figure from [24] to demonstrate how the ILI method is applied.

horizontal line (blue line) at half the intensity of the MIP. The midpoint of the line connecting those intersections is used as the sub-pixel location of the center in the x direction. This process is applied again for the y direction by extracting the column containing the MIP.

5.7.5 Intensity Quadratic Fitting (IQF)

$$f(x) = a_4x^4 + a_2x^2 + a_0 \quad , \quad f'(x) = 4a_4x^3 + 2a_2x \tag{6}$$

The IQF strategy begins by extracting the row or the column containing the MIP in the same way as for the previous method. To determine the location of the center, a quartic function of the form shown in (6) is fitted to the extracted points using a non-linear least squares technique.

Once the Gaussian curve is found, it is possible to determine the location of the center of the circular marker by finding the maximum of the quartic function. This is done by finding the positive zero-crossing of the derivative of the curve. As with the previous method, the technique needs to be applied independently for the x and y directions.

5.7.6 1D Intensity Gaussian Fitting (1DIGF)

This is the same technique as IQF, but instead of fitting a bi-quadratic curve, a Gaussian curve is fit to the data. The center is then simply taken as the mean of the Gaussian curve.

5.7.7 2D Intensity Gaussian Fitting (2DIGF)

This final method is an implicit function fitting problem using a 2D Gaussian surface. A Levenberg-Marquardt least-squares surface fitting algorithm is used. Again, the means in the x and y directions of the Gaussian surface are taken as the center of the circular marker.

5.7.8 Comparison of the Methods

In order to compare the accuracy of the seven methods, a set of matrices containing truncated 2D Gaussian distributions was used. This was used to simulate the distribution of the likelihoods obtained from the nearest neighbour classifier. The standard deviations in the x and y directions of the distributions were varied to simulate a circle being observed from different perspectives. The center of the circles was moved by changing the means of the Gaussian distribution.

Table 1 contains various statistics about the error between the real center and the center reported by each of the methods. A large number (14400) of sample matrices were used to generate the table. By looking at the table, IQF can easily be discarded because its mean error is simply unacceptable. This is probably due to the fact that a bi-quadratic curve does not represent the data well. MIP and NMS can also be discarded as their standard deviations are both high.

Algorithm	Min	Max	Std.Dev.	Mean	RMS	Runtime(ms)
MIP	0.00	0.67	0.14	0.35	0.38	0.02
NMS	0.00	0.67	0.14	0.35	0.38	0.26
IBW	0.00	0.47	0.09	0.15	0.18	0.02
ILI	0.00	0.01	0.00	0.00	0.00	0.01
IQF	2.75	3.73	0.20	3.37	3.38	0.11
1DIGF	0.00	0.04	0.01	0.01	0.01	0.35
2DIGF	0.00	0.00	0.00	0.00	0.00	9.95

Table 1: Comparison of the error produced by the various sub-pixel localization methods applied to noise-free images.

The results in Table 1 show how the methods fare when presented with noise-free images. Table 2 was generated from 43200 sample images containing various amount of additive noise. Compared to the previous table, the errors are noticeably larger. From this table it can be seen that 2DIGF does not

cope well with noisy measurements. ILI also seems to have some problems with noise.

Algorithm	Min	Max	Std.Dev.	Mean	RMS	Runtime(ms)
MIP	0.00	5.12	0.78	1.07	1.33	0.02
NMS	0.00	3.62	0.61	0.90	1.09	0.37
IBW	0.00	5.43	0.50	0.52	0.72	0.02
ILI	0.00	12.68	4.07	4.81	6.30	0.01
IQF	0.05	158.81	2.88	3.57	4.59	0.08
1DIGF	0.00	49.74	1.08	0.79	1.34	0.44
2DIGF	0.00	1012.80	20.26	9.04	22.18	24.52

Table 2: Comparison of the error produced by the various sub-pixel localization methods applied to noisy images.

Three potential methods are now left: IBW, ILI, and 1DIGF. To determine the best one to use, they were run with a varying amount of noise. The results of this test are shown in Figure 16. From this test, it can be seen that ILI is very sensitive to noise. IBW and IGF both seem to be influenced by noise in the same way, however, 1DIGF seems to be a little less stable as seen from the spikes. For this reason and because IBW runs roughly 19 times faster than 1DIGF, IBW is chosen as the sub-pixel sampling method.

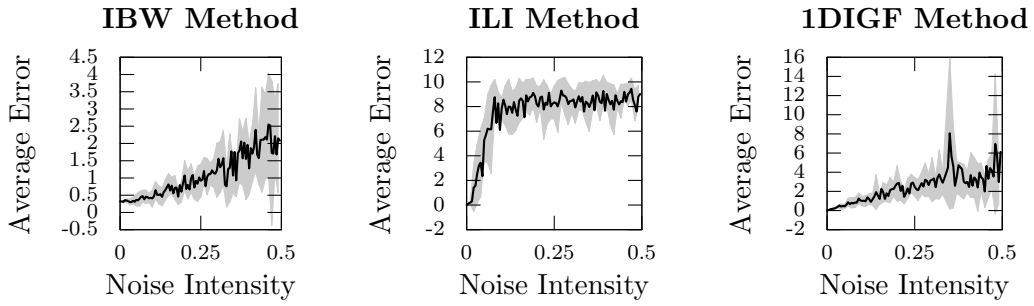


Figure 16: Plots showing the effect of noise on the average error for three of the methods.

5.8 System Evaluation

Once the sub-pixel sampling method is chosen, the entire tracking system can be executed. As shown in Figure 8, the ideal Bayes classifier identifies

potential circles in the input image. The nearest-neighbor classifier is then used to determine the likelihood of an actual circle being located at each of the potential regions. Intensity barycentric weighting can then be applied to the likelihoods of each region to accurately find the centers of the circles. Finally, a simple geometric quadrilateral region detector extracts the position of fiducials in the image.

In this section, different aspects of the overall tracker will be individually tested. All tests are performed on artificial sequences, where an image containing the marker is placed on a white background. While generating this artificial sequence, it is possible to change different parameters such as noise and amount of inter-frame motion to observe their influence on the tracking results.

5.8.1 Sensitivity to Noise

To test the noise sensitivity of the algorithm, a small program was devised. The influence of noise is measured over a 1000 simulated frame sequence and the average of the tracking error is reported. The program is then executed for different amounts and kinds of noise to generate the plots that will be shown in this section. As will be shown later, the program can also report the detection rate as the number of frames in which the marker was found over the 1000 frames sequence.

Figure 17 demonstrates that when the image contains additive Gaussian noise, the tracking error increases when the intensity of the noise increases. As the plot demonstrates, the error increases rapidly when noise is added, but settles to a linear growth at a noise intensity of approximately 30.

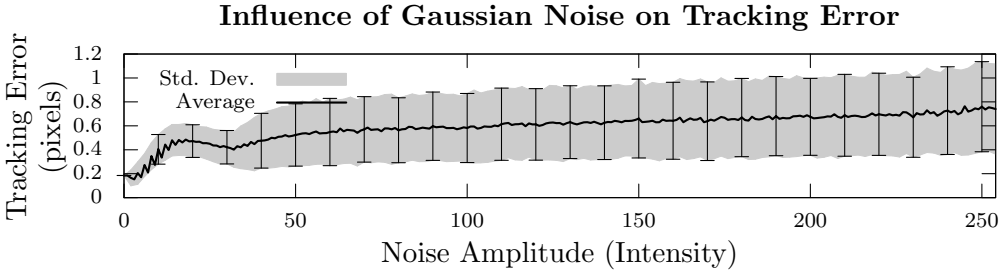


Figure 17: Plot of the average tracking error when the tracker is presented with image sequences containing different amplitudes of Gaussian noise.

The same experiment can be done once more with non-Gaussian noise to

determine how different types of noise influence the tracker. One type of non-Gaussian noise that is commonly found in digital images is salt and pepper noise. It can be described as a set of black and white pixels randomly distributed in the image. Figure 18 shows that when the image contains different amounts of salt and pepper noise, the tracking error increases rapidly. The plot shows the tracking error up to a point when forty percent of the pixels in the image are black or white noise. Measurement of the tracking error with more noisy pixels is difficult because the detection rate falls dramatically.

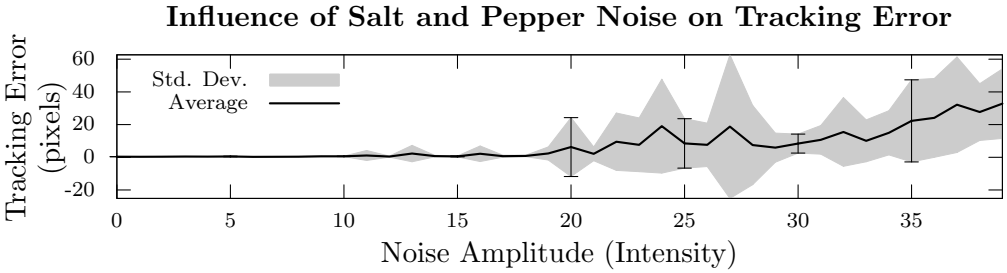


Figure 18: Plot of the average tracking error when the tracker is presented with image sequences containing salt and pepper noise with varying percentage of noisy pixels.

The relation between the accuracy of the detector to the amount of noise shown in Figures 17 and 18 does not give any information about the rate of detection. When there is a large amount of noise, the detector may fail to properly locate the circular markers in all frames. Figure 19 provides information about the percentage of frames in which the fiducial marker is successfully detected given different amounts of Gaussian noise. As can be seen, Gaussian noise has almost no influence of the detection rate.

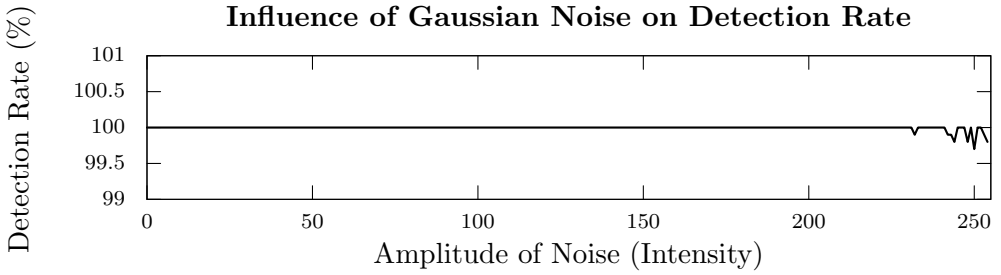


Figure 19: Plot of the detection rate when the tracker is presented with image sequences containing different amplitudes of Gaussian noise.

For non-Gaussian salt and pepper noise, the plot shown in Figure 20 is quite different. When there are less than 10% of the pixels affected by noise, the detection rate remains at almost perfect. When more than 10% of the pixels are affected, the detection rate steadily decreases to a zero detection rate at around 40%.

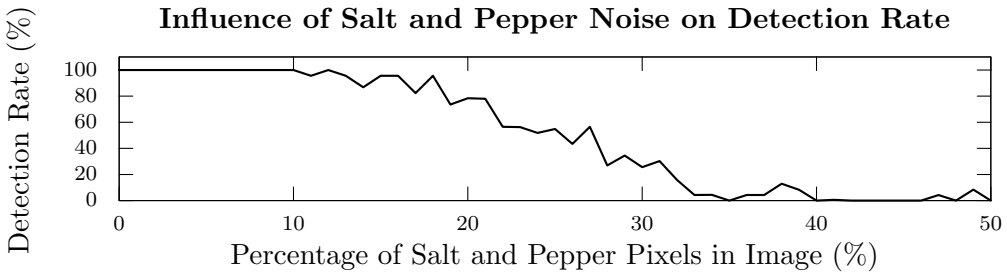


Figure 20: Plot of the detection rate when the tracker is presented with image sequences containing salt and pepper noise with varying percentage of noisy pixels.

5.8.2 Sensitivity to Motion

As mentioned earlier, the algorithm presented in this section is a detector. This means that no information is carried from one frame to the next and the entire image is processed at each time step. For this reason, the performance of the detector should be the same given any amount of inter-frame motion. Figure 21 confirms this reasoning and shows that the error remains fairly constant even when presented with sequences containing a large amount of motion. The gaps in the detection could be filled by using the predicted value from a Kalman filter.

5.9 Strengths and Weaknesses

The two first strengths of the algorithm presented here are the ability to cope with large motion and the robustness to noise. Both of these strengths come from the fact that the algorithm is executed on each incoming video frame independently and that no information is carried from one frame to the other. If an input frame is corrupted, for example by noise, the marker will simply not be detected in that frame, without influencing the detection of the next frames. If required, the gaps in the detection could be filled by the use of an

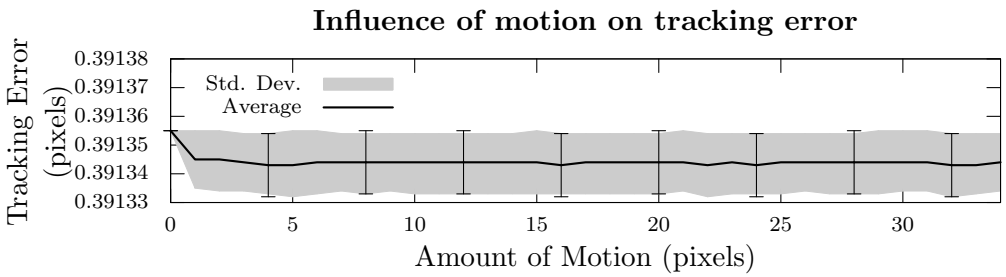


Figure 21: Plot of the average tracking error when the tracker is presented with noise-free image sequences containing different amounts of motion.

appropriate Kalman filter, as presented in Section 12.1. The detector is thus not prone to drifting, which gives this algorithm a clear advantage over the tracker that will be presented in the next section. The main weakness of the detector is that having to locate fiducials in each frame independently has a high computational cost.

6 Template Matching

This section details an incremental tracker that should be more computationally efficient than the detector of Section 5. The current tracker also uses natural texture information from the object being tracked and thus, no marker need to be added. The template tracking method implemented is presented in [25] and [26]. The second paper provides a thorough discussion of the mathematics involved. The major difference between the chosen method and other template-matching methods is that online computational costs are minimized by making two assumptions about the tracking task. These assumptions will be discussed later.

The overall structure of the program is shown in Figure 22.

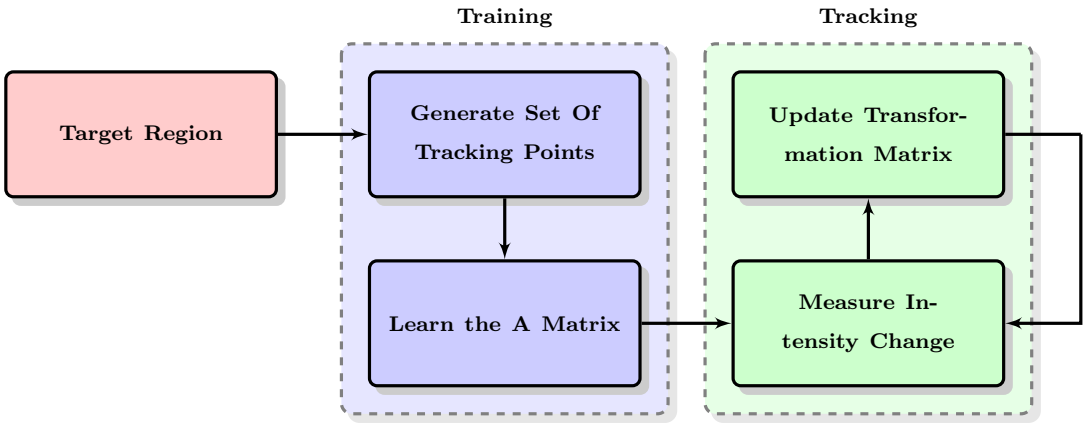


Figure 22: Block diagram of the template tracker

6.1 Motion Vector

$$\boldsymbol{F}\ P_{rect} = P_{img} \tag{7}$$

Instead of explicitly tracking the position of the target points in the image, the proposed method tracks the 3×3 transformation matrix $\boldsymbol{F}$ that projects the original tracking points from their homogeneous coordinates inside the tracked rectangle P_{rect} to their position in the image P_{img}. This projection is shown in Equation (7). Jurie and Dhome [26] refer to the projection matrix as the motion transformation matrix. A section of [25] is dedicated to the description of two different motion models.

$$\boldsymbol{F}_{similarity} = \begin{bmatrix} s \cdot \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & s \cdot \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

The first one, similarity motion, tracks the translation, rotation, and scale of the image patch being tracked. The transformation matrix representing this model is shown in Equation (8), where s is the scale, θ is the angle of rotation, and the pair (t_x, t_y) represents the translation along the x and y axis, respectively. This model is only valid when the camera's principal axis is orthogonal to the surface being tracked during the entire tracking sequence.

$$\boldsymbol{F}_{homography} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \tag{9}$$

The second model considers a full homography that allows the tracking of planes moving in three dimensional space under perspective projections. The transformation matrix for this second model is shown in Equation (9).

$$\boldsymbol{F} = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

For the context of the current thesis, the motion model can be greatly simplified by noting that the machine being tracked only moves along the x, y, and z axes. If the camera's principal axis is placed parallel to one of the three coordinate axis (x, y, or z), only translation and scale will be perceived by the camera. This allows the use of a reduced similarity model with transformation matrix shown in Equation (10). This matrix is a special case of Equation (8) when the rotation parameter θ is set to 0.

$$\boldsymbol{\mu} = [s, t_x, t_y] \tag{11}$$

While seemingly insignificant, this simplification greatly reduces both the computational cost and the amount of training examples required for proper tracking. As only three parameters are required to create the transformation matrix, it is possible to define a state vector $\boldsymbol{\mu}$ which has the following form:

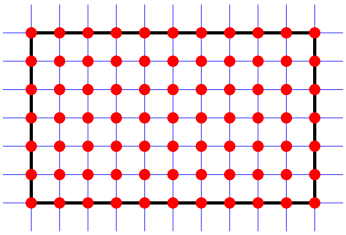


Figure 23: Selection of the tracking points (red) on regular grid (pale blue) given a user supplied rectangular region (black)

6.2 Training Stage

The program begins by waiting for a user to input a rectangular tracking region. A group of tracking points is then generated by placing points on a regular grid filling the region. The set of points will be referred to as $\mathcal{R}$.

$$\mathcal{R} = \begin{bmatrix} P_{0x} & P_{1x} & \cdots & P_{nx} \\ P_{0y} & P_{1y} & \cdots & P_{ny} \\ 1 & 1 & \cdots & 1 \end{bmatrix} \tag{12}$$

Figure 23 gives a graphical representation of how tracking points are chosen given a rectangular region. The set of points can be represented as a matrix, as shown in Equation (12). This matrix form is used mostly for transformation purposes.

$$\boldsymbol{F}_0 = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \Longleftrightarrow \boldsymbol{\mu}_0 = [1, x_0, y_0] \tag{13}$$

The point in the top-left corner is taken to be the origin of the tracking region and its coordinates are referred to as (x_0, y_0). From that, the initial transformation matrix and associated motion vector have the form shown in Equation (13).

$$\delta\boldsymbol{\mu} = \mathbf{A}_j \delta\mathbf{I} \tag{14}$$

From this, an intensity vector $\mathbf{I}_0$ representing the template to be tracked is obtained by sampling the input image at each point $F_0\mathcal{R}$. The goal of the training stage is then to learn the relation between a change of intensity $\delta\mathbf{I} = \mathbf{I}_{current} - \mathbf{I}_0$ and the movement that generated it $\delta\boldsymbol{\mu} = \boldsymbol{\mu}_{current} - \boldsymbol{\mu}_0$.

