International Master’s Thesis

Robot based 3D scanning and recognition of workpieces

ROBIN REICHER
Technology
Robot based 3D scanning and recognition of workpieces
Robot based 3D scanning and recognition of workpieces

Supervisor: Prof. Ivan Kalaykov
Examiners: Dr.
Dr.
MSc.
© Robin Reicher, 2011

Title: Robot based 3D scanning and recognition of workpieces

ISSN
abstract

Quality inspection of a product is central of many manufacturing processes. While inspection on flat surfaces can be made fairly autonomous today, highly reflective free-form objects is problematic in many ways. This thesis is one part out of a two-part project investigating in an autonomous way to recognize, model, store relevant information and inspect these kind of work pieces. This part will focus on the recognition, modeling and database design. The system, established in this thesis will use a robotic manipulator, an industrial camera and the handheld 3-D scanner Exascanner. We present a methodology for preparing a work piece to be inspected autonomously and a simple implementation of the proposed methodology. The implementation recognizes workpieces with a support vector machine trained on histogram of oriented gradients features. These features are extracted from several pictures taken from different angles around the workpiece. The use of different angles are to make the classifier more versatile and robust to object being rotated or moved. If the workpiece is not recognized a spiral shaped dome path is created, scaled with the help of the pictures already taken. This shape helps ensuring a high quality scan of objects were there is no shape information to be used. The robotic manipulator is used to move the scanner along the path around the object, creating a surface profile of the object. This profile is built up of triangular facets of various size and needs to be processed before inspection of the surface can be made. A recursive splitting algorithm is used to make the facets as equilateral as possible and to make their size more suitable for the viewing range of the surface inspection camera. As a final step this information is stored in a database to be used later as support during inspection.
Acknowledgements

Firstly, I would like to thank my supervisor Prof. Ivan Kalaykov for his involvement throughout my masters thesis as his ideas and advice have been very useful. Dr. Boyko Iliev has helped me understand the robotic manipulator while Dr. Dimitar Dimitrov has given me the theoretical background needed to control the manipulator and I am very thankful for their advice and understanding, thank you! Lastly, a quick thank you to my classmates who have helped on many occasions from math to encouragement, I very much appreciated your assistance.
# Contents

<table>
<thead>
<tr>
<th>Appendix</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Introduction</strong></td>
<td>17</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>17</td>
</tr>
<tr>
<td>1.2 Goals</td>
<td>20</td>
</tr>
<tr>
<td>1.3 Thesis Outline</td>
<td>20</td>
</tr>
<tr>
<td><strong>2 Methodology Related to the Project</strong></td>
<td>21</td>
</tr>
<tr>
<td>2.1 Scanning</td>
<td>21</td>
</tr>
<tr>
<td>2.1.1 Coordinate Measuring Machine</td>
<td>21</td>
</tr>
<tr>
<td>2.1.2 Manual 3-D scanning</td>
<td>22</td>
</tr>
<tr>
<td>2.1.3 Robot-Based 3-D scanning</td>
<td>23</td>
</tr>
<tr>
<td>2.2 Object recognition based on 2-D images</td>
<td>23</td>
</tr>
<tr>
<td>2.3 Summary</td>
<td>26</td>
</tr>
<tr>
<td><strong>3 System and Implementation</strong></td>
<td>27</td>
</tr>
<tr>
<td>3.1 System Architecture</td>
<td>27</td>
</tr>
<tr>
<td>3.2 Hardware</td>
<td>29</td>
</tr>
<tr>
<td>3.2.1 IRB 140 robotic manipulator</td>
<td>30</td>
</tr>
<tr>
<td>3.3 Camera and Classification</td>
<td>31</td>
</tr>
<tr>
<td>3.4 Measuring the Workpiece</td>
<td>33</td>
</tr>
<tr>
<td>3.5 Scanning</td>
<td>35</td>
</tr>
<tr>
<td>3.6 Surface profile processing</td>
<td>37</td>
</tr>
<tr>
<td>3.7 Database structure</td>
<td>37</td>
</tr>
<tr>
<td><strong>4 Experiments and Results</strong></td>
<td>41</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>41</td>
</tr>
<tr>
<td>4.2 IRB 140 Workspace</td>
<td>41</td>
</tr>
<tr>
<td>4.3 Achilles heel</td>
<td>42</td>
</tr>
<tr>
<td>4.4 3-D recognition</td>
<td>43</td>
</tr>
<tr>
<td>4.5 Main experiment</td>
<td>44</td>
</tr>
<tr>
<td>4.5.1 Scanning and surface preparation</td>
<td>47</td>
</tr>
<tr>
<td>5 Conclusion</td>
<td></td>
</tr>
<tr>
<td>5.1 Summary</td>
<td>51</td>
</tr>
<tr>
<td>5.2 Future Improvements</td>
<td>52</td>
</tr>
<tr>
<td>A C/C++</td>
<td>55</td>
</tr>
<tr>
<td>B RAPID</td>
<td>61</td>
</tr>
<tr>
<td>C Matlab</td>
<td>67</td>
</tr>
<tr>
<td>D Manual</td>
<td>81</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>The hand-held 3D scanner Exascanner.</td>
<td>18</td>
</tr>
<tr>
<td>1.2</td>
<td>Project general structure and how it is divided between the two authors.</td>
<td>19</td>
</tr>
<tr>
<td>2.1</td>
<td>A CMM measuring the dimensions of an object.</td>
<td>22</td>
</tr>
<tr>
<td>2.2</td>
<td>Setup used in the Rahayem and Kjellander’s &quot;Quadric segmentation and fitting of data captured by a laser profile scanner mounted on an industrial robot&quot;.</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>The general structure of two first parts. Recognition and scanning.</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>The IRB 140 in standing position we are working with.</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Taking consecutive pictures while moving camera.</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>The measuring process.</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Movement of the scanner around a box workpiece.</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>Normals.</td>
<td>38</td>
</tr>
<tr>
<td>3.7</td>
<td>Different styles of splitting polygons.</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>A picture being taken for recognition purposes.</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Objects we tried to classify with two different classifiers.</td>
<td>44</td>
</tr>
<tr>
<td>4.3</td>
<td>Experiment setup.</td>
<td>45</td>
</tr>
<tr>
<td>4.4</td>
<td>Measuring step failure.</td>
<td>45</td>
</tr>
<tr>
<td>4.5</td>
<td>Measuring step success.</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>The IRB 400 mounted in the roof.</td>
<td>53</td>
</tr>
</tbody>
</table>
## List of Tables

3.1 Example of the database used. N is the number of pictures taken for classification.  

3.2 The surfaceInfo matrix, used for inspection. x-y,1-3 are the three vertices making up the face of the triangle, n1 are the centerpoint and n2 the point 1 mm away from it, along the normal.  

4.1 Result of classifying 7 objects using a Artificial Neural Network, optimal result would have been a diagonal line (gray) of ones only.  

4.2 Result of classifying 7 objects using a support vector machine, optimal result would have been a diagonal line (gray) of ones only.
Chapter 1
Introduction

1.1 Background

Nearly every industry has quality inspection procedures of their products which may be different depending on the product. Some companies use conventional manpower on an assembly line or in a dedicated quality control booth while others strive to use state-of-the-art inspection techniques. Much effort has been made to develop reliable, cost efficient, precise solutions. The main reason companies do not get a renewed contract is due to a lack of product quality, especially work awarded by government agencies\(^1\) thereby making quality control a vital step of the production process. In this project we will look at different ways to solve the quality inspection problem. A methodology for solving this type of problem will be proposed together with a demonstration of the implemented system. We have chosen to use a micro-graph system to detect workpiece surface defects. This allows us to not only find irregularities of the workpiece shape but also defects on the surface itself. By keeping the correct distance and angle to the surface, the reflections on the workpiece surface are kept to a minimum, significantly simplifying the inspection of chromium and other highly reflective surfaces.

The entire surface of the workpiece is to be inspected autonomously and will therefore be held by a robotic manipulator which sweeps the piece in front of a stationary mounted camera. Micro-graph inspection has a very limited depth of field, therefore the distance between camera and workpiece needs to be held with millimeter precision. If the distance is not kept constant for the duration of the inspection, images will become blurry and the surface could be mistaken to be smoother than it really is. In order to move the workpiece in front of the camera in an intelligent, efficient and precise way, some prior knowledge about the workpiece surface profile is needed.

We want to be able to inspect an unknown free-form workpiece without any

prior knowledge about the workpiece in question. To inspect the workpiece properly, a full mesh of the surface profile needs to be obtained. This is done in a step prior to the inspection where the workpiece is put on a special reference board\(^2\) and scanned by the commercial scanner *Exascan* (Fig. 1.1). This scanner is compact enough to be mounted on the robotic manipulator and will produce a full 3-D mesh. In order to be a suitable aid during inspection, the mesh surface and normals need additional processing.

The surface profile of the workpiece will be used to create an appropriate path during the micro-graph inspection. It will also enable us to control other parts of the inspection such as:

- Keeping the image focused.
- Ensure that the entire workpiece is scanned.
- Prevent collisions between the workpiece and the camera.
- Hold the camera perpendicular orientated to the surface.
- Plan a trajectory for inspection of the object as fast as possible without sacrificing quality.