This relation is the solution $\mathbf{A}_j$ to Equation (14). The current implementation of this model is valid only when tracking planar surfaces with constant illumination and reflectance.

To obtain the global relation between motion and intensity change, the initial tracking region is randomly shifted to simulate motion. Multiple such shifts are created to ideally cover all possible motions from the initial region. These shifts are generated by creating a set of randomized transformation matrices with values taken from a Gaussian distribution centered around the initial values. The following equation shows the form of a random transformation matrix:

$$\boldsymbol{F}_{random} = \begin{bmatrix} s_r & 0 & x_r \\ 0 & s_r & y_r \\ 0 & 0 & 1 \end{bmatrix} \Longleftrightarrow \boldsymbol{\mu}_{random} = [s_r, x_r, y_r] \tag{15}$$

where

$$\begin{aligned} s_r &\sim \mathcal{N}(1, 0.05) \\ x_r &\sim \mathcal{N}(x_0, 5) \\ y_r &\sim \mathcal{N}(y_0, 5) \end{aligned}$$

The choice in standard deviation of the Gaussian distributions will influence the amount of inter-frame motion that the tracker can properly track. A small standard deviation will restrict the maximum allowed movement between two consecutive frames while a large value allows for larger displacement, but sometimes leads to instability. The values are thus very dependent on the situation and must be chosen on a case by case basis.

$$\boldsymbol{M} = [\delta\boldsymbol{\mu}_{\mathbf{0}}^T \ \delta\boldsymbol{\mu}_{\mathbf{1}}^T \ \cdots \ \delta\boldsymbol{\mu}_{\boldsymbol{m}}^T] \tag{16}$$

$$\boldsymbol{V} = [\delta\mathbf{I}_{\mathbf{0}} \ \delta\mathbf{I}_{\mathbf{1}} \ \cdots \ \delta\mathbf{I}_{\mathbf{m}}] \tag{17}$$

Given a set of m random transformation matrices, each with an associated $\delta\boldsymbol{\mu}$ and $\delta\mathbf{I}$ vector, two matrices can be built, as shown in Equations (16) and (17). $\boldsymbol{M}$ is a $3 \times m$ matrix, while $\boldsymbol{V}$ is a $n \times m$ matrix, where n is the number of points in $\mathcal{R}$, as stated earlier.

$$\boldsymbol{M} = \boldsymbol{A}\ \boldsymbol{V} \tag{18}$$

The general relation between the motion $\delta\boldsymbol{\mu}$ and change in intensity $\delta\mathbf{I}$ can now be found by solving Equation (18) for $\boldsymbol{A}$.

$$\boldsymbol{A} = \boldsymbol{M} \cdot (\boldsymbol{V}^t \cdot \boldsymbol{V})^{-1} \cdot \boldsymbol{V}^t = \boldsymbol{M} \cdot \boldsymbol{V}^+ \tag{19}$$

As n is chosen to be much greater than three, Equation (18) is an over-determined system of linear equations. For this reason, a simple linear least-square solution of the form shown in Equation (19), where V^+ is the Moore-Penrose pseudoinverse, is suitable.

Because the motion model is simplified, a few hundred random shifts ($n \approx 200$) have been found to provide a good result.

6.3 Tracking Stage

As we will now see, once the training stage has been completed and the $\boldsymbol{A}$ matrix is known, the tracking algorithm is trivial. The tracking starts with a projection matrix $\boldsymbol{F}_{current} = \boldsymbol{F}_0$. When a new frame arrives from the camera, the image is sampled at each point $\boldsymbol{F}_{current} \cdot \mathcal{R}$ to obtain $\boldsymbol{I}_{current}$. From this intensity vector, $\delta\boldsymbol{I} = \boldsymbol{I}_{current} - \boldsymbol{I}_0$ can be computed. The motion $\delta\boldsymbol{\mu}$ between two consecutive frames can then be found using Equation (14). The current state vector is then updated by adding $\delta\boldsymbol{\mu}$ to the previous state vector $\boldsymbol{\mu}$. As the state vector is an alternate representation of the transformation matrix, updating the state vector effectively updates the transformation matrix.

6.4 Determining Optimal Parameters

While there is no clear mathematical way of determining the values for the parameters used by this method, an empirical data analysis provides enough information to choose adequate parameter values.

6.4.1 Number of Training Examples

The strength of the method is to be able to learn the changes in brightness associated with changes in the transformation matrix. This training is done by generating multiple random shifts and noting the associated effect on the brightness vector. Figure 24 shows the average tracking error for different

numbers of initial training shifts. Each data point of Figure 24 is found by looking at the average tracking error over a 1000 frame sequence containing an image moving in a circle.

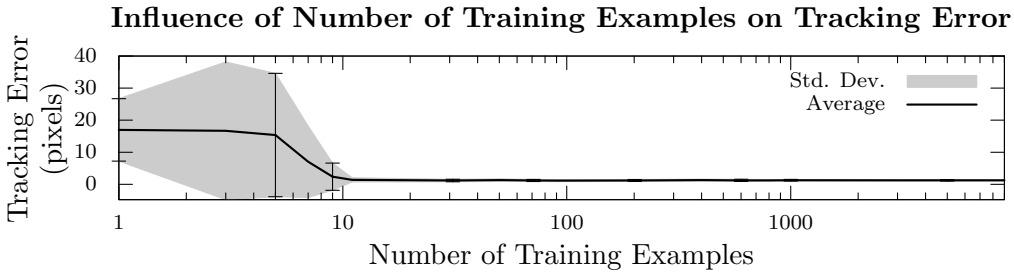


Figure 24: Plot of the average tracking error when the tracker is trained with different number of examples.

Figure 24 demonstrates that the number of examples does not have an important influence on the tracking accuracy, when the number of examples is greater than approximately 10. Thus, in some very unlikely situations, all examples may be similar and thus the tracker is unable to learn an appropriate relation between motion and brightness changes. By looking at the graph, it was decided that 100 examples would be used. Choosing a smaller number would increase the probability of running into situations when the examples would all be similar and choosing a larger number would increase the time required by the training stage with no significant gain in tracking accuracy.

6.4.2 Number of Tracking Points

The plot shown in Figure 25 is generated by measuring the average tracking error of a moving square image over a thousand frames for different amounts of tracking points. It can be seen that the error is quite large when the number of points is smaller than approximately 30. Like in Figure 24, there is a peak near 100, probably due to the generation of a bad training set. It is also interesting to note that when the density of tracking points on the target image is too high, the error increases. This can be seen from the errors measured with more than 5000 points. Choosing 600 points seems to yield a small average error for the current example. The image being tracked in the present test is a square with sides measuring 256 pixels. This translates to roughly one tracking point per one hundred image pixels, or one point for each ten pixel by ten pixel square area.

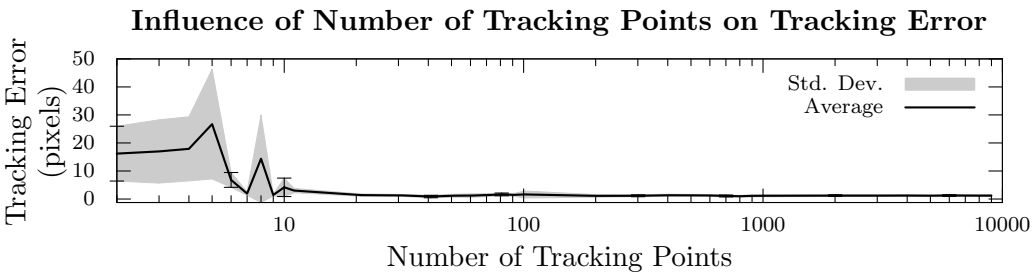


Figure 25: Plot of the average tracking error when different numbers of tracking points are used.

6.5 Oscillations

Noise in the camera image creates variation in intensity, which can confuse the tracking system. This results in the detected region oscillating around the correct position. An example of oscillation along the horizontal axis can be seen as the red curve of Figure 26. This curve was generated by tracking a motionless surface. One way to reduce the effect of such behavior is to use Kalman filtering. For this solution, three individual filters are configured to filter each component of $\boldsymbol{\mu}$. For the implementation details of the Kalman filter used, please refer to Section 12.1. The filter used to track the scale parameter was initialized with $Q_{i,i} = 0.0001$ and $R_{i,i} = 0.1$, meaning a very small amount of process noise and a large amount of measurement noise. This roughly translates in the assumption that scale should not vary by large amounts quickly. Similarly, the filters used to track the translation parameters were initialized with $Q_{i,i} = 0.1$ and $R_{i,i} = 0.8$. These values were found experimentally. If the filtering strength is too high, tracking performance is reduced because small motions would not be detectable. The output of such a strong filter is shown as the blue curve of Figure 26. Thus, the filters must be configured to reduce the effect of the oscillation, but eliminating the oscillation is an unattainable goal. The output of a filter that reduces the effect of the oscillation without causing too much lag is shown in blue on Figure 26. If a large amount of lag is present, the tracker will loose the true position of the template and drifting will occur.

Figure 26 has been generated by tracking a plane using a real camera. It can be noted that even with the oscillation, the position of the plane can be found to sub-pixel accuracy.

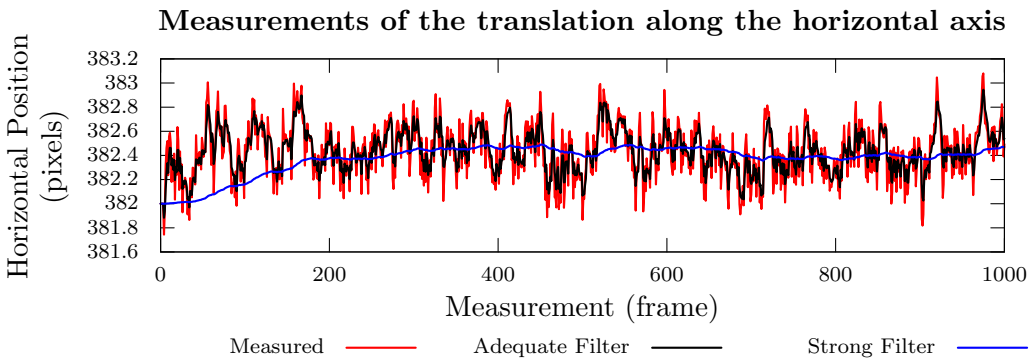


Figure 26: Plot of the detected horizontal position of a tracked region over a period of 1000 frames (red) and the outputs of Kalman filters (black and blue).

6.6 System Evaluation

As done for the fiducial marker detector of Section 5.8, the template tracker will now be tested.

6.6.1 Sensitivity to Noise

As the algorithm considers multiple tracking points, the overall effect of noise should be minimal if the noise distribution is Gaussian (white noise). The effect of noise is also reduced by the use of Kalman filtering. To test the sensitivity of the algorithm to noise, Figure 27 was generated by tracking an image over a sequence of images. This test was executed 256 times with different noise amplitudes.

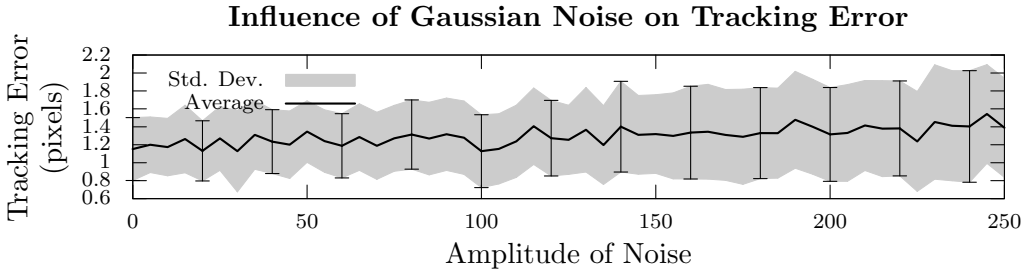


Figure 27: Plot of the average tracking error when the tracker is presented with image sequences containing different amplitudes of noise.

As can be seen, in the presence of higher Gaussian noise amplitude, the average error increases. While being expected, it is important to note that the

error only increases by a single pixel. This means that the template tracking algorithm presented in this section handles noisy images quite well.

To test the performance of the template tracker when presented with images containing non-Gaussian noise, an experiment with variable amounts of salt and pepper noise was conducted. The testing methodology was the same as used in Section 5.8.1 to generate Figure 18. The results of the experiment are shown in Figure 28. It can be seen that the effect of non Gaussian noise on the average error is similar for both the fiducial marker detector and the template tracker.

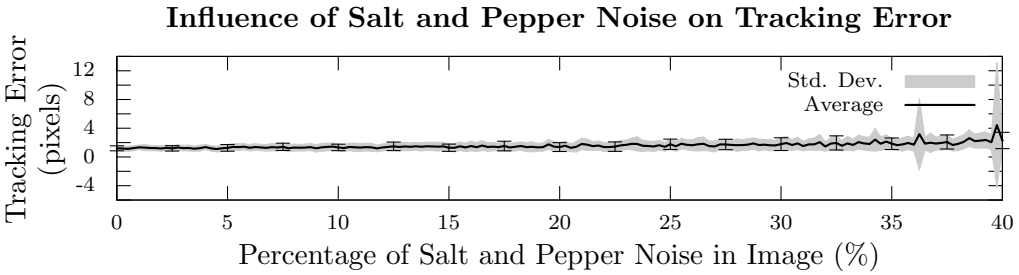


Figure 28: Plot of the average tracking error when the tracker is presented with image sequences containing salt and pepper noise with varying percentage of noisy pixels.

6.6.2 Sensitivity to Motion

The goal of this test is to determine how much inter-frame motion is allowed by the tracker. The average error was measured over sequences of a thousand frames with different amounts of inter-frame motion. Figure 29 is a plot of the results. The average error seems to grow somewhat linearly for motions between zero and two pixels. There is a discontinuity between two and three pixels, where the error increases.

From Figure 29, it can be seen that the tracker fails at a value between two and three. To be safe, the motion should be kept below two pixels. In a standard 30 Hz video feed, moving two pixels per frame means a total motion of 60 pixels per second, which is acceptable. Faster motions could be tracked by capturing images from the camera at a higher rate.

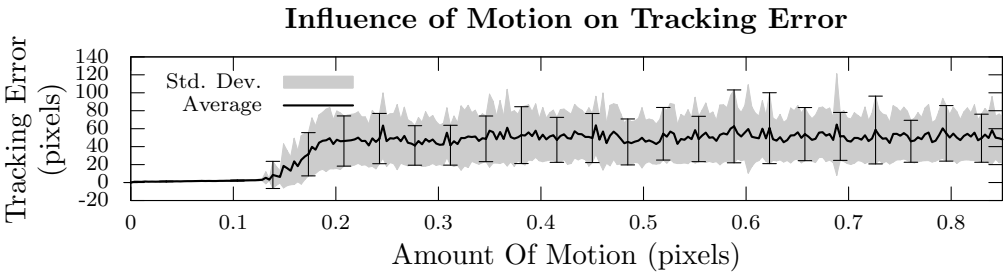


Figure 29: Plot of the average tracking error for different amounts of motion.

6.7 Strengths and Weaknesses

The particular strength of the method is its very small online computational cost. If the tracked object is known to be the same, the training can be done a single time and used indefinitely. The algorithm's strength also comes from its ability to cope with large amplitudes of noise.

The major weakness of the template matching algorithm is that it is based on the assumption that the same change in position will result in the same change in brightness. While being a valid assumption in certain cases, it is only true when both illumination and reflectance is uniform across the scene and that the light intensity is maintained at a constant level throughout the tracking sequence. This does not cause much problem in a factory environment where lighting can be precisely controlled, but renders the algorithm virtually useless in most real-world situations. The second potential problem with the use of this method is that it is prone to drift. For this reason, the tracking region should be periodically reset to prevent inaccuracies.

Finally, a small downside of this tracking method is that it requires manual initialization. As will be seen in later section, an automatic initialization can be developed.

6.8 Improving the Template Tracker

As stated earlier, the topic of object tracking plays an important role in the augmented reality literature. For this reason, methods of addressing the downsides of certain types of tracking algorithms have been developed.

A recurring topic in the augmented reality field is the fusion of information from different types of sensors to provide a more complete representation of the world. Many tracking methods use external sensors such as gyroscopes

[27] or inertial sensors [28] to acquire more information about the object being tracked. The use of such sensors can be an integral part of a smart CNC machine controller, as shown by the "Other Sensor" block of Figure 1 of Section 1. As the scope of the current thesis will be restricted to vision systems, these sensors will not be explicitly discussed.