To scan workpieces that have already been scanned is not desirable, therefore a third step next to *scanning* and *inspection* is necessary. The *recognition* step will be performed before the two other steps and begins by capturing a series of pictures of the workpiece. With the help of these pictures, a classifier will determine if a piece has been scanned before or not. To make this feasible, a simple database structure holds the data needed for training a classifier. This database will also hold all surface profiles, normals, images and other information about

\(^2\)http://www.youtube.com/watch?v=xbgvctYXx3g&ct=3m20s
each unique workpiece. The whole process should require minimal operator involvement to make it as applicable as possible in an industrial environment. Even though this is a known environment, the architecture should be able to handle the most frequent problems that might occur such as different lightning conditions or minor misplacements of the workpiece.

Figure 1.2: Project general structure and how it is divided between the two authors.

This thesis will deal with the recognition and scanning. The proposed methodology for these two will be explained together with an implementation of a working system. Inspection will be investigated by Syed ZilleHussnain in his master thesis "Robot-based Micrography of free-form Workpieces for industrial inspection"[1] that will be completed around the end of 2011.
1.2 Goals

The goal of this thesis is to implement a system capable of recognize and scan free-form workpieces without any prior knowledge about them. It is however not to have a product ready for industrial use. The project consists of various fields such as machine learning, image processing and mechatronics. The goal is both to find a good concept for a methodology but also to implement and test the system.

- To find a reasonable methodology for scanning and recognizing free from workpieces.
- Generate an efficient motion for the scanner to follow in order to scan each workpiece properly.
- Suggest a database structure for storing relevant information for all possible workpieces in a given class.
- Preparing the surface profile mesh of the workpiece to be used during inspection.

1.3 Thesis Outline

This is organized as follow:

Chapter 2 will describe the previous work in the subject done by researchers.

Chapter 3 is going through our idea both conceptually and how it was implemented in C/C++, MATLAB and RAPID.

Chapter 4 presents the results we found when experimenting.

Chapter 5 contains our conclusions and some ideas for future improvements.
Chapter 2
Methodology Related to the Project

The goal of this section is to provide an overview of what current techniques are available concerning quality control, 3-D scanning and object recognition. Because inspection is such a vital step for most manufacturing industries, extensive research has been made to solve this in a reliable and cost efficient way.

2.1 Scanning

To virtually represent an object is in many fields very practical or even crucial. Therefore many different ways of measuring an object dimensions and shape have been proposed.

2.1.1 Coordinate Measuring Machine

A popular device is a coordinate measuring machine (CMM). The technique is widely used in industrial environments and can be as precise as $1 \times 10^{-6}$ meters. A CMM is measuring objects by using a probe next to the surface, the probe can be mechanical, measuring the object by touching it or indirect by using laser, optics or white light \(^1\).

There are many different types of CMM's, some completely automated and others moved by an operator. The autonomous version of this machine is often limited to only scan a portion of the whole surface. The CMM shown in figure 2.1 is an operator-free version, only capable of scanning an object from above. In the manufacturing industry a CMM is mainly used to verify that a random samples dimensions is within given error-threshold. To scan the entire surface with this type of scanner is not practical, instead a few predefined points are

\(^1\)http://en.wikipedia.org/wiki/Coordinate-measuring_machine
2.1.2 Manual 3-D scanning

Handheld or in other ways manual scanning have until recently been a timeconsuming and complex process. Lately a new generation of commercial handheld scanners have been available such as Polhemus "FastSCAN", zcorp's "ZScanner" or the "ExaScanner"(Fig. 1.1) from Creaform3D. Most stationary scanners keep track of their exact posture to provide a correct pointcloud, these three scanners use reference points to orientate themselves around the object making it possible to get exact scans without the need of a steady hand.

The Exascanner is created by Creaform 3d and is part of their HandyScan-series. It is designed to be used by an operator sweeping the scanner randomly over the workpiece. The scanner is able to track its positioned relative to the workpiece by using reference points. These points are randomly placed on the workpiece or scattered over a "reference board". The approach using a board works better for our application since the solution should be as autonomous as possible. Another important feature with the Exascan is the ability to either output a point-cloud or a full 3-D mesh structure.

The Exascan 3-D scanner has been used in some projects before. Du Zhiqiang et al. scanned a complex Buddha sculpture in the article "Detail-preservation
2.2. OBJECT RECOGNITION BASED ON 2-D IMAGES

3-D Modeling for elaborate Buddha sculpture"[2]. Their project focused on experiments with different precision modes of the scanner, the result is a high detail 3-D model of the sculpture with over 2 million facets but with the least amount of polygons.

There were however not any research papers to be found about the Exascanner used together with a robotic manipulator.

2.1.3 Robot-Based 3-D scanning

Rahayem and Kjellander used the same type of robotic manipulator we use (see more in sec. 3.2.1) for their paper "Quadric segmentation and fitting of data captured by a laser profile scanner mounted on an industrial robot"[3]. They emphasized the industrial need of a reliable and fast measuring system based on 3-D scanners. Their work focused on the segmentation of the 3-D point cloud, something we will leave untouched in this thesis. Their work is still relevant for this project since the physical setup they used is similar. A camera and a laser source are mounted at the tool center point (TCP) (Fig. 2.2) of the robotic manipulator.

Their setup used a turntable, a kind of motor-driven potter-wheel on which the object is placed upon. This is popular in automated 3-D scanning and significantly reduces the need of the scanner being mobile. If the setup is properly calibrated the scanner only needs to be moved vertical or in an arc alongside the object. The problem with this method in our setup is the reference board we use, the object needs to be remained stationary for the full duration of the scan. A solution to this could be to put the reference-board on the turntable.

Gu and Li from University of Calgary in Canada 2003 published an article that aimed to compare and review free-form surface inspection[4]. This mainly took up different issues around CMM’s but also covered some non-contact solutions that could be used for a laser profile scanner as well:

"In many of existing non-contact inspections including vision and laser scanning, substantial human involvement using laborious and time-consuming techniques is still required."

2.2 Object recognition based on 2-D images

Image recognition in the manufacturing industry is used sparsely. In 1999 the book Industrial Image Processing[5] stated that:

"..., despite a multitude of successful applications, digital image processing have not yet established itself as an accepted element of manufacturing technology". Twelve years later this still holds true, although to a lesser extent. Recognizing 3-D objects with only a set of 2-D images has been a popular topic of research for decades but there is still no reliable general method. One of the main problems is the rotation of the object, a sheet of paper looks like a line from one
angle and like a box from another.

Feed-forward Artificial Neural Networks (ANN's) have been proposed as a good solution to be used in the field of image processing and recognition, *Image processing with neural networks—a review* [6] by Egmont-Peterson, de Ridder and Handels explains this their paper from 2002. They promote ANN's being used in image processing and emphasize the need to work with image descriptors rather than on a pixel-by-pixel basis. The same also holds true for Support-vector-machines (SVM), already in 1999, Chapelle, Haffner and Vapnik demonstrated that it is possible to use SVMs to classify images represented as histograms with high accuracy and performance [7].

Image descriptors are a way to simplify an image. For example, a 800 x 600 image with 256 colors has $4 \times 10^5$ pixels with 256 possible states resulting in a total of $10,24 \times 10^7$ configurations. Image descriptors break down and remove less informative information from the image. Many different image descriptors exist today:

**SIFT** Scale-invariant feature transform \(^2\) is invariant to scale, orientation, affine distortion and partially to illumination changes.

\(^2\)http://en.wikipedia.org/wiki/Scale-invariant_feature_transform
2.2. OBJECT RECOGNITION BASED ON 2-D IMAGES

**HOG** Histogram of oriented gradients \(^3\) is newer than SIFT and is invariant to geometric and photometric transformations. It is particularly useful for detecting humans in a scene.

**SURF** Speeded Up Robust Features \(^4\) is inspired by SIFT but is made to be faster and claims to be more robust to image transformations.

In this project we tried these different techniques to evaluate which was most suitable for our project. More info about this can be found in section 3.3.

\(^3\)http://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
\(^4\)http://en.wikipedia.org/wiki/SURF
2.3 Summary

There are no papers to be found regarding the entire process we describe in this project. An abundance of papers exist that addresses either Recognition, Scanning or Inspection. A sound methodology and implementation of a system that addresses all three problems is non-existent today.

CMMs are better suited for inspection rather than scanning since they have problems with unknown objects as well as scanning entire surfaces. A 3-D scanner suites our purpose better and has been tested in previous experiments with positive results[3].

The Exascanner is a good choice for this kind of project because of the focus on the 3-D model rather than the scanning itself. Surprisingly, there are hardly any papers using this scanner, this thesis will hopefully contribute to filling that gap.

A turntable will not be used as, the physical workspace in front of the robot is too small to fit a turntable with a reference board. Also, the robot we will be using is flexible enough to scan the entire object from all sides, except the bottom.

For recognition a series of images will be taken to train a feed-forward artificial neural networks or a Support vector machine, tests will be preformed to decide which will work best for our system. The images raw data will not be used for classification directly, instead the image descriptors will be extracted and used for both training/classification. A few different descriptors will be tested before the final decision is made.
Chapter 3
System and Implementation

This cross-disciplinary project covers various fields such as mechatronics, machine learning and image processing. This section we will first describe the general idea about the system implementation followed by a deeper description of the techniques used.