7 Three-Dimensional Data From Two-Dimensional Tracking

The previously covered trackers do not provide any information about the depth of the tracked object. While it may seem problematic for the purpose of tracking three-dimensional objects, the information from the camera calibration can be used to compute depth.

$$X_w = a_x \cdot Z_w + b_x \tag{20}$$
$$Y_w = a_y \cdot Z_w + b_y \tag{21}$$

For any known point on the image, the calibration model can be used to find the equations relating the position on the Z_w axis to the positions on the X_w and Y_w axis of the point in real-world coordinates. These are linear equations of the form shown in Equations (20) and (21).

Given a single 2D point, there are obviously an infinite amount of valid solutions and it is thus impossible to find the depth of the point. The problem can only be solved by adding one or more constraints to restrict the solution space to a single point. A common way to add such a constraint is to use a stereo camera setup. From two images, four equations are obtained and the solution is found where the lines intersect.

$$s^2 = (X_{w1} - X_{w2})^2 + (Y_{w1} - Y_{w2})^2 + (Z_{w1} - Z_{w2})^2 = D_{12}^2 \tag{22}$$
$$s^2 = (X_{w2} - X_{w3})^2 + (Y_{w2} - Y_{w3})^2 + (Z_{w2} - Z_{w3})^2 = D_{23}^2 \tag{23}$$
$$s^2 = (X_{w3} - X_{w4})^2 + (Y_{w3} - Y_{w4})^2 + (Z_{w3} - Z_{w4})^2 = D_{34}^2 \tag{24}$$
$$s^2 = (X_{w4} - X_{w1})^2 + (Y_{w4} - Y_{w1})^2 + (Z_{w4} - Z_{w1})^2 = D_{41}^2 \tag{25}$$
$$2s^2 = (X_{w1} - X_{w3})^2 + (Y_{w1} - Y_{w3})^2 + (Z_{w1} - Z_{w3})^2 = D_{13}^2 \tag{26}$$
$$2s^2 = (X_{w2} - X_{w4})^2 + (Y_{w2} - Y_{w4})^2 + (Z_{w2} - Z_{w4})^2 = D_{24}^2 \tag{27}$$

When tracking a rectangular surface, the positions of four two-dimensional points are known. This means that a total of eight equations relating these points to their real world positions exist. In this particular case, the constraint that can be added to solve the system is the known dimension of the sides of the rectangle. If we consider a square of side s with vertices (X_{w1}, Y_{w1}, Z_{w1}),

(X_{w2}, Y_{w2}, Z_{w2}), (X_{w3}, Y_{w3}, Z_{w3}), and (X_{w4}, Y_{w4}, Z_{w4}), labeled in a clockwise order, the constraints would have the form of the expressions shown in Equations (22) through (27).

By using Equations (20) and (21), the previous six equations can be rewritten only in terms of X_w's. Thus six equations containing four variables need to be solved. In most cases, no proper mathematical solution exists due to slight inaccuracies in the calibration model and noise in the input data. For this reason, the system of equations is solved by using a Levenberg-Marquardt (LM) algorithm to find the least squares solution. A simple least squares approach might also be appropriate, but as the LM algorithm had already been used for the non-linear curve fitting of Section 5.7.7, it was also used for the current task. While the LM algorithm might have a higher computational cost, if the 2D-to-3D conversion step is done offline, the added computational complexity would not impact the performance of the final system. More details about the offline computation will be presented in Section 7.1.

$$X_{w1} = X_{w2} = X_{w3} = X_{w4} \tag{28}$$

Simplifications similar to those made to the template tracker can be used to simplify the minimization problem. If the tracking region is a plane that is orthogonal to the X axis of the camera model, it can be noted that the depth of any point lying on the tracked plane will be the same. This simplification adds an additional constraint on the system, shown in Equation (28).

$$(s^2 - D_{12}^2), (s^2 - D_{23}^2), (s^2 - D_{34}^2), (s^2 - D_{41}^2), (2s^2 - D_{13}^2), (2s^2 - D_{24}^2) \tag{29}$$

$$\begin{aligned} R = \Big[& (s^2 - D_{12}^2)^2 + (s^2 - D_{23}^2)^2 + (s^2 - D_{34}^2)^2 \\ & + (s^2 - D_{41}^2)^2 + (2s^2 - D_{13}^2)^2 + (2s^2 - D_{24}^2) \Big]^{\frac{1}{2}} \end{aligned} \tag{30}$$

This final constraint reduces the parameter space of the optimizer to a single dimension, which greatly reduces computation time. To provide a rough estimate of the increase in speed, the average time required to run the minimization stage both with and without the final constraint has been measured

over a thousand frames. The average time required to run the optimization without the constraint is 0.00026 seconds and when the constraint is added, the time decreases to 0.00006 seconds. The simplification allows the system to be solved roughly four times faster. The six remainders used for the Levenberg-Marquardt minimization are shown in Equation (29) and the remainder function has the form shown in Equation (30).

Given a marker known to be located at exactly $X_w = 0$, a sweep can be done to find the values of the remainder function for different values of X_w. The plot of the remainder is shown in Figure 30 where it can be seen that the minimum is indeed very close to $X_w = 0$.

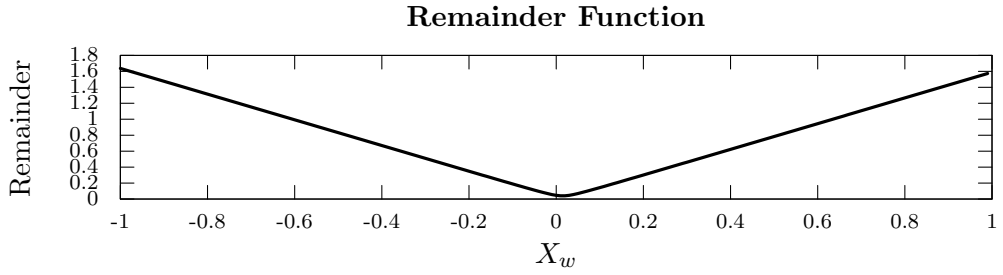


Figure 30: Plot of the remainder function for different values of X_w.

While leaving the marker at a position of $X_w = 0$, the two-dimensional position of the four circles contained in the fiducial marker can be randomly shifted to simulate uncertainty in the 2D detector's output. Figure 31 demonstrates that the error grows linearly with respect to the amplitude of noise in the 2D positions of the circles. Please note that the amplitude of the curve in Figure 31 is shown as an example and is valid only for the calibration parameters used. This means that if any of the camera parameters change, the curve would have to be regenerated.

7.1 Implementation Notes

The optimization stage is the most computationally expensive step in the overall tracking system. For a practical implementation, the computational costs could be greatly reduced by precomputing a table of conversions from two dimensional point sets to their respective three dimensional coordinates. The use of such a lookup table is only possible if a somewhat large amount of storage is available. The exact amount of storage required is dictated by the

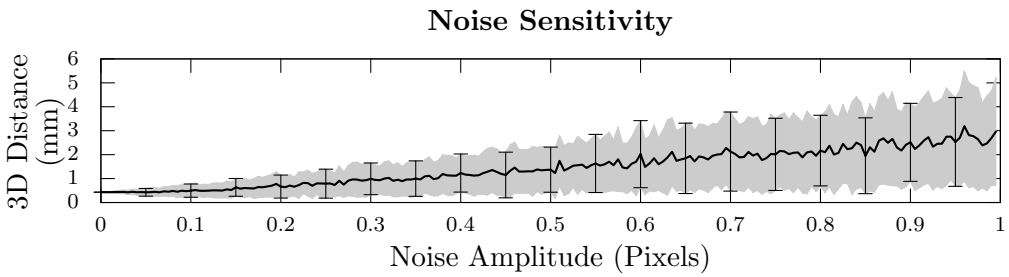


Figure 31: Plot of the error for different amplitudes of noise.

size of the bins used to discretize the space of 2D coordinates.

As an example, if we consider an image with a resolution of 640 pixels by 480 pixels, the lookup table would contain $640 \cdot 480 = 307200$ cells. Now, assuming that the three-dimensional volume in which the marker is allowed to move is discretized into integer units such as millimeters or micrometers and that there is less than 2^{16} such units along each of the three axes, the three-dimensional coordinates could be stored as an array of three 16-bits integers (short int). Considering that each cell of the table is required to hold a three element array, the total memory requirement is $640 \cdot 480 \cdot 3 \cdot 16 = 14745600$ bits, roughly 15 megabytes. If the desired precision requires a larger number of bits to store the integer representation of the three-dimensional coordinates or if the image has a larger resolution, the memory requirement will be greater. With the available memory of modern computers, storing such a lookup table should not be a problem.

8 Model-Based Tracking

The trackers of Sections 5 and 6 both required an extra step to convert the output of the 2D tracker to a 3D position, as demonstrated in Section 7. The present section will detail a tracker that tracks directly in the three dimensional space by fitting the 3D model of an object to edges in the input image. The model-based tracker implemented here is presented in multiple papers co-authored by Tom Drummond [7],[29], and [30]. A diagram showing the different steps of the tracking algorithm is presented in Figure 32.

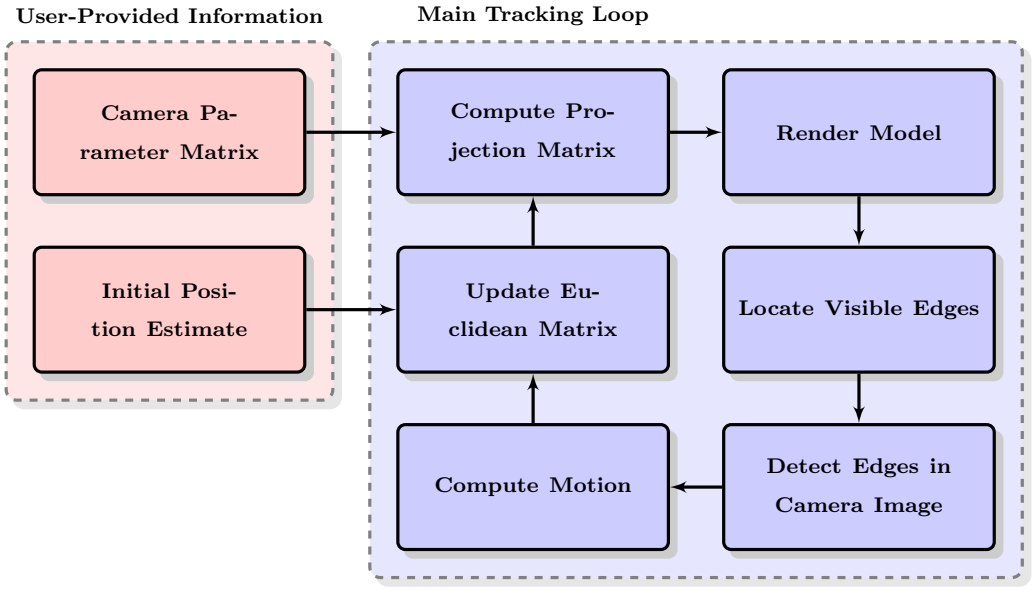


Figure 32: Block diagram of the methodology of the model-based tracker, reproduced from [29]

The following subsections will provide more information about the algorithm of the tracker.

8.1 Camera Model

For the purpose of simplifying the problem, the camera model [11] used for the implementation of the tracker is the same as the one OpenGL uses. This reduces the amount of coordinate transformations required to convert between different coordinate systems. It is also interesting to note that OpenGL uses homogeneous coordinates, so that both projection matrices and geometric transformations are represented as 4 by 4 matrices.

$$\mathcal{M} = \begin{bmatrix} s\mathbf{R} & T \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{31}$$

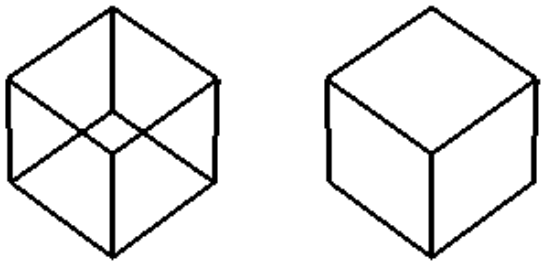
The OpenGL camera system is governed by the model-view, the projection and the view-port matrices. To prevent confusion with the real camera projection matrices, a caligraphic font will be used for the OpenGL matrices. The projection matrix $\mathcal{P}$ defines the size and shape of the perspective frustum, namely, the volume of space that will be visible when a scene is rendered. This matrix will be constant through the execution of the tracker. The view-port $\mathcal{V} = [X, Y, W, H]$ is a 4-dimensional vector defining the size of the rendered image where X and Y represent the pixel coordinates of the north-west corner of the view-port, W represents the width of the image, and H the height. Like the projection matrix, the view-port remains constant. Finally, the model-view matrix $\mathcal{M}$ encodes the position (T), rotation ($\mathbf{R}$), and scale (s) of the scene with respect to the camera. Its form is shown in Equation (31). This matrix will be updated for each new frame acquired from the camera.

The scale parameter s can be set to 1 as the current algorithm assumes that the size of the object being tracked remains constant throughout the execution. Changes in size of the object on the projection plane are caused by translations along the z-axis.

8.2 Model Rendering

In order to obtain edge information, a line-art rendering of the model is generated. The OpenGL library provides a polygon mode to generate such renderings by drawing only the edges of polygons. An example of a cube rendered using this polygon mode is shown in Figure 33(a). While containing the model's edges, this first rendering does not provide information about which edges are hidden because of occlusion. Figure 33(b) demonstrates that occluded edges can be removed by painting the surfaces with an offset fill. At this point, line drawings of simple polyhedral models can be obtained.

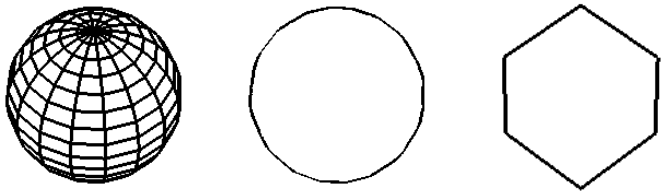
If the model of the tracked object is a tessellated smooth model, the rendering method detailed earlier will cause some problems. This can be seen in Figure 34(a). For smooth surfaces, only the contours should be rendered. Enabling face culling when rendering the model hides edges that should not be detectable, leaving only the contour of the model, as demonstrated in Figure 34(b). Figure 34(c) shows that face culling removes visible edges when render-



(a) (b)

Figure 33: Rendering of the edges of a polyhedral model (a) without painting the surfaces and rendering of the same model (b) with an offset surface fill to hide edges that are non visible.

ing simple geometric models. For this reason, care must be given to determine the best way to render a model based on its geometry.



(a) (b) (c)

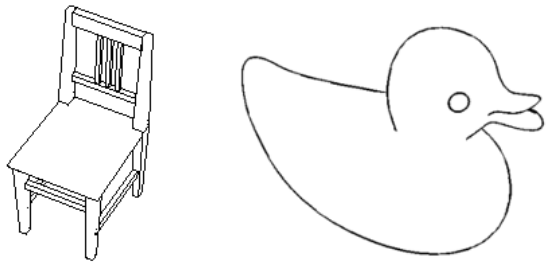
Figure 34: Rendering of (a) the edges of a tessellated smooth model. Rendering (b) with face culling enabled. (c) Face culling of simple geometric models.

From these rendering methods, models of varying complexity can be rendered. Figure 35 provides two examples of more complex models being rendered. Both models shown here are Creative Commons models.

8.3 Locating Edges

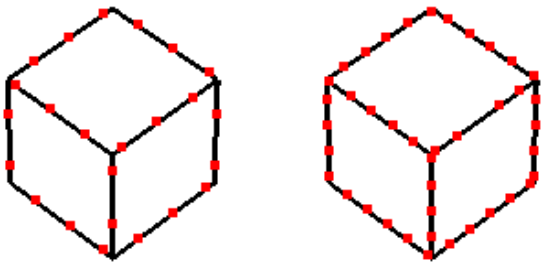
Once an edge image is rendered, the edges of the model must be located in the image from the camera. Points are placed at regular intervals on the visible edge segments of the rendered image. The exact number of points can be changed, as seen in Figure 36. The set of tracking points will be referred to as $\mathcal{P}$.

For each point in $\mathcal{P}$, a line in the direction perpendicular to the edge is searched in the camera image to locate an edge.



(a) (b)

Figure 35: Rendering of (a) a more complex polyhedral and (b) smooth model.

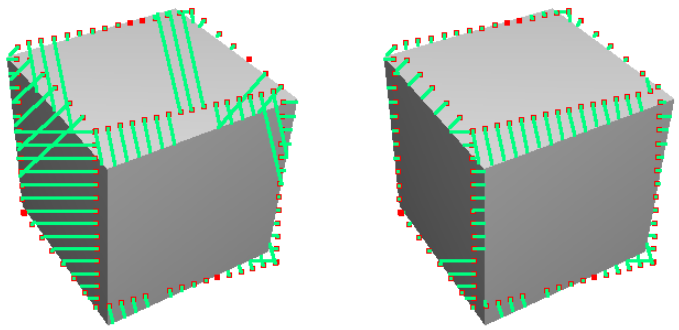


(a) (b)

Figure 36: A variable number of points on visible edge segments can be chosen.

The first approach evaluated for the current implementation was a naive maximum gradient search. The problem with this approach is that the maximum gradient along the search path does not necessarily correspond to the correct edge. Such a situation can be seen in figure 37(a), where the red points are the point in $\mathcal{P}$ and the green line represent the distance to the detected edges in the direction of the normal. It can be seen that edges are properly detected, but the edges found are often not the ones corresponding to the tracking points.

The second approach was to use non-maxima suppression on the gradients along the normal and weight each remaining gradients by an inverse function of their distance to the point in $\mathcal{P}$ where the search started. As can be seen in Figure 37(b), there are still a few mis-detections, but most edges are located properly.



(a) (b)

Figure 37: Search for the edge closest to points on the visible line segments. Naive search for the strongest edge is shown in (a). Edge search using non-maxima supression and distance weighting is shown in (b).

$$\mathbf{e} = [d_{\mathcal{P}0}, d_{\mathcal{P}1}, \cdots, d_{\mathcal{P}n}]^T \tag{32}$$

An error vector $\mathbf{e}$ can be constructed from the distances between the tracking points and the edges found, as shown in Equation (32). This vector quantifies the edge-normal distance between the current state of the model and the image from the camera.

8.4 Computing Motion

$$G_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad G_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{33}$$

$$G_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad G_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Now that an error vector is known, the transformation required to modify the state of the model to fit the camera image can be computed. Motion is considered to be a mixture of six transformations [29], namely, translations and rotations along each of the three axes. The set of matrices in Equation (33) show the form of each transformation, which are referred to as motion generators. The description of the algorithm will include all six generators even if only the first three generators are required as the CNC machine considered can only move along the three axes, as discussed in previous sections.

$$M = \exp\left(\sum_{j=1}^{6} \alpha_j G_j\right) \tag{34}$$

$$M \approx I + \sum_{j=1}^{6} \alpha_j G_j \tag{35}$$

The transformation matrix **M** can be created from the generators from the matrix exponential shown in Equation (34). Equation (35) provides a first order approximation of the exponential. This approximation, used to simplify computations, is stated to be appropriate for small displacements between

frames [29]. This assumption will be tested in a later section.

$$\mathbf{JA} = \mathbf{e} \tag{36}$$

The $\mathbf{A} = [\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6]$ vector of Equations (34) and (35) represents the quantity of each generator that needs to be applied to correct the state of the model. This is what needs to be found. Equation (36) shows the system that can be solved to find $\mathbf{A}$.

$$\mathbf{J}_{ij} = \frac{\partial e_i}{\partial \alpha_j} = \mathbf{n} \begin{pmatrix} \dfrac{\partial u}{\partial G_j} \\ \dfrac{\partial v}{\partial G_j} \end{pmatrix} \tag{37}$$

The $\mathbf{J}$ matrix of Equation (36) is a Jacobian matrix with elements of the form shown in Equation (37) (reproduced from [29]), where $\mathbf{n}$ is the unit normal to the edge where the i^{th} point is located.

$$\frac{\partial u}{\partial G_j} = \frac{u'}{w'} - \frac{u}{w} \tag{38}$$

$$\frac{\partial v}{\partial G_j} = \frac{v'}{w'} - \frac{v}{w} \tag{39}$$

The partial derivatives of the projected 2D coordinates u and v with respect to the j^{th} generator are found numerically using Equations (38) and (39).

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \mathcal{PM} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{40}$$

$$\begin{pmatrix} u' \\ v' \\ z' \end{pmatrix} = \mathcal{PM}(I + G_j) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{41}$$

The two sets of coordinates (u, v, w) and (u', v', w') from Equations (38) and (39) are the projected 2D coordinates of the 3D coordinates (x, y, z) of the tracking point. The projections are done using Equations (40) and (41).

$$\mathbf{A} = [\mathbf{J}^T\mathbf{J}]^{-1}\mathbf{J}^T\mathbf{e} \tag{42}$$

$$\mathcal{M}_t \leftarrow \mathcal{M}_{t-1}(I + \sum_{j=1}^{6} \alpha_j G_j) \tag{43}$$

Once the Jacobian matrix has been filled, the system described in Equation (36) can be solved as a linear least squares problem. The form of the solution is shown in Equation (42). Finally, the model-view matrix can then be updated by using Equation (43).

8.5 Stabilization

$$\begin{bmatrix} \mathbf{J} \\ \mathbf{I} \end{bmatrix} \mathbf{A} = \begin{bmatrix} \mathbf{e} \\ \mathbf{d} \end{bmatrix} \tag{44}$$

Stabilization refers to the addition of constraints to make the solution more robust. The stabilization method proposed in [6] was implemented. The first step of the stabilization is to prevent cases when no measurements are available. This situation translates in the **J** and **e** matrices being empty, and thus no solution could be computed by using Equation (36). To solve this, Equation (36) is replaced by Equation (44), where **I** is the identity matrix.

$$\mathbf{IA} = \mathbf{d} \tag{45}$$

This new system is very similar to that of (36), but contains a set of added rows. If no measurements are available, Equation (44) becomes the simple linear system shown in (45).

$$\begin{bmatrix} \mathbf{J} \\ \mathbf{W} \end{bmatrix} \mathbf{A} = \begin{bmatrix} \mathbf{e} \\ \mathbf{Wd} \end{bmatrix} \tag{46}$$

The solution to Equation (45) is obviously $\mathbf{A} = \mathbf{d}$. Thus, **d** should be set to a default solution. Such a solution can be computed from the velocity of the object or simply set to zero, as done in the current implementation. To make the method even more robust, the lower part of Equation (44) is multiplied by a diagonal matrix, as shown in Equation (46). The elements on the diagonal of **W** are inversely proportional to the standard deviation of their column in **J**. This prevents the default solution from having a large influence on the results when measurements are available.

8.6 Tracking Performance

8.6.1 Small Displacement Assumption

As stated in Section 8.4, the inter-frame motion is assumed to be small for the method to be valid. No formal definition of small is proposed in [29]. This is obviously due to the fact that the maximum inter-frame motion is dependent on the shape of the model being tracked. To obtain a qualitative idea of what "small" means, a model was tracked over sequences of 1000 frames with varying amounts of motion. In Figures 38 and 39, the error is defined as the pixel distance between tracking points and image edges. It must be noted that an average distance greater than approximately 15 reflects that edges could not be located and the tracker failed to properly track the model. Figure 38 shows that the tracker fails at a value between two and three.

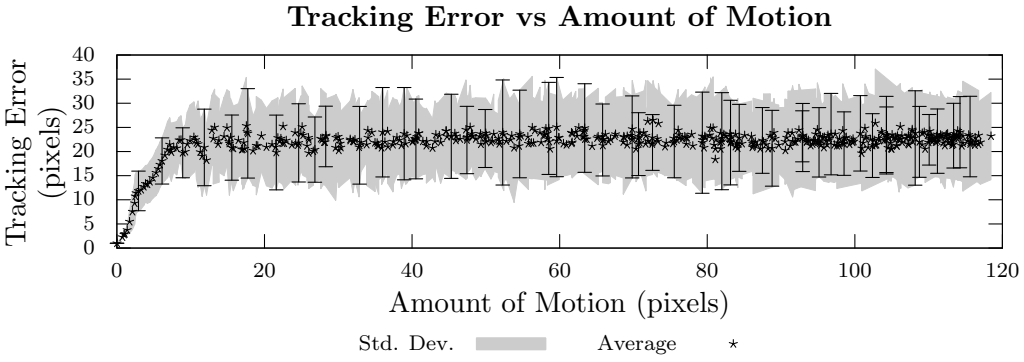


Figure 38: Plot of the tracking error given different amounts of motion.

Restricting the plot to the range where the method does not fail yields the plot shown in Figure 39. The error seems to increase linearly with respect to motion, but fails when the amount of motion is greater than approximately two and a half pixels.

As can be seen, the small motion assumption is valid up to a motion of two and a half pixels. Given an image sequence running at thirty frames per second, this translates into a motion of 75 pixels per second.

8.6.2 Lag

The model tracker presented in this section provides a position that lags behind the true position of the object. Figure 40 shows that if the true horizontal mo-

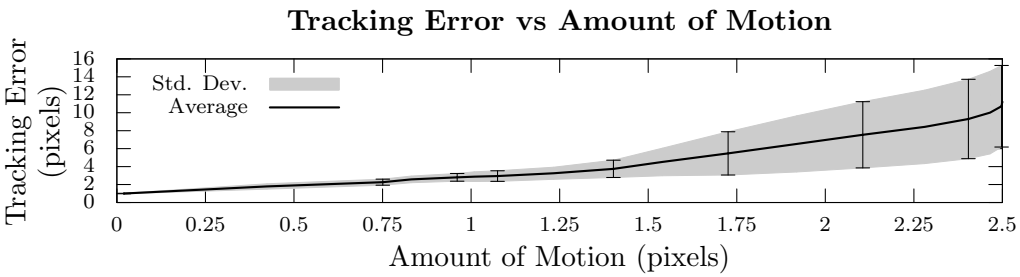


Figure 39: Plot of the tracking error given different amounts of motion.

tion of an object is sinusoidal, the output of the tracker will also be sinusoidal, but with a phase difference.

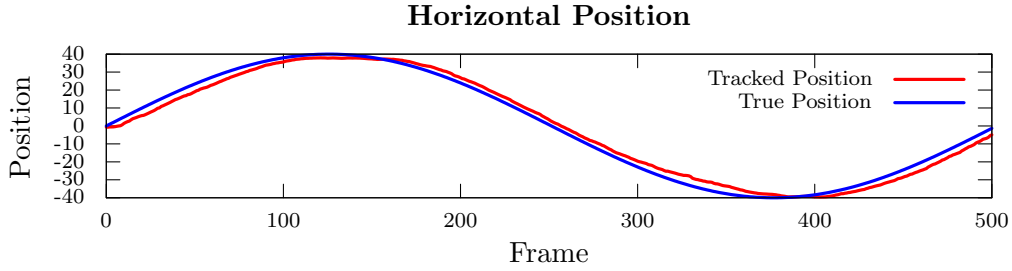


Figure 40: Plot of the output of the tracker showing lag.

8.6.3 Sensitivity to Noise

Figure 41 shows the relation between the average tracking error and the amplitude of additive Gaussian noise. The error is measured in 2D on the image plane. It can be seen that the error stabilizes at a value of approximately 100 pixels. This is due to the fact that when a large amplitude of noise is present, the edge detection fails to find the real edges of the model and instead find random points where noise causes large intensity gradients. Given a high enough noise amplitude this misdetection happens for all of the tracking points. Because of the random nature of the noise, the model is pulled almost equally in all directions and the resulting motion is nullified. The results shown in Figure 41 indicate that with any amount of noise, the algorithm fails.

It can also be seen from Figure 41 that even low noise intensities can cause instabilities in the tracking algorithm. To make the results better, filtering can be applied. The ideal filter to use should be one that removes noise

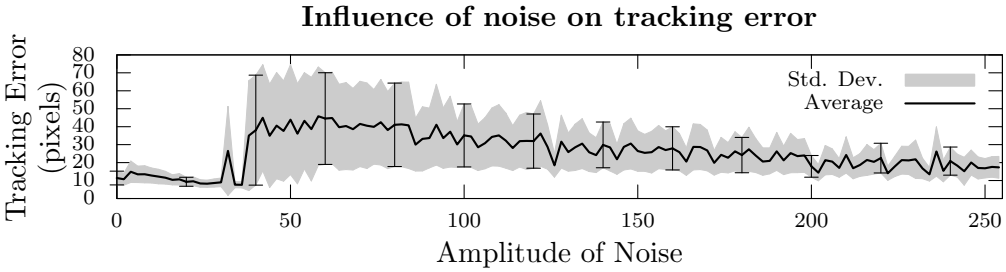


Figure 41: Plot of the average tracking error when the tracker is presented with image sequences containing different amplitudes of noise.

without blurring strong edges. Such an ideal filter can be approximated by the application of a band-pass filter in the frequency domain. A low-pass filter smooths out high frequency noise and edges while a high pass filter accentuates them. The band-pass filter shown in Figure 42 aims to smooth out noise while retaining high intensity gradients around edges.

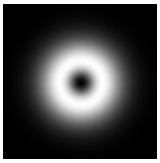
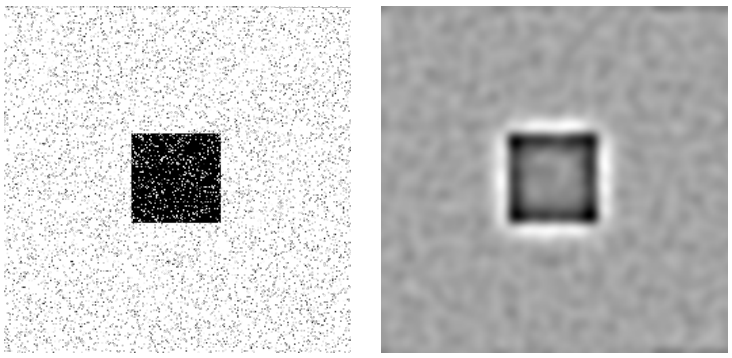


Figure 42: Spatial representation of the bandpass filter used.

Figure 43 gives an example of a noisy image and the resulting image after applying the filter. While the results are good, this filtering comes with an important computational cost. Frequency-based filtering requires the application of two discrete Fourrier transformations to transfer from spatial domain to frequency domain and back. This computational complexity could be minimized by finding an equivalent spatial kernel to be applied to the image via a convolution, but this is outside of the scope of this thesis.

Band-pass filtering does remove some of the noise, but creates stability problems. Tracking results shown in Figure 44 show that the average tracking error increases slightly from that of Figure 41.

Another way of removing the effect of noise is to extract edges prior to the application of the algorithm. This can easily be accomplished by applying horizontal and vertical Sobel filters and combining the result. As can be seen in Figure 45(a and b), the application of such a filter retains some of the noise



(a) (b)

Figure 43: Example of (a) a noisy image and (b) output of bandpass filter.

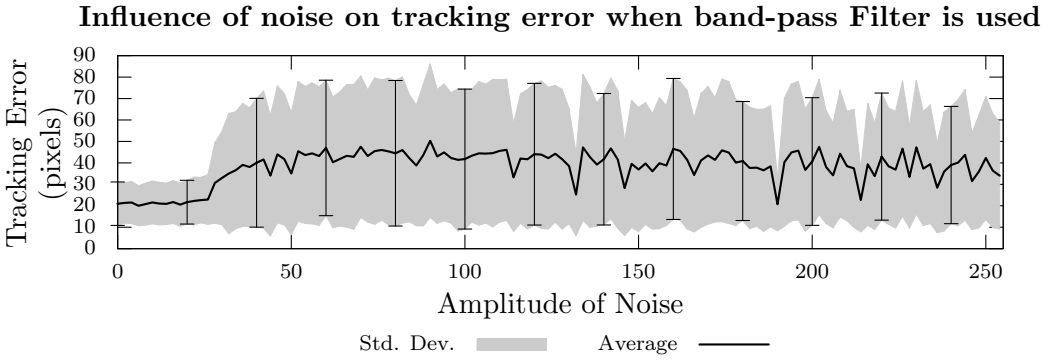
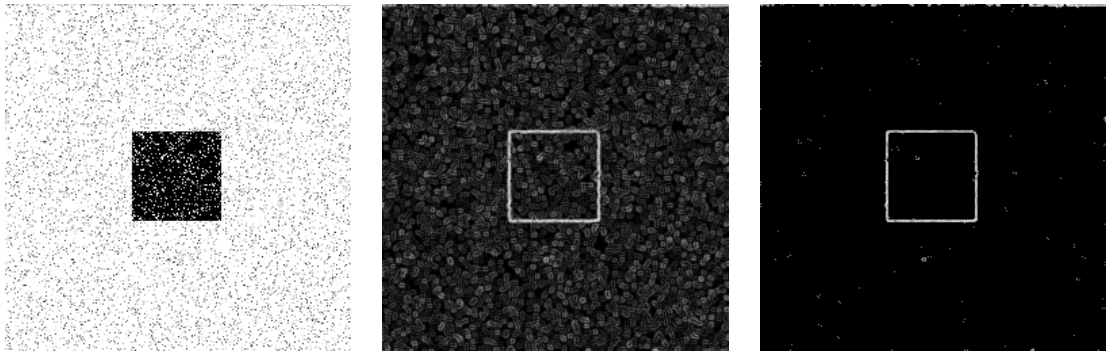


Figure 44: Plot of the average tracking error when the tracker is presented with band-pass filtered image sequences containing different amplitudes of noise.

of the original image. Thresholding the output of the filter as done in 45(c) provides a good approximation of the edges.



(a) (b)

Figure 45: Example of (a) a noisy image, (b) output of Sobel edge detection filter, and (c) thresholded result.

While edges are detected correctly in most cases, the average tracking error also increases when Sobel edge detection is added to the tracker. Figure 46 clearly shows that the tracking performance is not stable even with explicit edge extraction.

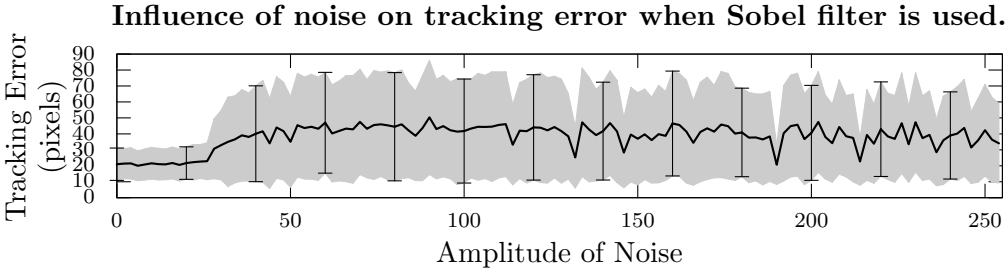


Figure 46: Plot of the average tracking error when the tracker is presented with band-pass filtered image sequences containing different amplitudes of noise.

8.7 Strengths and Weaknesses

The most significant advantage of this tracker with respect to the two-dimensional trackers presented in Sections 5 and 6 is that no modification to the machine is required. A precise model of the tracked object is all that is required. This makes this tracker the less invasive of the three presented.

As shown throughout Section 8.6, the tracker is not stable. The instability has been observed to be mostly due to drift that accumulate over the entire sequence. This tracker is thus not an ideal choice for long term tracking. It can also be seen that the accuracy of this tracker does not compare to the sub-pixel accuracy of the two dimensional trackers of Sections 5 and 6. Finally, the model-based tracker is very sensitive to noise, which makes it a sub-optimal choice for real-world deployment.

9 Simulator Tests

The goal of the simulator is to test the ability of the methods to work in a setting similar to the real world while allowing control over most scene parameters.

By looking at the test results from Sections 5, 6, and 8, it can be seen that the fiducial detector provides a more precise 2D position than the template matching method, which in turn, is better than the model-based tracker. The fiducial detector has also been shown to perform well on sequences containing high amplitudes of noise and large motions. For these reasons, the fiducial marker detector will be used from this point.

The use of a simulator provides the possibility of running experimental tests that would be impractical and potentially quite lengthy in the real world. It also eliminates uncertainty in the measuring tools and thus, provides an almost ideal testing environment.