3.1 System Architecture

To ensure a flexible, easy accessible architecture it is essential to keep the methodology and implementation modular. The general structure can be seen in figure. 3.1 where the inspection-step starts at the "end" block. Each of the steps will be described briefly in this section to give the reader a general idea of how the project is structured, later in this chapter each step is more thoroughly explained.

The first step in our project is taking a few pictures consecutively. While taking them the robotic manipulator moves the camera in 90° arc around the workpiece, the motion starts from a top-down look of the workpiece moving along the side until the manipulator base is reached. The focus of the camera remains steady on the workpiece. These pictures are temporarily saved to be used for calculating the dimensions. All picture histogram of oriented gradients[8] is calculated in order to classify them easier. More about this in section 3.3.

If the classification finds a matching workpiece in the database, the piece is known and will be sent for inspection. If the classifier does not find any match the workpiece is unknown and the program proceeds toward scanning the workpiece. The user is prompted to give a name and tag for the workpiece, the tag is used on all files associated with it. The workpieces mesh-file will for example be named "tagMESH.stl", where tag is specified by the user. The first and last pictures which were taken at initial and final pose of the ma-
Figure 3.1: The general structure of two first parts. Recognition and scanning.
3.2 HARDWARE

A manipulator with the camera from the photo-set are used to calculate the dimensions of the workpiece. This estimation is rough but sufficient for the next task which is to create the manipulator motion path for the scanning. This path has the shape of a spiral put on top of a dome and is described in detail in section 3.5.

When the manipulator is moving on the path the Exascanners 3D capturing software acquires a 3-D surface profile of the workpiece. The scanner produces a full 3-D CAD-mesh in an ascii encoded stl-format which is a common format in computer-aided-manufacturing. This mesh is however not suitable to directly be used for inspection and needs to be processed further. The entire mesh is investigated and modified in order to fit the inspection camera viewing range (sec. 3.6). Each facet of this mesh have a normal vector describing the perpendicular angle to the facet. These normals are modified and used to orient the camera during the inspection. Most 3-D file formats including the stl-format use unit normal vectors where the length of the normal is exactly 1.0 and extends from the origin, to be more useful for our cause the normal is moved to originate from the center of the facet it is concerning instead. The new information about the normal is saved in a database of known workpieces together with some graphical primitives and other information related to the workpiece.

To be able to recognize the workpiece the next time, the classifier is retrained with the new set of graphical primitives.

3.2 Hardware

Three different pieces of hardware will be used. A camera, a robotic manipulator and the previously mentioned 3-D scanner Exascan. The first two of these are available in the Örebro university Center for Applied Autonomous Sensor Systems (AASS) lab.

The camera chosen for recognition is the Marlin F033C fire wire camera by Allied Vision\(^1\). This is a small camera made for an industrial environment and should suite our purposes perfectly. 640 x 480 monochrome pictures will be used in this project together with a 12.5 mm television lens.

The most important component of our system is the 3-D Exascan device, which is to produce the surface profile of the unknown workpiece. Unfortunately the Exascanner did not arrive to the lab. This sadly makes the implementation incomplete but since our concept is modular, no other part has been affected. For test purposes in the project, instead of doing an actual scan, a surface

\(^1\)http://www.covistech.com/uploads/AVT_Datasheets_Product_Images_App_Notes/MARLIN_F_033B_C.pdf
profile was created in a CAD program and sent to the "surface-preparation"-step as if it was provided by the scanner.

### 3.2.1 IRB 140 robotic manipulator

The camera and the scanner will be both placed on ABBs IRB 140 manipulator. It is a small six axes robot and can be used for a variety of tasks. Mostly used together with smaller tools, the usual industrial tasks includes spraying, welding and gluing. When lifting workpieces or tools with a weight less than 6 kilos the position precision is only about 0.03 mm\(^3\). This suits the project purpose well since the camera, the 3D-scanner and the workpieces used, weigh far less than 6 kilos. The Exascanner weighs 1.25 kilos, the camera only 0.12 kilos plus the camera lens on about 0.1 kilos. Combined they weigh 1.47 kg, leaving 4.75 kilos to be used on the mounting unit to properly fasten our tools on the robot tool center point (TCP).

The scanning is performed in front of the robot or left of it as seen in figure 3.2. The scanner is mounted facing down to the floor or perpendicular to the robots TCP orientation. The reason for this is that the Exascanner together with most scanners is significantly longer than high. A too long scanner would make the workspace smaller than needed and the workpieces risk not being

---

\(^{1}\)ABB’s IRB 140 compact data sheet.
properly scanned. The workspace is rather small with a maximum of about 450 mm wide in front of the robot.

3.3 Camera and Classification

We choose to work with Histogram of oriented gradients (HOG) descriptors. It suites our project well since it is geometric invariant and focus on the brighter parts of the image. In 2006 a group of researchers combined it with a Support vector machine with promising results[9]. The implementation of HOG we found also worked well without implementation.

To classify a 3-D workpiece based on 2-D images is complex, several techniques often need to be combined in order to get a reliable solution. Humans classify by using two eyes to get a feeling of depth trough the binocular vision system they are born with.

For computer algorithms there are mainly five different challenges when dealing with this type of classification, all needed to be addressed either by pre-processing the image and/or by training an appropriate classifier.

1. **Background** - Without any depth-information it is hard to identify the difference between the background and the actual workpiece.

2. **Lighting** - A workpiece with a light source close to it looks very different depending on where that source is directed. The genera lighting are also important since every pixel is affected by the difference of the ambient light that changes during the day.

3. **Scaling** - Scale invariant image features need to be extracted and compared in order to detect the difference between a small and big object.

4. **Rotation around an axis between camera and workpiece** - The simpler of two rotation problems but far from trivial. If the camera or workpiece is rotated around the axis between the two, the image will have the exactly same features but in different positions. There are many proposed solutions to this, a popular one is to find SIFT-features (Scale-invariant feature transform [10]) which are invariant to both scale and this type of rotation.

5. **Rotation around an axis other than between camera and workpiece** - The harder version of the last problem is if the workpiece is rotated around some other axis than the one between the camera and workpiece. Even if this is just a few degrees off, it might make the shape of the workpiece completely different. Humans partly solve this by having a rough idea
how the workpiece look in 3-D space and thus being able of rotating it to see if it might fit. Modern vision classification algorithms have not found any good general solution to this but combinations of 3-D models and machine learning is often used.

To classify a workpiece based on vision alone is a task that has been keeping researchers busy for decades. The human brain is very good at this and can overcome most problems computer algorithms have. One of the biggest advantage human classification has is the ability to look at the bigger picture and prior knowledge about how things in general are. When we see a round workpiece laying on a table we presume that it is a ball and not the round tip of a long pole. When we do not recognize something we might pick up the workpiece and turn it a few times to find a feature we recognize, if the workpiece is too big we simply walk around.

This is exactly how we solve problem number five, rotation around the axis other than between camera and workpiece.

A few pictures are taken consecutively in an almost 90° arc motion to get many pictures of the same workpiece from different angles. This way we can train the classifier to be more robust and recognize a workpiece even a few degrees rotated.

In the special case of this thesis all problems except scaling need to be addressed. This is because the distance between the workpiece and the camera will always be the same.
After much consideration and some basic testing I decided that the classification part should be performed by using the SVM technique. A SVM will be trained for each object, this SVM will determine if the picture provided characterise the workpiece. Since we will have more than one picture for each classification, the number of classification will be the number of pictures times the number of objects plus one for the empty scene. To be able to determine if a classification was successful a certainty-variable is going to be used. This is calculated by summing up the best objects classification results and dividing it by the number of pictures taken. The number one will mean the classifier recognized each image it was given, we will furthermore call this a 100 % certainty.

As previously mentioned in section 2.2 it should be avoided to work on a pixel by pixel basis with neural networks[6].

3.4 Measuring the Workpiece

To keep a proper range between the scanner and workpiece is crucial to ensure the quality of the scan, many scanners have a short range and therefore the scan trajectory needs to be as tight as possible without crossing the workpiece. We do this by defining how big the spiral-dome-path should be. For this we need to get a radius and a height.

To measure a workpiece based on 2-D images is not a trivial problem. In this project several different image process techniques are used to get the system robust enough to work in different lighting conditions.

This measuring is made by using the Marlin camera mounted on the robotic manipulator. The measuring is done in a series of steps, first a 10x10 median filter is used together with an equally big Gaussian filter with sigma 2.0 to remove noise and most shadows (fig. 3.4). To get a binary image we then apply a canny edge detector with a threshold of 2.0. Last part is to fit all points in a rectangular box which is done by a MATLAB function called \texttt{minboundrect}\(^2\). \footnote{http://www.mathworks.com/matlabcentral/fileexchange/13624-minimal-bounding-rectangle}

\texttt{minboundrect} creates a box containing the workpiece represented by 4 2-D coordinates. These coordinates are used to calculate the dimensions of the piece in millimeters. This is definitely not a precise number. When measuring from above the pixels per cm ratio we use to translate from pixels into millimeters is calculated from the table surface, therefore all workpieces higher then a laying piece of paper will be slightly misscalculated.

The same is true when measuring from the side in order to get the height, the workpiece is treated like a flat pieces of paper standing up with the broad side against the camera. If we would underestimate the workpiece size this could have consequences, mainly hitting the object with the scanner. But since we always underestimate the width, depth and height of the workpiece, the rough
Figure 3.4: The measuring process.
estimation is acceptable. To measure the height is done more or less in the same way, only difference is that the camera is closer to the table when the photo used for measuring.