9.1 Setup Procedure

This section aims to describe all the steps required to create a complete tracking system.

For the first part of the setup, it will be assumed that a single fiducial marker containing the four black circles is placed on the rotary tool of the machine as shown in Figure 47.

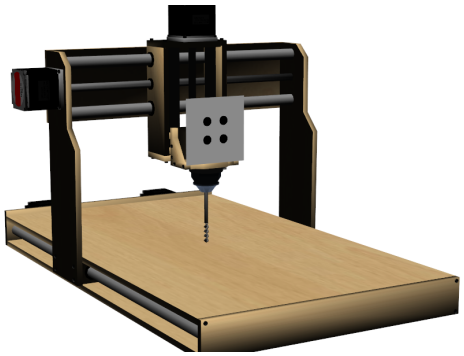


Figure 47: Sample frame showing the position of the fiducial marker on the machine.

A set of frames from different camera positions and orientations is generated to extract positive and negative examples to train the circle detector of Section

5. This step is required for both the fiducial tracker and the camera calibration step. For the simulator tests, the size of the sub-window used by the circle detector has been increased to 16 pixels by 16 pixels to allow slightly larger circles to be found.

9.1.1 Camera Calibration

The virtual camera of the simulator can be calibrated by first generating a sequence of 1000 images containing the calibration image, a sample frame of this sequence is shown in Figure 48(a). The result of applying the detector to a frame of the sequence is shown in Figure 48(b). From these points, the calibration template is found geometrically, as shown in Figure 48(c). Once the calibration template is properly detected, the algorithm detailed in Section 4 is applied to get a calibrated camera model, which allows the drawing of the three Cartesian axes, as done is Figure 48(d).

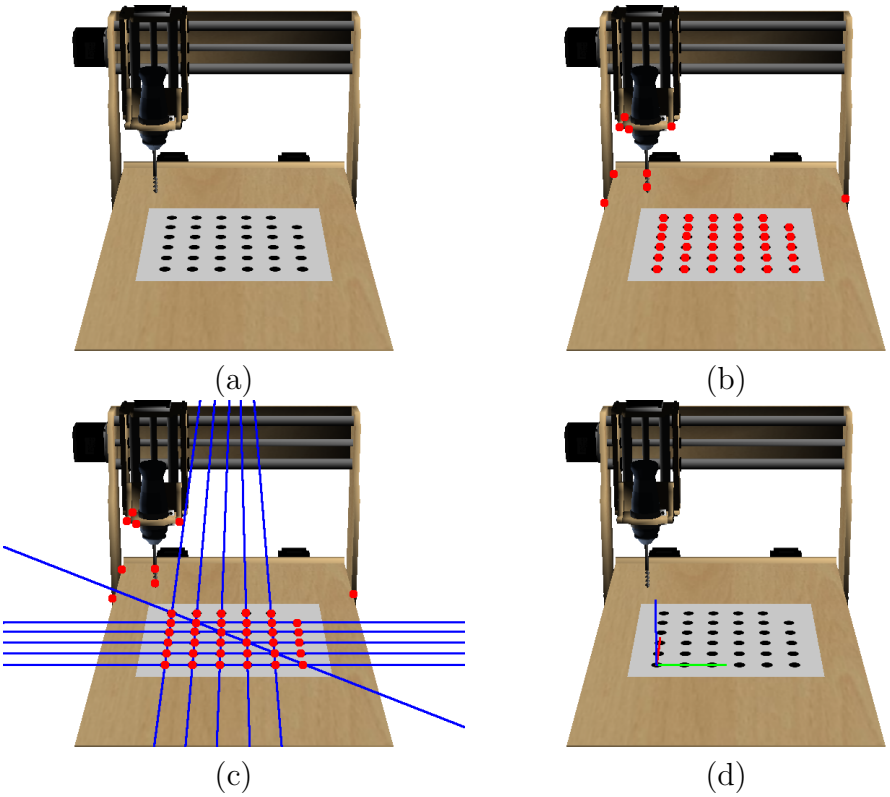


Figure 48: The four steps of camera calibration applied to the simulator: (a) Image containing the calibration template; (b) circle marker extraction; (c) geometric calibration marker detection; and (d) calibrated virtual camera.

It is important to keep in mind that the standard coordinate system used

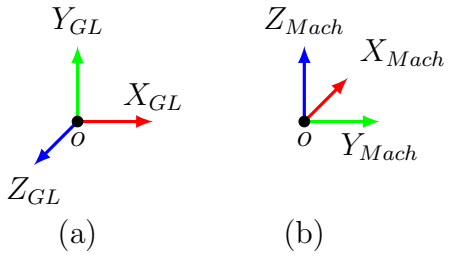


Figure 49: Coordinate System used by OpenGL (a) and the CNC machine (b).

by OpenGL to generate the image of the machine is not the same as the one found when calibrating the camera. To prevent potential confusion, the two coordinate frames are shown in Figure 49. The coordinate frame for CNC machines commonly places the Z axis perpendicular to the ground and the X axis along the longest side of the machine. Figure 49(b) shows the coordinate system used by most CNC machines.

9.1.2 Sensor Planning

For the tracking system to succeed, it is crucial for the camera to be positioned and oriented properly. Sensor planning is a research area within the field of computer vision that aims to find methods to determine sensor parameters to maximize the efficiency of vision algorithms. One approach to find the optimal position and orientation is to generate scenes with different sets of parameters and test the performance of the visual system on them. The metrics used to judge the performance are determined by the problem at hand [31]. In the current case, the optimal camera configuration is one that maximizes the perceived two dimensional movement on the image plane when the machine moves in the three dimensional plane. This configuration will provide the greatest resolution along each of the three axis.

In a similar fashion to [32], spherical coordinates were used for this test. The camera position was assumed to be on the surface of a sphere centered around the machine and the camera was pointed toward the center of the sphere. The radius of the sphere was kept constant while the angles around the Y and Z machine axis were modified. The first point to consider is that under certain extreme camera orientations, the marker might not be detectable. Figure 50 displays a frame in which the marker is barely visible and where the

fiducial detector might fail to produce a proper detection.

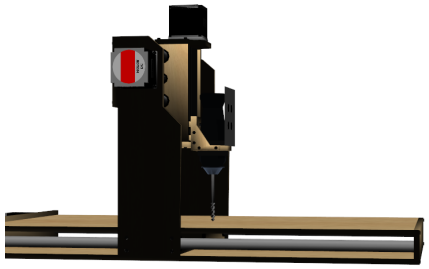


Figure 50: Frame showing that the marker is barely visible.

The first step to determine the optimal camera orientation is thus to find the set of viewing angles that produce images with the marker clearly detectable. To test the visibility of the marker, the camera was moved to different positions on the sphere surface and the fiducial detector was applied to a few hundred frames. The ratio between the number frames where the marker was successfully detected and where the detector failed for each camera orientation is shown in Figure 51.

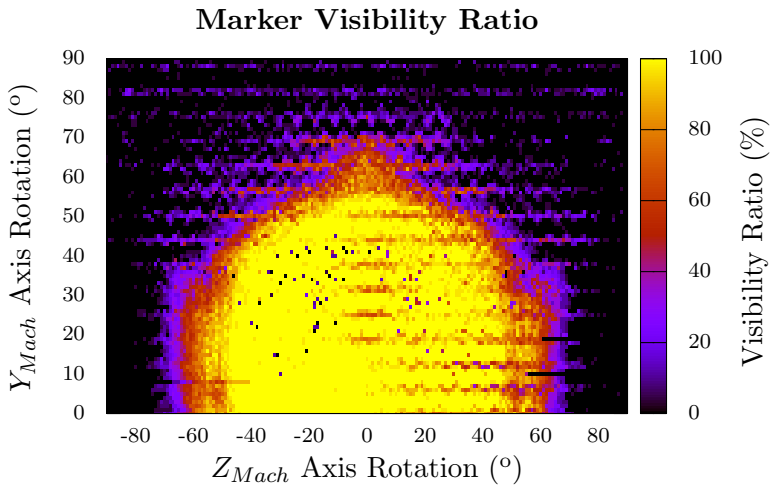


Figure 51: Visibility ratio of the marker with respect to different camera orientations. The scale of the right of the plot relates the shading color to the visibility ratio.

To restrict the search for the optimal settings, the simulator was used to compute the two-dimensional distance that occurs when the marker on the

machine moves by a unit distance, in this case, 1 mm, for different camera orientations. The experiment was executed separately for motion along each of the three axis. The results of this experiment are shown in Figure 52.

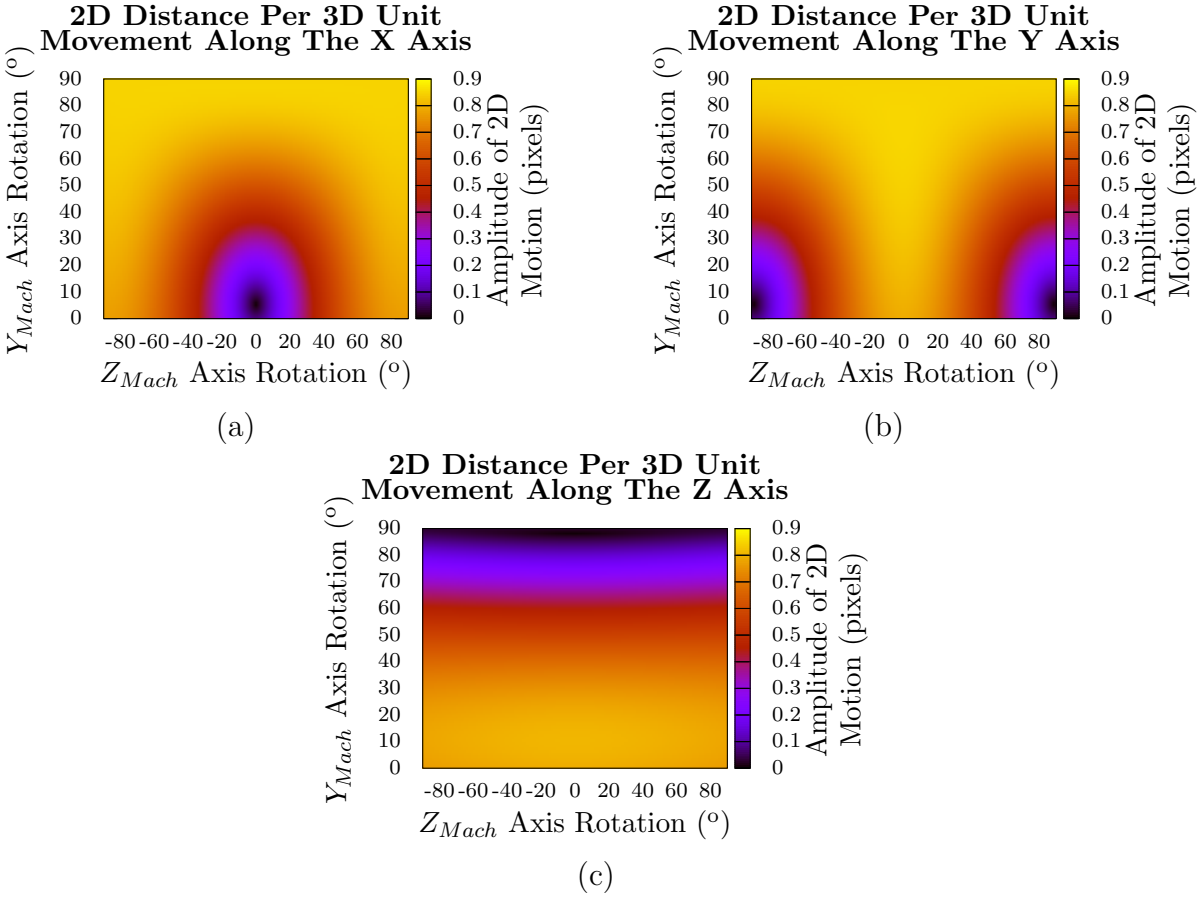


Figure 52: Image representation of the surfaces showing the amount of 2D motion caused by a unit 3D movement along the (a) X machine axis, (b) the Y machine axis, and (c) the Z machine axis. The scales on the right of each plot relate the shading color to the amplitude of 2D motion.

For each camera orientation, a vector $M = [V, M_x, M_y, M_z]^T$ containing the normalized visibility ratio (V) and the normalized amount of motion along each axis (M_x, M_y, M_z) can be constructed from the graphs in Figures 51 and 52. A graph containing the 2-norm $\|M\|$ of each of these vectors can be seen in Figure 53. The maximum of this surface is located at 42 degrees around the Y axis and -33 degrees around the Z axis. This represents the optimal monocular camera configuration.

Now that the orientation of the camera has been found, the distance between the camera and the machine must be determined. It is important to

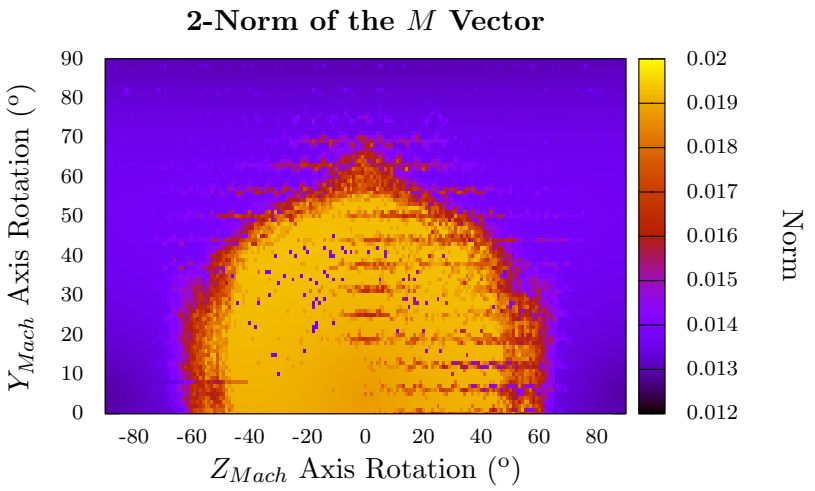


Figure 53: Norm of the M vector for different camera orientations. The scale of the right of the plot relates the shading color to the amplitude of the 2-Norm.

note here that the distance between the camera and the machine is the radius of the sphere on which the camera is moved. The study done here considers a fixed field of view and a varying camera distance. The field of view used for the test is set to a value very similar to that of the physical camera used in Section 10. The combination of field of view and camera distance must be set so that the entire workspace of the machine is visible. Thus, if a fixed field of view is considered, the camera must be placed at a distance allowing the entire machine to be seen. In a similar manner to what was presented earlier, the visibility ratio of the marker for different camera distances is plotted in figure 54, while the plot of the amount of 2D motion generated from a unit 3D displacement is shown in Figure 55.

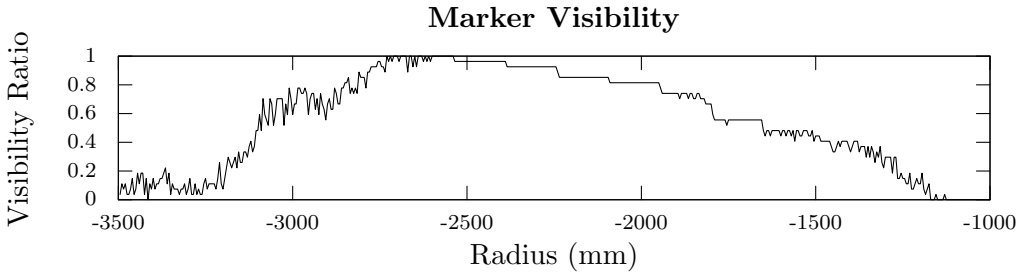


Figure 54: Plot of the visibility ratio for different camera distances.

A good way of embedding the information from both figures into a single

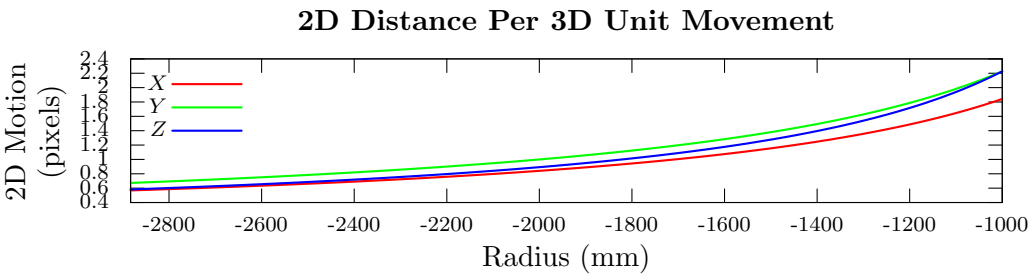


Figure 55: Plot of the amount of 2D motion caused by a unit 3D movement along the three machine axis for different viewing radii.

graph is to multiply the average of the normalized values from Figure 55 with the ratio from Figure 54. The multiplied curve can be seen in Figure 56. The maximum of this curve can be found at a radius of 1948 units (millimeters).

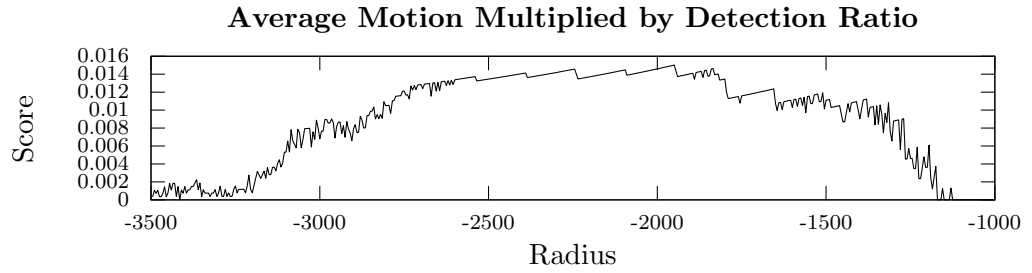


Figure 56: Multiplication of the curve in Figure 54 and the normalized average of the three curves in Figure 55.

Camera resolution must be chosen so that circles fit within the 16 pixels by 16 pixels window used by the circle detector. This restriction can be easily overcome by changing the size of the circles composing the fiducial marker. Figure 57 shows that the amount of perceived 2D motion increases linearly with respect to camera resolution, thus, larger camera resolutions are preferable.

Other camera properties such as focal length and radial distortion will not be tested, but the fiducial detector has been shown to be robust to changes in camera parameters when trained properly.

9.2 Expected Tracking Accuracy

In order to predict the accuracy that can be expected from the complete system, we must consider the two most important accuracy limiting factors.

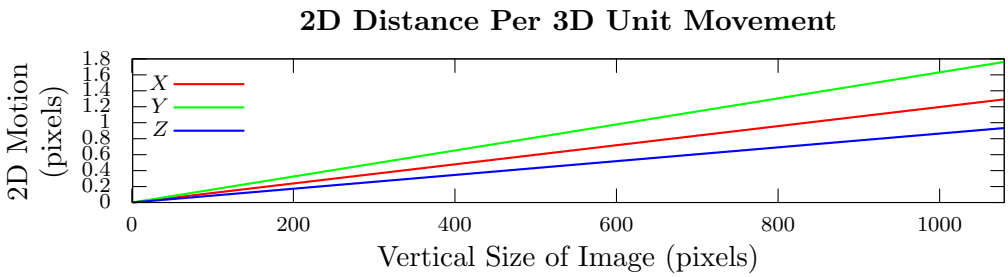


Figure 57: Amount of perceived 2D motion caused by a unit translation in 3D for different camera resolutions. (vertical camera frame dimension is shown)

9.2.1 Fiducial Detector

The first one is the accuracy of the 2D fiducial detector. As seen in Section 5, if less than 10% non-Gaussian noise is assumed to be present in an image, the markers can be found with an error of approximately 0.6 pixels, regardless of the camera resolution. This means that to be properly detectable, a motion of the marker in 3D space needs to generate a shift of at least 0.6 pixels on the image plane.

The shift in the camera image due to a movement in 3D space depends on a few factors. These factors include the amplitude of the 3D movement, the resolution of the camera, the camera position, and the camera field of view. For the current test, the virtual camera will be positioned according to the results of the experiments presented previously in this section. The field of view will also be the same as in the previous experiments. To summarize the setup, the camera is placed quite far from the CNC machine (approximately two meters) and has a small field of view (approximately 16°).

As an example, we can find the camera resolution required to achieve a 3D tracking resolution of 0.01 mm. To do so, the camera projection system used by OpenGL will be used to mathematically determine the required camera resolution. For convenience, a few of the projection equation from the OpenGL documentation have been reproduced here.

$$P = \begin{bmatrix} \frac{\cot(\frac{f}{2})}{a} & 0 & 0 & 0 \\ 0 & \cot(\frac{f}{2}) & 0 & 0 \\ 0 & 0 & \frac{z_f+z_n}{z_n-z_f} & \frac{2\cdot z_f\cdot z_n}{z_n-z_f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{47}$$

The OpenGL projection matrix P has the form shown in Equation (47), where f is the vertical field of view, a is the field of view aspect ratio, z_n is the distance to the near clipping plane of the frustum, and z_f is the distance to the far clipping plane of the frustum. The field of view aspect ratio is taken to be the same as the image field of view. The projection matrix is the OpenGL equivalent to a real camera intrinsic parameters matrix.

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{48}$$

The modelview matrix M encodes the position and orientation of the scene relative to the camera. It is the OpenGL equivalent of a real camera's extrinsic parameters matrix. This matrix, whose form is shown in Equation (48) contains the 3x3 rotation matrix $\mathbf{R}$ and the 3-dimensional translation vector $\mathbf{T}$.

$$u = F_x \cdot \frac{\mathbf{v}'_0 + 1}{\mathbf{v}'_2 + 1} \tag{49}$$

$$v = F_y \cdot \frac{\mathbf{v}'_1 + 1}{\mathbf{v}'_2 + 1} \tag{50}$$

Now, to project a point in space represented by $\mathbf{v} = [x, y, z, 1]^T$ to a set of screen coordinates of the form $\mathbf{w} = [u, v, 1]^T$, the two transformation matrices P and M are applied to $\mathbf{v}$ to get $\mathbf{v}' = PM\mathbf{v}$. The u and v components of the screen coordinates can then found by applying Equations (49) and (50), where F_x and F_y are the width and height of the camera image (OpenGL window), respectively.

$$\mathbf{w}_m = \left[F_x \cdot \frac{\mathbf{v}'_{m0} + 1}{\mathbf{v}'_{m2} + 1}, F_y \cdot \frac{\mathbf{v}'_{m1} + 1}{\mathbf{v}'_{m2} + 1}, 1 \right] \tag{51}$$

At this point, the projection from 3D world coordinates to 2D screen coordinates has been defined. It is now possible to find a relation between a motion in 3D space and the associated shift in the 2D image. Given a three-dimensional displacement $\Delta\mathbf{v}$, the new position of the point $\mathbf{v}_m$ is equal to $\mathbf{v} + \Delta\mathbf{v}$. The new position of the 2D image has the form shown in Equation (51).

$$\Delta u = \left| F_x \cdot \left(\frac{\mathbf{v}'_{m0} + 1}{\mathbf{v}'_{m2} + 1} - \frac{\mathbf{v}'_0 + 1}{\mathbf{v}'_2 + 1} \right) \right| \tag{52}$$

$$\Delta v = \left| F_y \cdot \left(\frac{\mathbf{v}'_{m1} + 1}{\mathbf{v}'_{m2} + 1} - \frac{\mathbf{v}'_1 + 1}{\mathbf{v}'_2 + 1} \right) \right| \tag{53}$$

$$\|\Delta \mathbf{w}\| = \sqrt{(\Delta u)^2 + (\Delta v)^2} \tag{54}$$

The amplitudes of the horizontal and vertical shifts on the image plane caused by the 3D motion are shown in Equations (52) and (53), respectively. The resulting amplitude of the 2D shift can be computed using Equation (54).

To obtain the tracking resolution stated earlier, a motion of 0.01 mm will need to cause a shift of the marker on the 2D image of at least 0.6 pixel. The 3D motion of 0.01 mm can be encoded as $\|\Delta \mathbf{v}\| = 0.01$ and the required 2D shift can be written as $\|\Delta \mathbf{w}\| = 0.6$.

Because of the assumption that the projection and model-view matrices are known and kept constant, there are eight unknowns: the horizontal and vertical resolutions (F_x and F_y) of the camera, the components of the initial position $\mathbf{v} = [x, y, z, 1]^T$, and the components of the motion vector $\Delta \mathbf{v} = [\Delta x, \Delta y, \Delta z, 0]^T$. If the aspect ratio of the camera is assumed to be known, then $F_x = a \cdot F_y$ where $a = 4/3$ in most "fullscreen" cameras.

$$0.6^2 = F_y^2 \cdot \left[\frac{16}{9} \cdot \left(\frac{\mathbf{v}'_{m0} + 1}{\mathbf{v}'_{m2} + 1} - \frac{\mathbf{v}'_0 + 1}{\mathbf{v}'_2 + 1} \right)^2 + \left(\frac{\mathbf{v}'_{m1} + 1}{\mathbf{v}'_{m2} + 1} - \frac{\mathbf{v}'_1 + 1}{\mathbf{v}'_2 + 1} \right)^2 \right] \tag{55}$$

Another simplifying assumption that can be made is that the initial position is $\mathbf{v} = [0, 0, 0, 1]^T$. For the current projection and modelview matrices, this means $\mathbf{v}' = [0.0000, 0.0000, 0.9967, 1.0000]^T$. The system to solve can be simplified to the form shown in Equation (55).

The only missing component needed to solve the system is information about the 3D displacement. The norm of the displacement vector is known, but the direction is not. A quick solution to this problem is to try different displacement directions and compute the average required frame resolution.

The plot shown in Figure 58 can be generated by varying the desired 3D resolution and computing the required vertical camera resolution.

For the example considered here, if a 3D resolution of 0.01 mm is required, a camera resolution of 81616 pixels by 61212 pixels (approximately 5000 mega pixels) would be necessary. By today's standard, with the fiducial tracker presented in this thesis, a tracking resolution of 0.01 millimeter is not feasible.

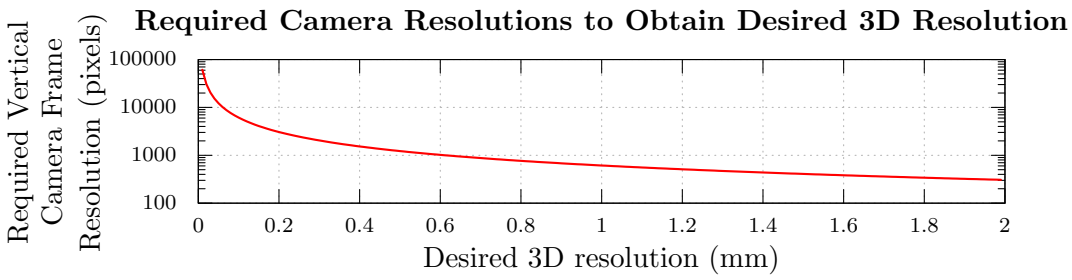


Figure 58: Plot of the required vertical camera resolution for different 3D resolutions

However, the plot in Figure 58 shows quite clearly that the required camera resolution drops rapidly when the 3D resolution decreases. The plot can also be used to predict the 3D resolution for any given camera resolution. Table 3 shows the expected 3D resolution for common 4:3 frame sizes . These values are only valid for the projection matrices used for this test, but they can be generated for any camera system by changing the projection matrices.

Name	Resolution (pixels)	Maximum 3D Resolution (mm)
QXGA	2048 × 1536	0.40
UXGA	1600 × 1200	0.51
SXGA+	1400 × 1050	0.58
XGA	1024 × 768	0.80
SVGA	800 × 600	1.02
VGA	640 × 480	1.27

Table 3: Table of common 4:3 camera resolutions and the 3D resolution expected based on the fiducial detector

9.2.2 Conversion between 2D and 3D coordinates

The second accuracy limiting factor is the accuracy of the process used to convert 2D coordinates into 3D coordinates. As seen in Section 7, if uncertainty is present in the 2D positions of the four corners, the computed 3D position is not exact. If we again assume that the 2D position of the fiducial has an error of 0.6 pixels and that all camera properties other than resolution remain constant, the error caused by a 0.6 pixel shift can be computed using the same method as in Section 7. Instead of computing the error for different amplitudes

of the noise, different camera resolutions will be used. As the direction of the shift caused by the 0.6 pixel shift is unknown, the error is computed for 100 different 2D shifts of amplitude 0.6 pixels. The results of this test are shown in Figure 59.

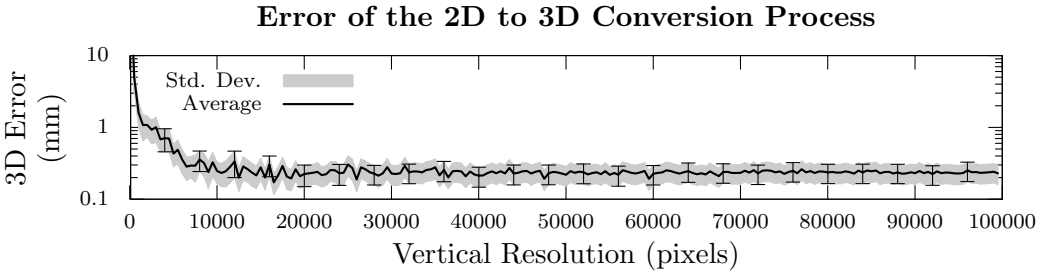


Figure 59: Plot of the 2D to 3D conversion error for different camera frame resolutions

The noteworthy part of Figure 59 is that the curve settles to a value of 0.23 mm. Even if the camera resolution increases, the inaccuracies in the fiducial detector limit the 3D resolution to 0.23 mm. The maximum theoretical average 3D resolution achievable with the tracking system detailed in this thesis using the camera configuration presented in this section is thus 0.23 mm. With high resolutions, the standard deviation, also shown in Figure 59, settles to a value of approximately 0.07 mm. Table 4 presents the average 3D error expected for a few common frame sizes.

Name	Resolution (pixels)	Average 3D Error (mm)	Standard Deviation (mm)
QXGA	2048 × 1536	0.76	0.29
UXGA	1600 × 1200	0.98	0.37
SXGA+	1400 × 1050	1.12	0.43
XGA	1024 × 768	1.56	0.59
SVGA	800 × 600	2.02	0.75
VGA	640 × 480	2.55	0.94

Table 4: Table of common 4:3 camera resolutions and the expected average 3D error expected and the standard deviation of the error based on the 2D to 3D conversion

It is important to note that all of these prediction do not take into consideration potential inaccuracies in the camera model due to the calibration. For

this reason, the errors that will be obtained in the following tests will most likely be higher.

As most of the error comes from random sources such as noise, it is difficult to predict the exact resolution of the system. However, the analysis done in this section provides an upper bound on the resolution of the tracking system.

9.3 Tracking Tests

The visual simulator will now be used to provide a better approximation of the performance of the system applied to real camera input. To do so, three tracking sequences, each containing 1000 frames, will be generated for three different simulated camera resolutions. The three sequences show a machine travelling along each of the three machine axis. While generating the sequences, the true position of the machine is recorded in a file. Applying the fiducial tracker to each sequence and computing the distance between the detected machine position and the position recorded in the ground Truth file allows the creation of Table 5. The execution time per frame will also be recorded to provide an idea of the frame rate that can be expected when running the tracker. The execution time is recorded in Table 6.

Table 5 clearly demonstrates that as the frame resolution increases, both the mean error and the standard deviation of the error decrease. While the errors are higher than the theoretical minimum, these findings are consistent with the results found in Section 9.2. The difference between the current results and those of Section 9.2 can be attributed mostly to uncertainties in the camera calibration. Section 9.2 assumed a perfect camera calibration, while the current experiment was executed with the use of the camera calibration algorithm presented in Section 4. This algorithm, while found to be adequate, is limited in precision by the use of the circle detector and the parameter optimization stage, which both introduce slight inaccuracies. Table 6 shows that higher resolutions require higher execution times per frame. The perfect balance between tracking error and frame rate has to be determined based on the action being tracked.

9.4 Concluding Remarks

The visual simulator has proven to be an efficient sensor planning tool to determine the parameters of the vision system, such as camera position and

resolution. It was also able to provide a theoretical limit of the accuracy of the system. Finally, the overall tracker has been executed on simulated sequence to provide an estimate of the accuracy on real image sequences.

Measured errors

Resolution	Sequence	Minimum (mm)	Maximum (mm)	Mean (mm)	Std. Dev. (mm)
640×480	X Motion	00.38	36.57	11.45	06.51
	Y Motion	00.29	40.31	13.32	07.29
	Z Motion	00.19	27.14	09.38	05.88
800×600	X Motion	00.22	18.95	11.03	04.74
	Y Motion	00.29	22.76	10.05	05.54
	Z Motion	00.31	17.34	08.00	04.10
1280×960	X Motion	00.47	16.95	06.54	03.14
	Y Motion	00.12	16.29	06.28	03.19
	Z Motion	00.13	14.68	05.85	02.63

Table 5: Statistics concerning the distance between real position and detected position measured over sequences of 1000 frames

Measured Execution Time Per Frame

Resolution	Sequence	Minimum (ms)	Maximum (ms)	Mean (ms)	Std. Dev. (ms)
640×480	X Motion	50	170	67	09
	Y Motion	50	170	65	09
	Z Motion	40	160	61	08
800×600	X Motion	80	170	91	12
	Y Motion	80	180	93	14
	Z Motion	80	190	93	16
1280×960	X Motion	290	420	318	11
	Y Motion	290	430	316	10
	Z Motion	300	430	328	11

Table 6: Statistics concerning the computation time per frame measured over sequences of 1000 frames

Overall, this section has shown that, in theory, the sub-pixel accuracy of the fiducial detector limits the accuracy of the overall system and that, in a real implementation, the combination of camera calibration and 2D to 3D conversion is the limiting factor.

10 Results

10.1 Experimental Setup

The platform used for the real world experiments is a FireBall V90 CNC Router from Probotix, shown in Figure 10.1.

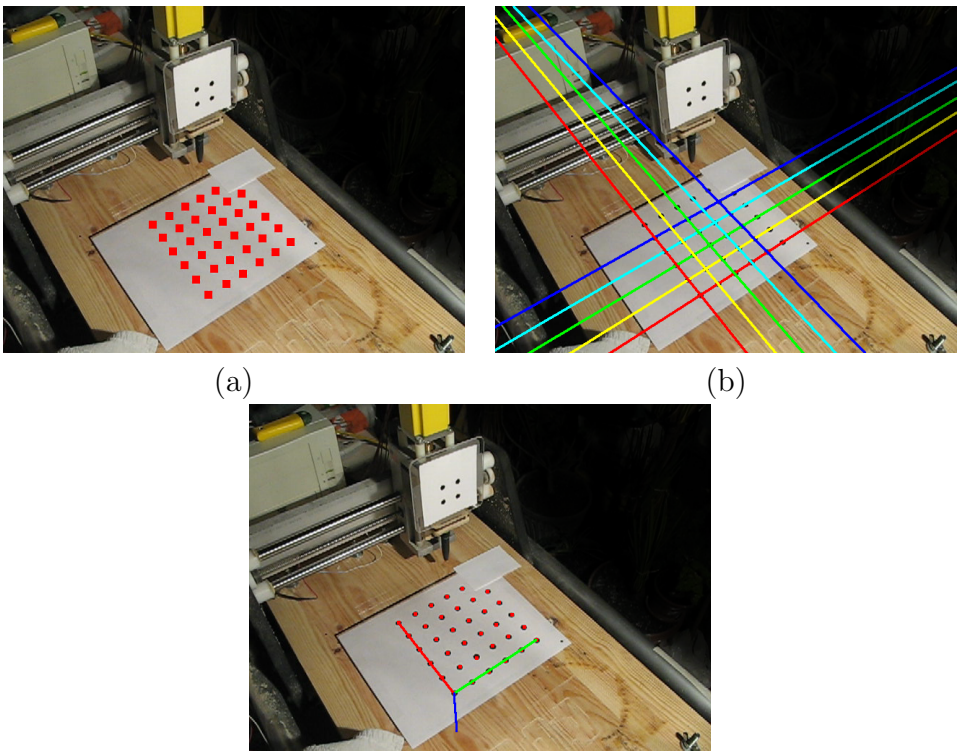
Figure 60: Physical machine used for the last stage of testing. source: `http://www.probotix.com/`

The obvious problem when testing a tracking algorithm on a physical machine is the limited ability of obtaining accurate measurements of the machine to compare with the output of the tracker. This can be solved by using a pen plotting attachment and drawing points on a piece of paper located under the machine. While using a pen is a valid solution, it only records the position of the machine on the X and Y axis. This means that the accuracy of measurements along the Z axis will be limited. As seen in previous sections, the 2D to 3D conversion stage optimizes on the X axis and computes the Y and Z position from the X value found. The simulator has demonstrated that given a proper camera calibration, this step is successful. For this reason, the current tests will limit the motion of the machine to the X and Y axis.