### 3.5 Scanning

The Exascan is created to be as user-friendly as possible and produces a whole 3-D mesh in .stl format. To get around the problem of the absent scanner we have replaced this step by simply providing a .stl file created in a CAD-program. It will then be possible to do experiments of the whole system even though it is somewhat off a *best-case-scenario* where the scanner works flawless.

The motion of the scanner during the scan is important to achieve a high quality scan. Several different ideas were discussed. One of the first was simply moving the scanner in a number of 180° degree arcs over the object. This would scan the object from above much more then from the sides, creating a scan of uneven quality, the motions would also be unnecessary jerky. The motion path chosen for the scanner has the shape of a dome with a spiral wrapped around it (Fig. 3.5) where the scan starts at the top of the dome. This shapes have two advantages.

- **Regularity** - The entire trajectory is kept without any interruptions or sharp motions, like the hand-held scanner is designed for.

- **Homogeneous** - No part of a generic object is scanned more than any other part.

\[
r = a + b\theta \\
(3.1)
\]

\[
z = b\sqrt{1 - \left(\frac{x}{a}\right)^2 - \left(\frac{y}{a}\right)^2} \\
(3.2)
\]

The trajectory and dome are flexible and can easily be adjusted to fit the scanner characteristics. Equation 3.2 and are combined to create the shape, the complete implementation can be seen in (sec. C).

The dome is twice the height and diameter of the measured workpiece to ensure that the scanner would not hit the piece. There is also a minimum height of 200 mm, the dome lower boarder ends at this level regardless of how high the workpiece is. If the workpiece is under 200 mm high the trajectory will become a flat spiral instead to ensure safety. There are many ways to get the scan to be tighter.

- Lower minimum height.

3[^3](http://www.creaform3d.com/en/handyscan3d/handyscan3d-video.aspx)
Figure 3.5: Movement of the scanner around a box workpiece.
• Lower safety margin between the dome and the workpiece.
• Execute the trajectory slower.
• Increase b in equation 3.1 to get a denser spiral-dome.

3.6 Surface profile processing

The surface profile created by the scanning process needs to be modified to better fit the task of inspection. The inspection camera is checking every facet in the 3-D surface profile. Some of these facets are small while other parts might be bigger (more flat). The camera should be able to see the entire surface of a facet.

To accomplish this, two techniques have been implemented. The most obvious one is to keep a perpendicular angle between the surface and the camera optical axis, this greatly increases the viewing area. The normals from the .stl mesh file are translated to originate from each facets center and the camera optical axis is put colinear with them. (Fig. 3.6)

The second technique is to make the facet size smaller or equal to the viewing range. To achieve this polygons too big facets are split repeatedly until the distance between the center and the corner of the facet is smaller than the camera’s viewing range. Many type of slices were tried out, the first one tested (Fig. 3.7a) divided each facet into 4 smaller ones of the same shape. This looked like a nice idea but turned out to make some triangles unnecessary small and narrow. The idea finally settled was to simply split the triangle in half (Fig. 3.7b). The longest side of the triangle is calculated and splitted, this way we get triangles as equilateral as possible, thus simplifying the inspection.

3.7 Database structure

A database is created to hold information about each unique workpiece. The information in this database is mainly used for recognizing already scanned workpieces but also holds information about the piece name, tag and mesh-file (table 3.1).

The database is created in Matlab and saved as a .mat-file. It should only be seen as a concept for a database, a Matlab database is definitely not optimal regarding accessibility, speed or flexibility. In our project we will however only work with a few workpieces making it a good solution. For a bigger project some other database structure is recommended, such as SQL, XML or a c++ implemented list structure.
(a) Normals shown as 'hair' of a deformed tube

(b) Close up picture.

Figure 3.6: Normals.
3.7. DATABASE STRUCTURE

(a) Spliting triangle polygons into four new ones.

(b) Spliting triangle polygons into two new ones, creating as equilateral triangles as possible.

Figure 3.7: Different styles of splitting polygons.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>Matlab SVM structure</td>
</tr>
<tr>
<td>Name</td>
<td>string</td>
</tr>
<tr>
<td>Tag</td>
<td>string</td>
</tr>
<tr>
<td>HOG</td>
<td>81xN double</td>
</tr>
<tr>
<td>id</td>
<td>integer</td>
</tr>
<tr>
<td>Mesh</td>
<td>string</td>
</tr>
<tr>
<td>Img</td>
<td>480x640xN double</td>
</tr>
<tr>
<td>SurfaceInfo</td>
<td>5x3 double (Tab. 3.2)</td>
</tr>
</tbody>
</table>

Table 3.1: Example of the database used. N is the number of pictures taken for classification.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>y1</td>
<td>z1</td>
</tr>
<tr>
<td>x2</td>
<td>y2</td>
<td>z2</td>
</tr>
<tr>
<td>x3</td>
<td>y3</td>
<td>z3</td>
</tr>
<tr>
<td>n1_x</td>
<td>n1_y</td>
<td>n1_z</td>
</tr>
<tr>
<td>n2_x</td>
<td>n2_y</td>
<td>n2_z</td>
</tr>
</tbody>
</table>

Table 3.2: The surfaceInfo matrix, used for inspection. x-y,1-3 are the three vertices making up the face of the triangle, n1 are the centerpoint and n2 the point 1 mm away from it, along the normal.
Chapter 4
Experiments and Results

4.1 Overview

The lack of scanner made the experiments a bit more fragmented then initially planned. The scanner have been replaced by a cardboard box of roughly the same size and the 3-D mesh that should be created by the scanner is replaced with a 3-D model created in advance by CAD program. The recognition part is working but should definitely have more descriptors for empty scenes or invalid scenes, right now the system has only been tested with a limited number of examples. The reason we used a camera to do some of the tasks was to get a fast system. The height estimator works but is very rough. If we would have a real scanner there would also be a "reference board" under the workpiece that the scanner needs. This board would need to be subtracted from the images before any measuring is made, this could making the measuring process more inaccurate.

4.2 IRB 140 Workspace

The IRB 140 is a small industrial robot which can rotate 360° around its base but has a limited workspace. If mounted with a tool, the inner workspace shrinks. There is a limit of how big tool you can have before this inner workspace becomes non-existent. With inner we refer to the workspace inside the robot's normal reach. Even without any tool the IRB 140 workspace is small, having a maximum dome radius of only a few centimeters.

There are many ways to get around this, one is to attach the robot to the ceiling and put the workpieces underneath it, more about this in section 5.2. A simpler solution is to raise the minimum boundary for how low the dome can be. As it can be seen in figure 3.2 the workspace is widest at height 352 mm. If all domes lower boarder would be to this height, the dome would be able to be
4.3 Achilles heel

One problem with all scanning is that second workpiece needs to be standing on a surface or mounted on a place by some fixture. When scanning, the floor or the fixture will cover a part of the object making it impossible to scan. One solution is to simply scan the object, turn it upside-down and then scan it again. The two scans need to be put together again, something far from trivial. The floor(fixture) also needs to be removed to get a full 3-D surface of the entire object. This is quite possible but the drawback is the involvement of an operator to turn the object. This could be solved by using another robot with a gripper. Another solution is to use a transparent reference board to put the
workpieces on. The object would then be scanned from above and underneath. An algorithm to remove mesh artifacts created by the transparent reference board might be needed together with a very dexterous robotic manipulator.

4.4 3-D recognition

We setup a simple experiment to determine if a Artificial Neural Network (ANN) or a Support Vector Machine (SVM) would be appropriate. The initial concept was that ANN’s would be the best solution for this project since they have traditionally been said that object recognition is one of the major strengths of ANN’s.

A total of six objects was chosen (Fig. 4.2), two of them with a very similar shape and color to really strain the classifiers ability. The objects were put one by one underneath the camera and pictures were taken as described in section 3.3. A total of eight pictures was taken during a 12 sec arc motion, The histograms of every pictures oriented gradients (HOG’s) were calculated and a separate ANN and SVM classifier were tested using the HOG identifiers for individual object. Each time a new object was added all previous classifiers was retrained using the newly added object as a 'Bad Example’. This process was fast initially but then gradually got slower as the training set grew. The last object did not take more then 4-5 seconds but most likely the needed processing time grows exponentially, making this solution impractical for big training-sets.

Besides the six objects a seventh classifier was also trained for identifying an empty scene.

After the training was completed every object was scanned again being put under the camera in roughly the same way. The results of the scan of all objects can be seen in table 4.1 and 4.2.

The ANN network was trained to represent a successful classification with object number 1. No limits to the output range was made. What we wanted to see was a diagonal row with ones, ranging from the upper left to the lower right. Every column is representing a classification and each cell in that column is the ANN’s answer to how similar that object is the one it was trained to recognize. Only a few objects wew classified correctly and the success-rate is marginally better then a random guess.

A network can be configured in many different ways, we settled a configuration quickly when the classification started to yield results. Since this was not the main topic of this thesis we decided to leave this open for others to improve.

The results from the SVM are very different. SVM’s classification is binary, either it is the object or it is not. As we can see in 4.2 the SVM succeeded to classify the objects much more accurate. The only object not being classified correctly was the battery. Besides that, there were only three incorrect classifi-
Figure 4.2: Objects we tried to classify with two different classifiers.

Figure 4.2: Objects we tried to classify with two different classifiers.

<table>
<thead>
<tr>
<th>Artificial Neural Network (ANN)</th>
<th>Electronics</th>
<th>Rubber</th>
<th>Cog</th>
<th>Big Lamp</th>
<th>Battery</th>
<th>Nothing</th>
<th>Small Lamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronics</td>
<td>0.8223</td>
<td>1.1023</td>
<td>-0.4244</td>
<td>-0.6136</td>
<td>1.8327</td>
<td>0.0642</td>
<td>2.5013</td>
</tr>
<tr>
<td>Rubber</td>
<td>-0.5117</td>
<td>0.0309</td>
<td>1.7092</td>
<td>1.4442</td>
<td>2.0701</td>
<td>1.7671</td>
<td>1.0073</td>
</tr>
<tr>
<td>Cog</td>
<td>-3.1818</td>
<td>-1.4047</td>
<td>0.8911</td>
<td>1.3350</td>
<td>0.2297</td>
<td>0.3781</td>
<td>-1.0030</td>
</tr>
<tr>
<td>Big Lamp</td>
<td>1.4068</td>
<td>2.4325</td>
<td>1.8358</td>
<td>0.8895</td>
<td>0.3252</td>
<td>1.8426</td>
<td>-1.8624</td>
</tr>
<tr>
<td>Battery</td>
<td>0.6531</td>
<td>-0.0239</td>
<td>0.5002</td>
<td>2.2752</td>
<td>4.2975</td>
<td>0.2125</td>
<td>2.2438</td>
</tr>
<tr>
<td>Nothing</td>
<td>1.9330</td>
<td>1.7925</td>
<td>-0.9832</td>
<td>0.6252</td>
<td>2.7998</td>
<td>-1.1972</td>
<td>1.4936</td>
</tr>
<tr>
<td>Small Lamp</td>
<td>-0.2220</td>
<td>0.6290</td>
<td>0.3514</td>
<td>-0.8973</td>
<td>0.3197</td>
<td>0.5727</td>
<td>-1.1700</td>
</tr>
</tbody>
</table>

Table 4.1: Result of classifying 7 objects using a Artificial Neural Network, optimal result would have been a diagonal line (gray) of ones only.

cations. Just like the ANN we did not spend very much time to configure the SVM, and did not continue through with investigation.

4.5 Main experiment

To test the entire process we set up some simple experiments will go through what was done step by step and describe the results. This will only cover the classification and scanning part of the project (fig. 1.2). Our starting setup consisted of an empty database, the six (Fig. 4.2) different object which were used in section 4.4 was reused in this part. The output of our absent scanner was substituted by three different premade 3-D meshes, these meshes do not reflect the physical objects shape and the output of the scanning experiments will therefore be written about in a separate part after the classification results.

We started the experiment by not placing anything in the blue scanning area (Fig. 4.3). The camera swept over the area as it should, taking a total of 10 pictures over 10 seconds. After the scan the user in prompt to put in name and a tag on the object, we choose Empty/Empty.
4.5. **MAIN EXPERIMENT**

<table>
<thead>
<tr>
<th></th>
<th>Electronics</th>
<th>Rubber</th>
<th>Cog</th>
<th>Big Lamp</th>
<th>Battery</th>
<th>Nothing</th>
<th>Small Lamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electronics</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rubber</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cog</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Big Lamp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Battery</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Nothing</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Small Lamp</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.2: Result of classifying 7 objects using a support vector machine, optimal result would have been a diagonal line (gray) of ones only.

Figure 4.3: Experiment setup.

Figure 4.4: Measuring step failure.
The first object to scan was the battery (Fig. 4.3). It was classified as the empty scene with a 100 % certainty since it was the only thing the support vector machine (SVM) classifier was trained for. The classifier retrains all the old networks and adds a new one for every object scanned. This means that the first few scans most probably are bad, but as the SVM gets more "bad examples" from other objects, it will get better. We disregarded that the object was classified as the empty scene and manually changed the "certainty" parameter to 0 %, making the program scan the battery.

When looking at the pictures taken of the battery we realized they were too dark, so we added a industrial light (left in Fig. 4.3) to brighten them up. We took new pictures and retrained the SVM for the empty scene and a battery.

The next object to be scanned was a small lamp, it was also classified as the empty scene and needed manually be added.

Object number four was a small circuit-board. The best classification was 70 % for the empty scene, since we required at least 80 % the object was defined as unknown and was properly trained for and added to the database.

The small Lamp was put in the classification area again in roughly the same position as last time, it was classified as a small Lamp with 80 % certainty.

Our next object was a small black cog. As we can see in figure 4.4 the part of program responsible for measuring objects failed. The combination of a brighter scene then usually together with a very dark workpieces made the program misjudge the size of the cog. This have worked very well up until now on every object(Fig. 4.5). The cog was also recognized as the empty scene, probably because of the dark color of the cog.

A small rubber was the second to last object, it got classified as the empty scene as well. Probably the size of the objects matter. Smaller objects have features harder to spot and therefore they are harder to classify.

The last object was a lamp of the same shape as the previous lamp, but about
4.5. MAIN EXPERIMENT

double in size. The best classification was 30 % for the small cog.

After all the objects had been scanned we tried to classify all the objects again and again. When the objects were put back in same pose as initially trained for we had about 70-80 % success rate. The position of the object had little importance for the results. The rotation had much more importance, which was unexpected since the HOG-descriptors should be rotation-invariant. The reason for this is probably that we take pictures from 10 different angles making the object look very different when rotated.

4.5.1 Scanning and surface preparation

The scanning trajectory was calculated for each of the objects (not the empty scene) in the experiment, except for some minor setbacks with the camera cord being jammed between the camera and the robot. Depending on how big the object was the trajectory of the robot’s TCP was changed accordingly. The trajectory has a limit to how low it is allowed to scan of 200 mm. Because of this, the "scanning-dome" did never change in height since all objects scanned were under 200 mm high. We could have added a bigger object but objects higher then 150 mm is outside the camera’s view-range. So every scanner-path in our experiment looked like a flat spiral on height 200 mm (fig. 4.6b). A flat spiral on height 200 with a width of about 100 mm seems to be just on the boarder of what this robot can do in this setup, the joints near the TCP reached nearly their max angles.

For testing the surface modification part of the code, a lot of meshes were tried. If the meshes were big and complex and the camera view range was set to a low value, a couple of minutes to process was needed. Three of these tests can be seen in figure 4.6a-4.6b.
(a) A flat spiral is created for objects with height < 200 mm.

(b) Spiral created for objects higher than 200 mm.
4.5. MAIN EXPERIMENT

(a) ONE.stl before surface preparation.
(b) ONE.stl after surface preparation with camera maximum view-range of 1 mm.

(a) TWO.stl before surface preparation.
(b) TWO.stl after surface preparation with camera maximum view-range of 0.01 mm.
(a) THREE.stl before surface preparation. (b) THREE.stl after surface preparation with camera maximum view-range of 2 mm.
Chapter 5
Conclusion

The main goal of this thesis was to create and test a feasible methodology for scanning and recognizing workpieces. This have been achieved and the surface profile needed for surface analysis produced is suitable for the application in mind.

5.1 Summary

Main contributions

• Generate a suitable robot motion path to enable scanning by a 3-D scanner to be performed.

• Define an algorithm for splitting polygons into manageable pieces for computing the surface normals needed for surface micro scan.

• Design a database prototype that can hold the necessary data for inspecting free-form objects.

database The structure of the code is made to be robust and flexible, every block in the flow chart diagram (fig. 1.2) in chapter 3.1 is replaceable.

The robot motion created for the scanner works as intended, one of the more important objectives of this thesis. The code should work for all ABB robots using RAPID and automatically scales the domes to fit the workpiece size. The domes probably needs to be fine-tuned further when tested with an actual 3-D scanner to take into account its real dimensions. The code is well commented and it is easy to modify a few parameters in order to adjust the spiral-dome:

• The distance between the rings.

• Number of rings.

• Speed of motion.
52

CHAPTER 5. CONCLUSION

• The minimum distance to the table.
• The resolution of the dome.

More about this can be found in the appendix (Sec. C).

A database prototype is also constructed in Matlab and saved as a .mat-file. Even though simple, it works well and can handle typical errors like duplicate items, etc. Since a scanner never arrived it is difficult to know exactly how it can be interfaced and what problems could occur. It is not clear also how the produced surface profile data are compatible with the stl-file format. The developed system as a whole needs a lot of work to be made functional in real industrial environment. The main problem is the workspace size and the difficulties recognizing 3-D object with 2-D images without a bigger set of pictures.

5.2 Future Improvements

This project is mainly a feasibility study and to get this system to properly work in an industrial environment a lot more testing and development is probably needed. Some of the improvements that could be made:

• **Bigger** robot to get a bigger workspace or to attach the IRB 400 to the ceiling (fig. 5.1). If mounted on properly on the correct height the workspace will be not only bigger but also more homogeneous around the work-piece.

• **Deformable** spiral shaped dome in the xy-plane. If the dome would better fit the object, this would improve the scan quality. This requires a more advanced measuring system together with some changes to the function "CreateSpiralDome" (Sec. C).