10.2 Tests

Two distinct tests will be done to determine the performance of the visual tracker. First, a qualitative test will be used to obtain a general idea of the tracking system. The second test will provide measurements of the accuracy

of the tracker. For each of the two tests, the camera will be positioned as close as possible to the optimal position found in Section 9 and calibrated using the method detailed in Section 4. Figure 61 shows some frames from the calibration step. No discussion of calibration will be made here as it has been covered in previous sections.



(a) (b)

(c)

Figure 61: Camera calibration for the physical tests: (a) circle marker extraction; (b) geometric calibration marker detection; and (c) calibrated camera.

For convenience, the camera will record image sequences of the machine and the tracking will be done offline. The execution time required per frame will, however, be measured to ensure that the algorithm is capable of running in real time.

10.2.1 Qualitative Test

This first test involves tracking the position of the machine while it draws a known geometric shape. This provides the opportunity to qualitatively compare the drawing from the machine to the output of the tracker. The drawing is generated by the pen plotter attachment and provides a record of the posi-

tions visited by the machine. The output of the tracker along the X and Y machine axis can be represented as a scatter plot to obtain an image of what the tracker thinks the machine drew.

The geometric shape chosen is an Archimedean spiral. This shape provides a single continuous pen stroke which reduces the amount of travel on the Z axis.

The first test uses a video sequence with a resolution of 640 pixels by 480 pixels. According to the simulator tests executed earlier, this resolution should provide crude tracking results but should demonstrate the general functionality of the system.

Figure 62(a) shows the drawing generated by the machine while Figure 62(b) shows that the tracker was able to track the position of the machine accurately enough to determine that the machine drew a spiral. The green data points in Figure 62(b) represent the raw output of the tracker while the red points are the output of the Kalman filters.

It can be seen that when the machine moves along the Z axis, the X and Y values are slightly affected. These Z axis motions occur at discontinuities in the drawing that can be found at around $(-20, 130)$, $(20, 15)$, and $(40, 70)$.

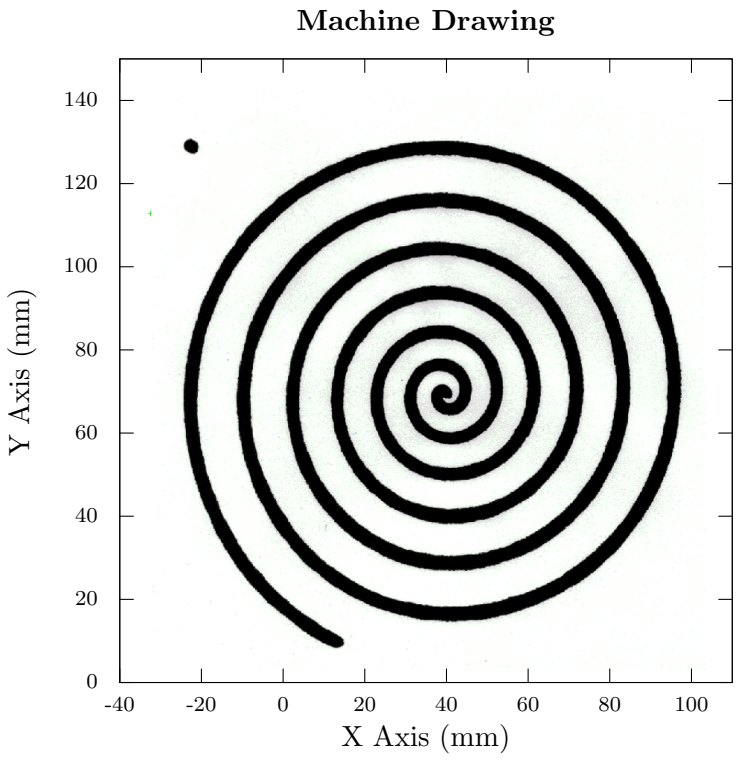
The time required to apply the fiducial detector, the two-dimension-to-three-dimension conversion, and the filtering has been measured to to be 0.083980 second. This translates to roughly 12 frames per second, which is similar to what was measured with the help of the simulator.

Overall, this first test shows that the tracker behaves as expected from the tests on the simulator. The next tests will provide a better estimate of the performance of the tracker.

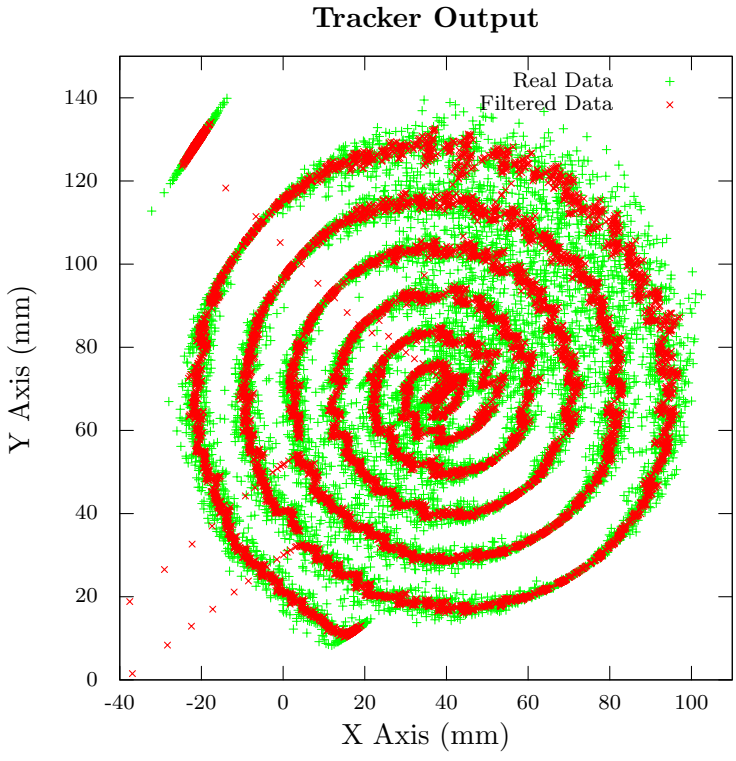
10.2.2 Quantitative Test

For the second test, the machine is manually moved to different positions on a regular grid. At each position, the machine will be instructed to draw a point on a piece of paper and the camera will take a series of ten photos of the machine. The piece of paper on which the machine has drawn the points will then be scanned at a high resolution to allow precise measurements of the positions of the points. The tracker will be executed on each set of ten photos and its output will be compared with the measured points on the piece of paper to characterize the error of the tracker.

The camera used for this test is a Canon Powershot A630, which has a



(a)



(b)

Figure 62: Machine drawing (a) and tracker output (b) for a sequence recorded at 640x480.

native sensor resolution of 3264 pixels by 2448 pixels. As this resolution is very large, captured images were resized to a resolution of to 1632 by 1224, which is a slightly higher resolution than the 1280 by 960 used with the simulator, but the aspect ratio remains the same.

If we refer to Figure 59, we can determine that the expected maximum resolution, based solely on the error of the 2D-to-3D conversion, at a horizontal image resolution of 1632 pixels is lower than 1 mm (0.67 mm to be exact). As we are working with real data, the detected position of the fiducial corners will most likely not be exact and we can thus refer to Figure 31 to determine that the error caused by the 2D-to-3D conversion can range from approximately 0.5 mm in the best case to approximately 3 mm if the fiducial detector is off by 0.6 pixels in the 2D image. Overall, this means that the expected average error for the current test should be in the range of 1.5 mm to 4 mm. The expected average error can also be extrapolated from the results of the simulator. At the current resolution, an average error smaller than 2.95 mm and a standard deviation smaller than 1.61 mm are expected.

The piece of paper was scanned at a resolution of 600 dots per inches after drawing approximately 140 points. The resulting image can be seen in Figure 63.

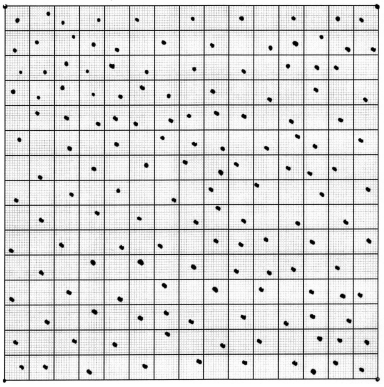


Figure 63: Scan of the points drawn by the machine (scaled)

The points in Figure 63 can be measured accurately by noting that a single pixel on a 600 dpi scan is equivalent to 0.0423 mm, which far exceeds the expected three dimensional resolution of the tracker. Once measured, both these points and the output of the tracking algorithm can be plotted to visually inspect the error. This plot is provided in Figure 64.

The full table of measurements can be found in the appendix. To compare

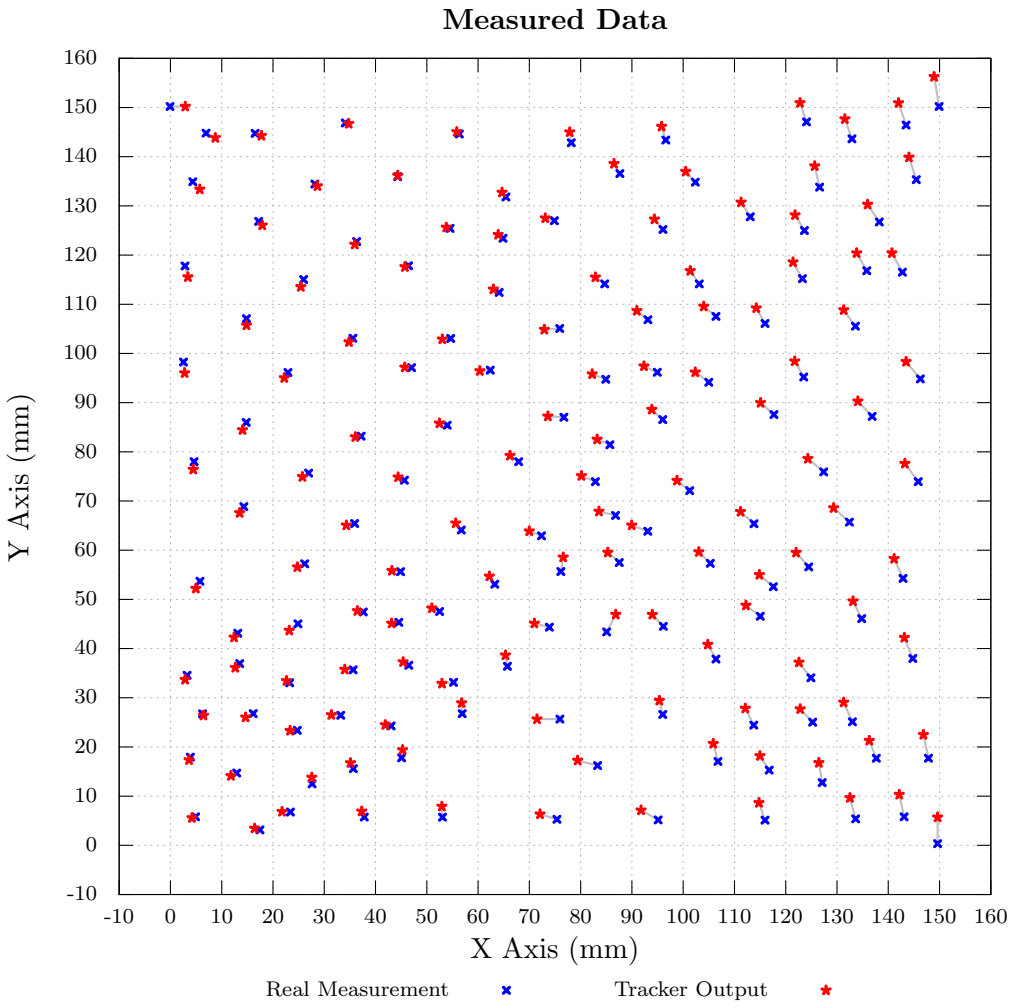


Figure 64: Plot of the points measured from the scanned image (blue) and the output of the tracker (red).

these results with those from the simulator, statistical information about the last column of the results table, containing the Euclidean norm of the error, can be extracted. All errors lie between 0.26 mm and 6.11 mm. The error is found to be 2.59 mm on average with a standard deviation of 1.30 mm. Both the average error and the standard deviation are slightly smaller than that of the simulator at a resolution of 1280 by 960, as expected.

11 Conclusion

This thesis has presented three different approaches to tracking objects moving in three dimensional space. Fiducial marker tracking has been shown to provide sub-pixel accuracy and robustness to noise and fast motion. The template tracking algorithm, while being computationally simpler, fails to track objects that move fast and is prone to drift. Finally, the model-based tracker implemented has been found to be finicky with respect to the initial pose estimate and also suffers from drift problems. After characterizing the three tracking algorithms, the fiducial marker tracker was chosen as the most reliable and accurate method of the three for the task of CNC machine tracking.

The two tests performed in Section 10 have shown that the tracking accuracy of the real world system is consistent with the predictions and results obtained from the simulator in Section 9. This demonstrates that using a visual simulator is a valid means of testing computer vision algorithms before physical implementation and that the fiducial tracker can indeed be applied to the task of tracking a CNC machine. The previous two sections have also confirmed that the accuracy of the tracking system increases with resolution. The theoretical accuracy of the tracker was shown to be limited by the inaccuracy of the fiducial detector. For practical applications, the slight inaccuracies of the camera calibration model are shown to cause most of the tracking error.

Even if the tracker has inherent sources of error, Figure 58 indicates that, in theory, any required accuracy should be achievable by either using a larger image resolution or, inversely, using a zoom lens to restrict the field of view of the camera, which would increase the amplitude of the 2D shift on the image plane when the tracked object moves in 3D space. In order to obtain substantial benefit from a narrower field of view, the entire 3D volume in which the marker can travel would most likely not be visible. This means that further development of the tracking method presented in this thesis would require either the use of an array of cameras, each looking at a different part of the 3D space, or a steerable camera that can be controlled to keep the marker in its field of view.

Overall, this thesis has demonstrated that computer vision is a reliable source of information for CNC machine controllers depending on the machining task being executed. The system shown is accurate enough to track an object down to a few millimeters, which is adequate in some low precision

machining tasks. The accuracy is also enough to detect large deviations of the machine's end effector from its predicted position, due to motor failure or other unforeseen events.

References

[1] X. W. Xu and S. T. Newman. Making cnc machine tools more open, interoperable and intelligent: a review of the technologies. *Computers in Industry*, 57:141–152, February 2006.

[2] F. Zhao, X. Xu, and S. Q. Xie. Survey paper: Computer-aided inspection planning-the state of the art. *Computers in Industry*, 60:453–466, September 2009.

[3] J. Zhang, S. K. Ong, and A. Y. C. Nee. Development of an ar system achieving in situ machining simulation on a 3-axis cnc machine. *Computer Animation and Virtual Worlds*, 21:103–115, March 2010.

[4] D. Kragic and H. Christensen. Survey on visual servoing for manipulation. Technical report, Computational Vision And Active Perception Laboratory, 2002.

[5] E. Marchand and P. Bouthemy. A 2d-3d model-based approach to real-time visual tracking. *IVC*, 19:941–955, 2001.

[6] D.G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:441–450, 1991.

[7] T. Drummond and R. Cipolla. Real-time tracking of complex structures with on-line camera calibration. In *Proceedings of British Machine Vision Conference (BMVC'99*, pages 574–583, 1999.

[8] J.S. Philippe, G. Dudek, and C. Prahacs. A visual servoing system for an aquatic swimming robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[9] J. Okamoto and V. Grassi. Visual servo control of a mobile robot using omnidirectional vision. 2002.

[10] N. Andreff, B. Espiau, and R. Horaud. Visual servoing from lines, 2000.

[11] R. Y. Tsai. Radiometry. chapter A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses, pages 221–244. Jones and Bartlett Publishers, Inc., , USA, 1992.

[12] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62 –66, jan. 1979.

[13] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[14] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[15] M. Fiala. Artag, a fiducial marker system using digital techniques. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 590–596. IEEE Computer Society, 2005.

[16] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. *International Workshop on Augmented Reality*, 0, 1999.

[17] L. Naimark and E. Foxlin. Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. *IEEE / ACM International Symposium on Mixed and Augmented Reality*, 0, 2002.

[18] G. Jiang and L. Quan. Detection of concentric circles for camera calibration. *IEEE International Conference on Computer Vision*, 1:333–340, 2005.

[19] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects: A survey. In *Foundations and Trends in Computer Graphics and Vision*, pages 1–89, 2005.

[20] D. Claus and A. Fitzgibbon. Reliable fiducial detection in natural scenes. In *Computer Vision - ECCV 2004*, volume 3024 of *Lecture Notes in Computer Science*, pages 469–480. Springer Berlin / Heidelberg, 2004.

[21] D. Claus and A. Fitzgibbon. Reliable automatic calibration of a marker-based position tracking system. *IEEE Workshop on Applications of Computer Vision and the IEEE Workshop on Motion and Video Computing*, 1:300–305, 2005.

[22] P. Hart. The condensed nearest neighbor rule. *Information Theory, IEEE Transactions on*, 14(3):515 – 516, may 1968.

[23] B.K.P. Horn. *Robot Vision*. The MIT Press, Cambridge, MA, USA, 1998.