• **3-D** object recognition by using the 3-D scanner together with 2-D images. Would literally add another dimension to the recognition, making it much more reliable.

• **Create** a more robust database structure designed for more objects and better error handling.

• **Unite** the different parts of the system better by re-write the code to be only c/c++. This would also make the recognition part faster and easily changed.

---

1 ABB’s IRB 140 compact data sheet.
5.2. FUTURE IMPROVEMENTS

Figure 5.1: The IRB 400 mounted in the roof.
Appendix A
C/C++

The code written in C/C++ is for controlling the IRB 140. The first section
*abb_advanced.cpp* is a short piece of code to tell the robot to execute the com-
mands on the ftp server.

**abb_advanced.cpp**

```cpp
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "sockABB.hh"

using namespace std;

int main()
{
    // run code on server
    char runstuff[256] = "[0,0,0,0,0,0]5";

    // create client that communicates with abb
    // constructor takes one argument, if true the client is
    // in verbose mode and all messages are printed
    sockABB abbClient(true);

    // connect with a server
    abbClient.connectSock();
    abbClient.writeSock(runstuff);

    sleep(1);
}
```
APPENDIX A. C/C++

// close the connection with the server
abbClient.closeSock();

sockABB.cpp

#include "sockABB.hh"

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <string.h>
#include <netdb.h>
#include <time.h>

using std::cout;
using std::endl;
using std::string;
using std::stringstream;

sockABB::sockABB(bool v) {
    bzero(buffer,1000);
    n = 0;
    sockfd = -1;
    verbose = v;
}

sockABB::~sockABB() {
    if (sockfd >=0) {
        close(sockfd);
    }
};

void sockABB::errorSock(const char *msg) {
    perror(msg);
    exit(0);
}

void sockABB::connectSock() {
    int portno = 1300;
    struct sockaddr_in serv_addr;
    struct hostent *server;
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) >= 0) {

    server = gethostbyname("192.168.200.91");
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host
"); exit(0);
    }

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
        (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *)&serv_addr,
        sizeof(serv_addr)) < 0) {
        errorSock("ERROR sockABBconnect : connection ");
    }
    cout << "sockABBconnect : Connection to irb140 established. "
        << endl;
} else if (sockfd < 0) {
    errorSock("ERROR sockABBconnect : opening socket :");
    sockfd = -1;
}

/** \brief Write to ABB socket. */

void sockABB::writeSock(char* pos) {
    if (sockfd >= 0) {
        bzero(buffer,256);
        strcpy(buffer,pos);
        n = write(sockfd,buffer,strlen(buffer));
        /* if (verbose) {
                cout << "sockABBwrite : command : " << pos << endl;
            } */
        if (n < 0) {
            errorSock("ERROR sockABBwrite :");
        } else {
            cout << "ERROR sockABBwrite : can’t write to invalid descriptor ";
        }
    }
/* * \brief Read from ABB socket.

Message cannot be longer than 1000 characters.
*/

void sockABB::readSock (string * msgOut) {
  if (sockfd >= 0) {
    bzero(bufferOut,1000);
    do {
      bzero(buffer,256);
      n = read(sockfd,buffer,255);
      strcat(bufferOut,buffer);
    } while (strstr(bufferOut,"END") == NULL);
    msgOut->assign(bufferOut);
    /* if( verbose ) {
      cout << "sockABBread : message : " << *msgOut << endl;
    } */
  } else {
    cout << "ERROR sockABBread : can't read from invalid descriptor " << endl;
  }
}

void sockABB::closeSock() {
  char closeMsg[256]="[ Close. ] 0 " ;
  int n ;
  string tmp ;
  /* if( verbose ) {
    cout << "sockABBClose : Closing socket... " << endl;
  } */
  n = write(sockfd,closeMsg,strlen(closeMsg));
  if (n < 0) {
    errorSock("ERROR closing the server (writing to socket)");
  }
  //this->readSock(&tmp);
  close(sockfd);
  cout << "sockABBClose : Connection to irb140 closed. " << endl;
}

void sockABB::rwSock(char* pos, string* msgOut) {
writeSock(pos);
readSock(msgOut);
// strcpy(msgOut, msg.c_str());
/* if (verbose) {
    cout << "rwSock : Read/Write done." << endl;
} */
}

void sockABB::sendQ(double* q) {
    char pos[256];
    string msgOut;
    q2msg(q, pos);
    // writeSock(pos);
    rwSock(pos, &msgOut);
    if (verbose) {
        cout << "sendQ : q = [ "
             << " ] done." << endl;
    }
}

void sockABB::q2msg(double* q, char* msg) {
    stringstream ss (stringstream::in | stringstream::out);
    string s;
    ss << "[ \" << q[0] << \", \" << q[1] << \", \" << q[2] << \", \"
    s = ss.str();
    if (s.length() <= 255) {
        strcpy(msg, s.c_str());
    }
    else {
        cout << "q2msg : Created string is too long. Message <msg> not created." << endl;
    }
}
Appendix B

RAPID

All rapid code is generated by matlab functions and have several parameters. The code shown below is therefore a example, showing how this generated code could look.

Arc Trajectory

%%
VERSION: 1.5
LANGUAGE: ENGLISH
%%
MODULE mod_testcode

PROC movee(

CONST robtarget StartPoint :=
[[515.000000, 0.000000, 200.000000],
[0.707107, 0.000000, 0.707107, 0.000000],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]],

CONST robtarget CirclePoint :=
[[515.000000, 70.710678, 170.710678],
[0.693520, −0.137950, 0.693520, −0.137950],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]],

CONST robtarget TargetPoint :=
[[515.000000, 100.000000, 100.000000],
[0.653281, −0.270598, 0.653281, −0.270598],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]]

);
[0, 0, 0, 0],
[0.9E9,9E9,9E9,9E9,9E9]];

CONST jointtarget INIT :=
[[0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000],
[0.9E9,9E9,9E9,9E9,9E9,9E9]];

ConfL \Off;
MoveJ StartPoint ,v1000\T:=2,z10,tool0;
MoveC CirclePoint, TargetPoint, v1000\T:=5,z10,tool0;
MoveAbsJ INIT,v200,fine,tool0;

ENDPROC

ENDMODULE

Scanner Trajectory

%%%
 VERSION: 1.5
 LANGUAGE: ENGLISH
%%%

MODULE mod_testcode

PROC movee()

CONST robtarget Target100 :=
[[515.000000, 0.000000, 240.000000],
[0.707107, 0.000000, 0.707107, 0.000000],
[0, 0, 0, 0],
[0.9E9,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target101 :=
[[538.917714, 41.426695, 235.126678],
[0.668129, −0.061350, 0.738970, −0.061350],
[0, 0, 0, 0],
[0.9E9,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target102 :=
[[486.556885, 49.264920, 232.897311],
[0.744288, −0.073176, 0.659792, −0.073176],
[0, 0, 0, 0],
[0.9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget Target103 :=
[[452.045038, 0.000000, 231.078411],
[0.794625, −0.000000, 0.607100, −0.000000],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];

CONST robtarget Target104 :=
[[481.175245, −58.586194, 229.457885],
[0.750363, 0.087483, 0.649345, 0.087483],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];

CONST robtarget Target105 :=
[[550.765324, −61.947358, 227.952435],
[0.645406, 0.092742, 0.752495, 0.092742],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];

CONST robtarget Target106 :=
[[589.866489, −0.000000, 226.517945],
[0.585741, 0.000000, 0.810498, 0.000000],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];

CONST robtarget Target107 :=
[[553.903992, 67.383691, 225.126616],
[0.638787, −0.101419, 0.755896, −0.101419],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];

CONST robtarget Target108 :=
[[474.775361, 69.671119, 223.758497],
[0.757315, −0.105155, 0.635892, −0.105155],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];

CONST robtarget Target109 :=
[[432.146610, 0.000000, 222.397556],
[0.821312, −0.000000, 0.570479, −0.000000],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];

CONST robtarget Target110 :=
[[472.467622, −73.668239, 221.029455],
[0.794625, −0.000000, 0.607100, −0.000000],
[0, 0, 0, 0],
[0, 9E9, 9E9, 9E9, 9E9, 9E9]];
[0.759791, 0.111856, 0.630630, 0.111856],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target111 :=
[[558.557990, -75.444652, 219.639984],
[0.628187, 0.114930, 0.760896, 0.114930],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target112 :=
[[604.031762, -0.000000, 218.213601],
[0.557777, 0.000000, 0.829991, 0.000000],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target113 :=
[[560.415646, 78.662207, 216.731653],
[0.623536, -0.120717, 0.762927, -0.120717],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target114 :=
[[468.735096, 80.133165, 215.169625],
[0.763879, -0.123497, 0.621277, -0.123497],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target115 :=
[[420.860374, 0.000000, 213.492106],
[0.837696, -0.000000, 0.546137, -0.000000],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target116 :=
[[467.164573, -82.853390, 211.641908],
[0.765723, 0.129017, 0.616747, 0.129017],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST robtarget Target117 :=
[[563.565950, -84.118693, 209.511205],
[0.614374, 0.131879, 0.766655, 0.131879],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST rrobotarget Target118 :=
[[613.529841, -0.000000, 206.833686],
[0.533822, 0.000000, 0.845597, 0.000000],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST rrobotarget Target119 :=
[[615.000000, -0.000000, 200.000000],
[0.525731, 0.000000, 0.850651, 0.000000],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST rrobotarget Target120 :=
[[565.000000, 86.602540, 200.000000],
[0.606961, -0.140694, 0.769421, -0.140694],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST rrobotarget Target121 :=
[[465.000000, 86.602540, 200.000000],
[0.769421, -0.140694, 0.606961, -0.140694],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST rrobotarget Target122 :=
[[415.000000, 0.000000, 200.000000],
[0.850651, -0.000000, 0.525731, -0.000000],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST rrobotarget Target123 :=
[[465.000000, -86.602540, 200.000000],
[0.769421, 0.140694, 0.606961, 0.140694],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];

CONST rrobotarget Target124 :=
[[565.000000, -86.602540, 200.000000],
[0.606961, 0.140694, 0.769421, 0.140694],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]];
CONST robotarget Target125 :=
[[615.000000, -0.000000, 200.000000],
[0.525731, 0.000000, 0.850651, 0.000000],
[0, 0, 0, 0],
[0,9E9,9E9,9E9,9E9,9E9]]; 

CONST jointtarget INIT :=
[[0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000],[0,9E9,9E9,9E9,9E9,9E9]], Confl \ Off;

MoveJ Target100, v100, z20, tool0;
MoveC Target102, Target101, v100,z20,tool0;
MoveC Target104, Target103, v100,z20,tool0;
MoveC Target106, Target105, v100,z20,tool0;
MoveC Target108, Target107, v100,z20,tool0;
MoveC Target110, Target109, v100,z20,tool0;
MoveC Target112, Target111, v100,z20,tool0;
MoveC Target114, Target113, v100,z20,tool0;
MoveC Target116, Target115, v100,z20,tool0;
MoveC Target118, Target117, v100,z20,tool0;
MoveC Target120, Target119, v100,z20,tool0;
MoveC Target122, Target121, v100,z20,tool0;
MoveC Target124, Target123, v100,z20,tool0;
MoveAbsJ INIT, v200, fine, tool0;
ENDPROC

ENDMODULE
Appendix C
Matlab

**OnlyMain.m**

clear;
clear;

cd /home/regen/Dropbox/matlab/Real2/

[Found, Tag, dx, dy, dz] = PhaseOne();

if ~Found
    PhaseTwo(Tag, dx, dy, dz);
end

**PhaseOne.m**

function [Found Tag dx dy dz] = PhaseOne()

    % Get pictures
    arc_t = 10; % time it takes for the arc
    n_pic = 10; % number of pictures to take (minimum 1)

    CreateArcTrajectory(350, 100, arc_t);
    system('./ftp_abb.sh');
    system('./RunStuff');

    pause(0.5);

    for n = 1:1:n_pic,
        [Im(:,:,:,n) dt] = TakePicture();
        pause((arc_t/n_pic) - dt);
    end
% make descriptors
for n = 1:1:length(Im(1,1,:)),
    \( H(:,n) = \text{HOG}(\text{Im}(::,n)) \);
end

% get dimentions
[dx, dy, dz] = GetDimensions(Im(:,1,1), Im(:,1,n_pic), 0.2, 'plot'); % Top

id = ClassifyObject(H);

% if not found:
if (id == 0)
    fprintf('New object\n');
    Name = input('Name: ', 's');
    Tag = input('Tag: ', 's');
    Found = 0;
    if AddToDB(Name, Tag, H, Im) == 1
        fprintf('Error when adding object to database\n');
        Found = 1; % Just to make it not preform phase 2...
    end
else if id == 1
    fprintf('Identified as empty scene\n');
    Found = 1;
    Tag = 'Empty Scene';
else
    Found = 1;
    load DB.mat;
    fprintf('Object identified as %s (%s)\n', DB(id).Name, DB(id).Tag);
    Tag = DB(id).Tag;
end

PhaseTwo.m

function [SurfaceInfo n] = PhaseTwo(Tag, dx, dy, dz)
P = DomeSpiral(dz, max(dx, dy), 1, pi/3);
\[ P(:, 1) = P(:, 1) + 515; \]
\[ \text{plot3}(P(:, 1), P(:, 2), P(:, 3)) \]
\[ Q = \text{GetQuadToC}(P); \]
\[ \text{Speed} = 100; \]
\[ \text{Zone} = 20; \]

Create Mod trajectory file, upload it then move.
CreateMod(P, Q, Speed, Zone);
\[ \text{system('./ftp_abb.sh');} \]
\[ \text{system('./RunStuff');} \]

% Maximum radius of Camera
\[ \text{CamMax} = 0.5; \]

% THIS PART SHOULD BE DONE BY A 3D SCANNER!!!!!!
% Read in a binary stl file without colors
\[ [x, y, z, n] = \text{stlread}(['mesh/' Tag 'MESH.stl']); \]
\[ [x, y, z, n] = \text{stlread}(\text{Tag}); \]

figure;
\[ \text{patch}(x, y, z, 'r'); \]
\[ \text{axis equal}; \]

% make it "lay" on the ground
\[ z = z - \text{min}(\text{min}(z)); \]

% Fix normals and too big surfaces
\[ [x y z n] = \text{FixSurface2}(x, y, z, n, \text{CamMax}); \]

load DB.mat;

num = length(DB);
\[ \text{SurfaceInfo} = \text{zeros}(5, 3, \text{length}(x(1,:))); \]

for \( t = 1: \text{num}, \) % had a , dz before...why? doesnt do shit?
\[ \text{DB(num).SurfaceInfo}(:, :, t) = [x(1, t) y(1, t) z(1, t);} \]
\[ x(2, t) y(2, t) z(2, t);} \]
\[ x(3, t) y(3, t) z(3, t);} \]
\[ n(1, 1, t) n(2, 1, t) n(3, 1, t);} \]
\[ n(1, 2, t) n(2, 2, t) n(3, 2, t);];} \]
end

save ( 'DB.mat' , 'DB' );

figure;
patch(x, y, z, 'g');
axis equal;

end
\end{}

CreateArcTrajectory.m

function [] = CreateArcTrajectory(radius, safety, varargin)
% Creates an arc ending at height <safety> and having an
% radius <radius>, optional parameters is time of movement and
% zone of movement.
% There is no output but and trajectory.mod will be created
% in the working
% directory. The Time is however always +2 seconds to get
% to the
% starting position of the arc.

optargin = size(varargin, 2);

if optargin == 0
    Time = 5;
    Zone = 10;
elseif optargin == 1
    Time = varargin{1};
    Zone = 10;
elseif optargin == 2
    Time = varargin{1};
    Zone = varargin{2};
end

Rot = makehgtform('xrotate',-pi/4);

Pos(1, :) = [515 0 radius 1];
Pos(2, :) = (Rot * Pos(1, :)')';
Pos(3, :) = (Rot * Pos(2, :)')';
Pos(:, 3) = Pos(:, 3) + safety;
Rot = GetQuadToC(Pos);

% open the file with write permission
fid = fopen('trajectory.MOD', 'w');

fprintf(fid, '%%%

VERSION: 1.5

LANGUAGE:

ENGLISH

MODULE mod_testcode

PROC movee()

    fprintf(fid, 'CONST robtarget StartPoint := [[%f, %f, %f], [%f, %f, %f, %f], [0, 0, 0, 0], [0,9E9,9E9,9E9,9E9]];\n', Pos(1, 1), Pos(1, 2), Pos(1, 3), Rot(1, 1), Rot(1, 2), Rot(1, 3), Rot(1, 4));

    fprintf(fid, 'CONST robtarget CirclePoint := [[%f, %f, %f], [%f, %f, %f, %f], [0, 0, 0, 0], [0,9E9,9E9,9E9,9E9]];\n', Pos(2, 1), Pos(2, 2), Pos(2, 3), Rot(2, 1), Rot(2, 2), Rot(2, 3), Rot(2, 4));

    fprintf(fid, 'CONST robtarget TargetPoint := [[%f, %f, %f], [%f, %f, %f, %f], [0, 0, 0, 0], [0,9E9,9E9,9E9,9E9]];\n', Pos(3, 1), Pos(3, 2), Pos(3, 3), Rot(3, 1), Rot(3, 2), Rot(3, 3), Rot(3, 4));

% Initial position
    fprintf(fid, 'CONST jointtarget INIT := [[0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000],\n', [0,9E9,9E9,9E9,9E9]);

    fprintf(fid, '\n\nConfL \n Off;');
    fprintf(fid, '\n\n');

    fprintf(fid, 'MoveJ StartPoint, v1000 \n T:=2, %i, tool0;\n', Zone);
    fprintf(fid, 'MoveC CirclePoint, TargetPoint, v1000 \n T=%i, z%i, tool0;\n', Time, Zone);
    fprintf(fid, 'MoveAbsJ INIT, v200, fine, tool0;\n');

    fprintf(fid, '\nENDPROC\n\nENDMODULE');

fclose(fid);

% view the contents of the file
type trajectory.MOD

end
GetQuadToC.m

function [ Q ] = GetQuadToC( pos )