[24] M. Rea, D. McRobbie, H. Elhawary, Z. Tse, M. Lamperth, and I. Young. Sub-pixel localisation of passive micro-coil fiducial markers in interventional mri. *Magnetic Resonance Materials in Physics, Biology and Medicine*, 22:71–76, 2009.

[25] F. Jurie and D. Dhome. A simple and efficient template matching algorithm. *IEEE International Conference on Computer Vision*, 2, 2001.

[26] F. Jurie and M. Dhome. Real time robust template matching. In *in British Machine Vision Conference 2002*, pages 123–131, 2002.

[27] K. Satoh, S. Uchiyama, and H. Yamamoto. A head tracking method using bird's-eye view camera and gyroscope. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '04, Washington, DC, USA, 2004. IEEE Computer Society.

[28] S. You and U. Neumann. Fusion of vision and gyro tracking for robust augmented reality registration. In *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, VR '01, Washington, DC, USA, 2001. IEEE Computer Society.

[29] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:932–946, July 2002.

[30] G. Klein and T. Drummond. Robust visual tracking for non-instrumented augmented reality. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '03, pages 113–, Washington, DC, USA, 2003. IEEE Computer Society.

[31] K. A. Tarabanis, P. K. Allen, and R. Y. Tsai. A survey of sensor planning in computer vision. *IEEE Transactions on Robotics and Automation*, 11(1):86–104, 1995.

[32] S. Sakane, M. Ish, and M. Kakikura. Occlusion avoidance of visual sensors based on a hand-eye action simulator system: Heaven. *Advanced Robotics*, 2(2):149–165, 1987.

[33] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.

12 Appendix

12.1 Linear Kalman Filter Implementation

The Kalman filter implementation done for this thesis is based on information from [33]. Please note that the notation in this section is unrelated to that of the rest of the thesis. Kalman filters are used in this thesis as a tool to reduce the effect of noise over multiple frames. To simplify the implementation, the data being filtered are assumed to be linear or at the very least locally linear, and thus, linear Kalman filters are used. The current section will detail the implementation of these filters.

$$X_k = AX_{k-1} + w \tag{56}$$

$$z_k = HX_k + v \tag{57}$$

To track the evolution of the a value x, a state vector X is constructed as $(x, \dot{x})^T$. Physically, x represents the position and $\dot{x}$ respresents the rate of change, or velocity, of x. The underlying system governing the value of the vector X is assumed to be of the form shown in Equation (56) and the measurements z have the form shown in Equation (57) , where v and w are noise components with respective normal distributions $\mathcal{N}(0, R)$ and $\mathcal{N}(0, Q)$. The A term is the state transition matrix and thus represents the expected relation between the previous value of X and its current value. The H term represents the relation between the measured component of X and its full form.

$$A = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \ , \ H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{58}$$

As we are only measuring the position component x of X, the forms of H is given by Equation (58). The position depends on the velocity with respect to time, thus the t term in the state transition matrix shown in Equation (58). The value of t is the time interval between measurement, that is, the inverse of the frame rate of the image sequence.

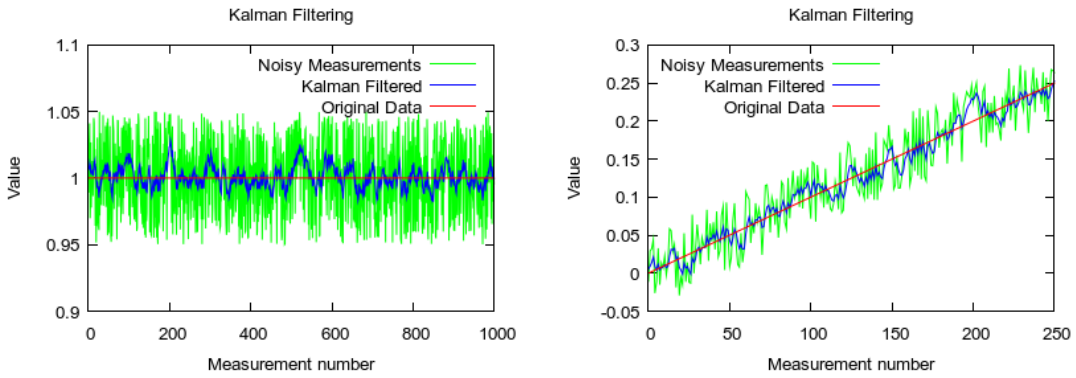
The noise covariance matrices Q and R were assumed to be diagonal to simplify computations. The values chosen for the diagonal of the Q matrix were small as it is assumed that the system has a small amount of noise. It

was assumed that the measurement process (camera) introduced most of noise and thus the values chosen for the R matrix were large. The exact values of $Q_{i,i}$ and $R_{i,i}$ were chosen based on the properties of the data to be filtered.

$$\hat{X}_k^- = A\hat{X}_{k-1} \tag{59}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{60}$$

The Kalman filtering method is applied as a two step process. The first step is to update the a priori state approximation of X, as shown in Equation (59), and the covariance matrix of the a priori state approximation error, as shown in Equation (60).



(a) Constant valued input

(b) Constant slope input

Figure 65: Output of a Kalman filter (blue) when presented with the noisy measurement (green) of a linear system (in red)

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{61}$$

$$\hat{X}_k = \hat{X}_k^- + K_k(z_k - H\hat{X}_k^-) \tag{62}$$

$$P_k = (I - K_k H)P_k^- \tag{63}$$

The second step is to compute the a posteriori state approximation, shown in Equation (62), and the covariance matrix of the a posteriori state approximation error, shown in Equation (63). These two equations require the computation of the Kalman gain matrix, as displayed in Equation (61).

	Noisy	Filtered	Improvement
Constant Value	0.024698	0.006366	74.22%
Constant Slope	0.024698	0.009415	61.88%

Figure 66: Measurement of the average absolute value of the distance between the true value and the two other curves.

Once these computations are done, the current state a posteriori estimation contains the filtered value of X.

The figures in 65 show an example of the reduction in the amount of noise when a Kalman filter is applied to noisy measurements. Table 66 shows the average absolute value of the distance between the true value and the two other curves (the noisy measurements and the filtered results). This average is computed over 1000 measurements even if Figure 65(b) only shows the values from 0 to 250.

12.2 Table of Measurements

Measured		Tracker		Error		
X (mm)	Y (mm)	X (mm)	Y (mm)	X (mm)	Y (mm)	Norm (mm)
0.1270	0.3387	2.87816	150.2117	2.7935	0.0000	2.7935
−0.0847	150.2117	148.94544	156.24651	−0.9699	6.0348	6.1123
149.9153	150.2117	149.619	5.70885	0.0000	5.3702	5.3702
149.6190	0.3387	4.20224	5.57637	−0.7084	−0.2233	0.7428
4.9107	5.7997	16.40466	3.44433	−1.0460	0.3117	1.0915
17.4507	3.1327	3.62021	17.3544	−0.2745	−0.5619	0.6254
3.8947	17.9163	11.78379	14.14499	−1.1372	−0.5540	1.2650
12.9210	14.6990	21.77412	6.84237	−1.6125	0.0690	1.6140
23.3867	6.7733	27.56774	13.81144	−0.0523	1.3138	1.3148
27.6200	12.4977	23.33744	23.31519	−1.4039	−0.0715	1.4057
24.7413	23.3867	14.66734	26.0354	−1.4710	−0.7379	1.6457
16.1383	26.7733	6.43641	26.41387	0.1711	−0.3171	0.3603
6.2653	26.7310	2.85046	33.68918	−0.4092	−0.8828	0.9730
3.2597	34.5720	12.65187	36.12379	−0.8618	−0.8189	1.1888
13.5137	36.9427	22.62228	33.44542	−0.6374	0.3974	0.7511
23.2597	33.0480	33.99693	35.74457	−1.6334	0.0719	1.6350
35.6303	35.6727	31.38155	26.52961	−1.8358	0.0949	1.8382
33.2173	26.4347	35.12023	16.75203	−0.5524	1.1640	1.2885
35.6727	15.5880	37.29523	6.92198	−0.4941	1.1646	1.2651
37.7893	5.7573	52.93028	7.91033	−0.1177	2.1953	2.1985
53.0480	5.7150	56.79045	28.9444	−0.1099	2.1711	2.1738
56.9003	26.7733	45.26556	19.40028	0.1856	1.6109	1.6216
65.7150	36.3923	45.37339	37.26915	−1.1036	0.6652	1.2886
46.4770	36.6040	50.97943	48.21349	−1.5182	0.6782	1.6628
52.4977	47.5353	43.18651	55.83493	−1.6818	0.2046	1.6942
44.8683	55.6303	36.44823	47.66762	−1.1718	0.2170	1.1917
37.6200	47.4507	24.76005	56.57489	−1.4206	−0.6641	1.5682
26.1807	57.2390	12.40446	42.27877	−0.7282	−0.8539	1.1222
13.1327	43.1327	4.94329	52.21057	−0.8140	−1.4724	1.6825
5.7573	53.6830	13.50295	67.59261	−0.8150	−1.2551	1.4965
14.3180	68.8477	4.44686	76.4212	−0.1675	−1.5798	1.5887

4.6143	78.0010	25.73561	74.92145	−1.2071	−0.7512	1.4217
26.9427	75.6727	34.30223	65.06646	−1.6244	−0.3522	1.6622
35.9267	65.4187	44.3974	74.85234	−1.2329	0.6190	1.3796
45.6303	74.2333	55.63794	65.49268	−1.0931	1.3863	1.7654
56.7310	64.1063	62.19341	54.70594	−1.0663	1.6156	1.9357
63.2597	53.0903	70.97981	45.12546	−2.9149	0.7651	3.0136
75.9690	25.6727	79.4143	17.26995	−3.8877	1.0470	4.0262
83.3020	16.2230	72.073	6.33607	−3.3033	1.0444	3.4645
75.3763	5.2917	91.77979	7.13925	−3.3425	1.9746	3.8822
95.1223	5.1647	105.87145	20.66772	−0.9019	3.5981	3.7094
106.7733	17.0697	112.10904	27.83642	−1.6586	3.3914	3.7753
113.7677	24.4450	114.77724	8.6575	−1.1918	3.5352	3.7306
115.9690	5.1223	126.46504	16.79277	−0.6470	4.0411	4.0926
127.1120	12.7517	132.51657	9.6629	−1.1241	4.2866	4.4315
133.6407	5.3763	146.85387	22.48043	−0.9778	4.7758	4.8748
147.8317	17.7047	142.1418	10.31331	−0.9485	4.5136	4.6122
143.0903	5.7997	131.30033	29.06166	−1.7053	3.9393	4.2926
133.0057	25.1223	143.13022	42.21759	−1.6534	4.2166	4.5292
144.7837	38.0010	122.58907	37.20234	−2.3216	3.1383	3.9037
124.9107	34.0640	104.79709	40.8273	−1.5952	2.9533	3.3566
106.3923	37.8740	95.35882	29.43857	−0.6525	2.8346	2.9087
96.0113	26.6040	93.97444	46.90186	−2.1639	2.3722	3.2109
96.1383	44.5297	85.30489	59.53382	−2.2304	2.0408	3.0232
87.5353	57.4930	69.98028	63.87571	−2.3904	0.9547	2.5740
72.3707	62.9210	66.2389	79.23934	−1.6774	1.2383	2.0850
67.9163	78.0010	52.45462	85.811	−1.5247	0.3923	1.5744
53.9793	85.4187	45.70462	97.19396	−1.3227	0.0820	1.3253
47.0273	97.1120	36.05288	83.0189	−1.1438	−0.1561	1.1544
37.1967	83.1750	22.1926	95.03157	−0.7284	−1.1068	1.3249
22.9210	96.1383	14.05585	84.44486	−0.7278	−1.5241	1.6890
14.7837	85.9690	2.76759	96.0201	0.2276	−2.2349	2.2465
2.5400	98.2550	14.82616	105.74311	0.0002	−1.3266	1.3266
14.8260	107.0697	3.39993	115.53478	0.5636	−2.2546	2.3239
2.8363	117.7893	17.91483	126.06276	0.7182	−0.7952	1.0715

17.1967	126.8580	5.71564	133.37429	1.3553	−1.5364	2.0487
4.3603	134.9107	8.75444	143.83165	1.8118	−0.9520	2.0467
6.9427	144.7837	17.75266	144.27574	1.2333	−0.4656	1.3183
16.5193	144.7413	80.15328	75.12521	−2.7254	1.1882	2.9731
82.8787	73.9370	73.60769	87.22663	−3.1233	0.1993	3.1297
76.7310	87.0273	82.24562	95.80917	−2.6650	1.0678	2.8710
84.9107	94.7413	93.88498	88.60012	−2.1264	2.0385	2.9456
96.0113	86.5617	90.95732	108.6842	−2.1753	1.8262	2.8403
93.1327	106.8580	102.35293	96.19786	−2.6424	2.0492	3.3439
104.9953	94.1487	101.37991	116.80546	−1.7528	2.6568	3.1829
103.1327	114.1487	114.25707	109.22787	−1.7119	3.1319	3.5692
115.9690	106.0960	111.28383	130.71998	−1.8065	2.9306	3.4427
113.0903	127.7893	121.42419	118.57075	−1.8355	3.3638	3.8319
123.2597	115.2070	125.65129	138.10624	−0.9527	4.2962	4.4006
126.6040	133.8100	135.96317	130.29821	−2.2918	3.5672	4.2400
138.2550	126.7310	131.52607	147.66458	−1.3949	4.0239	4.2588
132.9210	143.6407	144.03746	139.87013	−1.4235	4.5361	4.7543
145.4610	135.3340	142.00479	150.92567	−1.4665	4.4910	4.7244
134.5720	134.5297	140.70075	120.41561	−2.0509	3.8963	4.4031
142.7517	116.5193	131.31548	108.8458	−2.2829	3.3001	4.0128
133.5983	105.5457	143.47551	98.33232	−2.7898	3.5063	4.4808
146.2653	94.8260	134.07238	90.29297	−2.7856	3.0963	4.1649
136.8580	87.1967	143.24381	77.61206	−2.5982	3.6751	4.5007
145.8420	73.9370	129.33111	68.59017	−3.0819	2.8752	4.2148
132.4130	65.7150	141.14632	58.26387	−1.7323	3.9882	4.3482
142.8787	54.2757	133.12359	49.6457	−1.7024	3.5497	3.9368
134.8260	46.0960	122.02051	59.51971	−2.4668	2.9157	3.8192
124.4873	56.6040	112.23799	48.78858	−2.7997	2.2269	3.5773
115.0377	46.5617	103.04774	59.65212	−2.2439	2.3285	3.2337
105.2917	57.3237	111.17989	67.81569	−2.6301	2.4394	3.5872
113.8100	65.3763	89.96281	65.06552	−3.1275	1.2132	3.3546
93.0903	63.8523	98.79325	74.15259	−2.4767	2.0359	3.2061
101.2700	72.1167	115.08746	89.96883	−2.6172	2.3912	3.5451
117.7047	87.5777	124.32668	78.60694	−3.0817	2.6803	4.0842

127.4083	75.9267	121.75133	98.42429	−1.7623	3.2173	3.6683
123.5137	95.2070	121.81639	128.1461	−1.8243	3.1508	3.6408
116.6040	143.6830	100.4844	136.9765	−1.8863	2.1505	2.8605
102.3707	134.8260	94.39754	127.27927	−1.6561	2.0723	2.6527
96.0537	125.2070	82.87516	115.49725	−1.8238	1.3486	2.2683
84.6990	114.1487	72.91021	104.86416	−3.0165	−0.2158	3.0242
75.9267	105.0800	60.34002	96.4567	−2.0306	−0.1473	2.0360
62.3707	96.6040	53.02257	102.89254	−1.6341	−0.1555	1.6415
54.6567	103.0480	34.8235	102.32821	−0.7645	−0.7621	1.0795
35.5880	103.0903	25.42911	113.55151	−0.5399	−1.4862	1.5812
25.9690	115.0377	45.73024	117.60213	−0.7044	−0.2295	0.7409
46.4347	117.8317	62.97958	113.05013	−1.1268	0.6371	1.2944
64.1063	112.4130	73.09985	127.51802	−1.7685	0.5330	1.8471
74.8683	126.9850	63.91757	124.14643	−0.9508	0.7174	1.1911
64.8683	123.4290	53.79679	125.61644	−0.7329	0.1978	0.7591
54.5297	125.4187	35.95173	122.1571	−0.3559	−0.5946	0.6930
36.3077	122.7517	28.67633	134.03981	0.5060	−0.4052	0.6482
28.1703	134.4450	34.76506	146.71801	0.6587	−0.1400	0.6734
34.1063	146.8580	44.33128	136.18433	0.0133	0.2577	0.2580
44.3180	135.9267	55.84351	145.04381	−0.5065	0.4295	0.6641
56.3500	144.6143	64.6678	132.74247	−0.7509	0.9221	1.1892
65.4187	131.8203	77.87256	145.00065	−0.2978	2.1643	2.1847
78.1703	142.8363	86.51061	138.5751	−1.1517	2.0134	2.3196
87.6623	136.5617	95.79705	146.13567	−0.8070	2.7490	2.8650
96.6040	143.3867	122.78655	150.95585	−1.2774	3.8862	4.0908
124.0640	147.0697	133.8292	120.40486	−1.9705	3.5892	4.0945
135.7997	116.8157	136.29583	21.29044	−1.3665	3.5858	3.8373
137.6623	17.7047	114.97243	18.18391	−1.8009	2.8922	3.4071
116.7733	15.2917	86.8461	46.9205	1.7661	3.5338	3.9506
85.0800	43.3867	23.16865	43.67346	−1.6997	−1.3642	2.1794
24.8683	45.0377	41.9136	24.49209	−1.1344	0.2164	1.1549
43.0480	24.2757	43.12954	45.12257	−1.4001	−0.2114	1.4160
44.5297	45.3340	52.95448	32.89852	−2.2525	−0.2341	2.2647
76.1383	55.6727	83.57431	67.8882	−3.2414	0.8185	3.3431

86.8157	67.0697	83.21615	82.53695	−2.4989	1.0976	2.7293
85.7150	81.4393	92.35187	97.42127	−2.6011	1.2406	2.8818
94.9530	96.1807	104.00589	109.5597	−2.3864	2.0244	3.1294