[x y] = size(pos);

if y > 3
    pos(:, 4) = []; % removes potential scale-thingy
end

for i = 1: x,
    BadVec = [pos(i, 1) pos(i, 2) 0] - pos(i, :); % Where the TCP is pointing originally
    BadVec = BadVec / norm(BadVec);
    GoodVec = [515 0 0] - pos(i, :); % Where TCP should be pointing
    GoodVec = GoodVec / norm(GoodVec);
    OffRad = acos(dot(BadVec, GoodVec)); % how much we should rotate.
    ROT = makehgtform('yrotate', pi/2); % initial robot rotation matrix

    if OffRad > 0.01 % if it's at least 1/100 of a degree
        RAxis = cross(BadVec(1:3), GoodVec(1:3)); % Axis to rotate around
        RAxis = RAxis / norm(RAxis);
        AdjRot = makehgtform('axisrotate', RAxis, OffRad); % focusing rotation matrix
        ROT = AdjRot * ROT;
    end

    % Transform to Quatrinions
    q1 = (1/2) * sqrt(ROT(1, 1) + ROT(2, 2) + ROT(3, 3) + 1);
    q2 = (1/2) * (sign(ROT(3, 2) - ROT(2, 3)) * sqrt(ROT(1, 1) - ROT(2, 2) - ROT(3, 3) + 1));
    q3 = (1/2) * (sign(ROT(1, 3) - ROT(3, 1)) * sqrt(ROT(2, 2) - ROT(1, 1) - ROT(3, 3) + 1));
    q4 = (1/2) * (sign(ROT(2, 1) - ROT(1, 2)) * sqrt(ROT(3, 3) - ROT(1, 1) - ROT(2, 2) + 1));
\[ Q(i, 1:4) = [q1 \ q2 \ q3 \ q4]; \]
\[ Q(i, 1:4) = Q(i, 1:4) / \text{norm}(Q(i, 1:4)); \]

\textbf{TakePicture.m}

\texttt{function \{Picture \ Time\} = TakePicture()}
\texttt{tic;}
\texttt{grab_gray_image();}
\texttt{Picture = im2double(imread('image.pgm'));}
\texttt{Time = toc;}

\textbf{FixSurface2.m}

\texttt{function \{OutX \ OutY \ OutZ \ OutN\} = \text{FixSurface2}(InX, InY, InZ, InN, CamMax)}
\texttt{T_N = length(InX(1,:));}
\texttt{Triangles = zeros(3, 4, T_N);}
\texttt{Distances = zeros(1, 3);}
\texttt{Check_Triangles = 0;}
\texttt{OutN = zeros(3, 2);}
\texttt{for t = 1: T_N,}
\texttt{Triangles(:, :, t) = [InX(:, t) \ InY(:, t) \ InZ(:, t) \ InN(:, t)];}
\texttt{end}
\texttt{while \text{isempty} (Triangles)}
\texttt{T = Triangles(:, :, 1);}
\texttt{Triangles(:, :, 1) = [];}\]
\texttt{T_N = length(Triangles(1, 1, :));}
\texttt{CenterPoint = [sum(T(:, 1))/3 \ sum(T(:, 2))/3 \ sum(T(:, 3))/3];}
\texttt{for d = 1: 3,}
\texttt{Distances(d) = sqrt(sum((T(d, 1:3) - CenterPoint).^2));}
\texttt{end}
\texttt{if \text{max}(Distances) > CamMax}
\texttt{T = \text{Splitit3}(T);}
\texttt{end}
APPENDIX C. MATLAB

Triangles(:, :, T_N+1:T_N+2) = T;

continue;
end

OutX(:, Check_Triangles+1) = T(:, 1);
OutY(:, Check_Triangles+1) = T(:, 2);
OutZ(:, Check_Triangles+1) = T(:, 3);
OutN(:, :, Check_Triangles+1) = [CenterPoint' (CenterPoint'
+ T(:, 4))];

Check_Triangles = Check_Triangles + 1;
end
end

SplitIt3.m

function [Tout] = SplitIt3(T)

SidesLength = zeros(1, 3);
Lines = zeros(1, 3);

Lines(1, :) = T(1, 1:3) - T(2, 1:3);
Lines(2, :) = T(2, 1:3) - T(3, 1:3);
Lines(3, :) = T(1, 1:3) - T(3, 1:3);

SidesLength(1) = sqrt(sum(Lines(1,:).^2));
SidesLength(2) = sqrt(sum(Lines(2,:).^2));
SidesLength(3) = sqrt(sum(Lines(3,:).^2));

if SidesLength(1) >= SidesLength(2) && SidesLength(1) >= SidesLength(3)
    SplitPoint = T(1, 1:3) + (T(2, 1:3) - T(1, 1:3))/2;
    oppositePoint = 3;
elseif SidesLength(2) >= SidesLength(1) && SidesLength(2) >= SidesLength(3)
    SplitPoint = T(2, 1:3) + (T(3, 1:3) - T(2, 1:3))/2;
    oppositePoint = 1;
elseif SidesLength(3) >= SidesLength(1) && SidesLength(3) >= SidesLength(2)
    SplitPoint = T(3, 1:3) + (T(1, 1:3) - T(3, 1:3))/2;
    oppositePoint = 2;
else
    fprintf('Something is very wrong
');
otherTwo = [1 2 3];
otherTwo(otherTwo==oppositePoint) = [];
oppositePoint = T(oppositePoint, 1:3);

%% Creating the TWO triangles

x = [T(otherTwo(1), 1) T(otherTwo(2), 1);
     SplitPoint(1) SplitPoint(1);
     oppositePoint(1) oppositePoint(1)];

y = [T(otherTwo(1), 2) T(otherTwo(2), 2);
     SplitPoint(2) SplitPoint(2);
     oppositePoint(2) oppositePoint(2)];

z = [T(otherTwo(1), 3) T(otherTwo(2), 3);
     SplitPoint(3) SplitPoint(3);
     oppositePoint(3) oppositePoint(3)];

for t = 1: 2,
    Tout(:, 1, t) = x(:, t);
    Tout(:, 2, t) = y(:, t);
    Tout(:, 3, t) = z(:, t);
    Tout(:, 4, t) = T(:, 4);
end

end

GetDimensions.m

function [Dx Dy Dz] = GetDimensions(ImOldTOP, ImOldSIDE, s, varargin)

optargin = size(varargin,2);

[Sx, Sy] = size(ImOldTOP);

% X and Y
ImTOP = medfilt2(ImOldTOP, [10 10]);
G = fspecial('gaussian',[10 10],2);
ImTOP = imfilter(ImOldTOP,G, 'symmetric');
ImTOP = edge(ImTOP,'canny', s); %also good since it speeds up

% Z
APPENDIX C. MATLAB

```matlab
ImSIDE = medfilt2(ImOldSIDE, [10 10]);
G = fspecial('gaussian',[10 10],2);
ImSIDE = imfilter(ImOldSIDE,G, 'symmetric');
ImSIDE = edge(ImSIDE,'canny', s); %also good since it speeds up

if optargin > 0 && strcmp('plot', varargin(1))
    figure;
    imshow(ImTOP);
    hold on;
end

xVT = [];
yVT = [];
xVS = [];
yVS = [];

% Below part possible to make faster?
% Below part possible to make faster?
for x = 1: Sx,
    for y = 1: Sy,
      if ImTOP(x, y) > 0
        xVT(length(xVT)+1) = x;
        yVT(length(yVT)+1) = y;
      end
      if ImSIDE(x, y) > 0
        xVS(length(xVS)+1) = x;
        yVS(length(yVS)+1) = y;
      end
    end
end

if length(xVT) == 0 || length(xVS) == 0
    fprintf('No object found to measure\n');
    return;
end

% Pixel cm ratio (TOP) (specific on initial height)
PcRT = 0.31;

% Pixel cm ratio (SIDE) (specific on initial height)
PcRS = 0.08;

[rxT,ryT,area] = minboundrect(xVT,yVT);
```
\[ [rxS, ryS, area] = \text{minboundrect}(xVS, yVS); \]

\[
Dx = \text{PcRT} \times \sqrt{(rxT(1) - rxT(2))^2 + (ryT(1) - ryT(2))^2};
\]

\[
Dy = \text{PcRT} \times \sqrt{(rxT(2) - rxT(3))^2 + (ryT(2) - ryT(3))^2};
\]

\[
Dz = \text{PcRS} \times \text{abs}(\text{max}(\text{max}(ryS(1), ryS(2)), \text{max}(ryS(3), ryS(1)))) - (\text{min}(\text{min}(ryS(1), ryS(2)), \text{min}(ryS(3), ryS(1)))) / 2;
\]

\[
[Dx \ Dy \ Dz]
\]

if size(varargin, 2) && strcmp('plot', varargin(1))
    plot(ryT, rxT, 'r', 'LineWidth', 4);
end
end

ClassifyObject.m

function [id] = ClassifyObject(H)

PicNum = length(H(1, :));

% Do we have a network?
if exist('DB.mat') % We have!
    load 'DB.mat';
    DBL = length(DB);

    Pred = zeros(1, 10);
    for o = 1: length(DB),
        for n = 1: length(H(1, :)),
            Pred(o, n) = svmclassify(DB(o).SVM, H(:, n));
        end
    end

    [guarantee id] = max(sum(Pred'));
    guarantee = guarantee / length(H(1, :));

    if guarantee < 0.8
        id = 0;
        DBL = DBL + 1;

        DB(DBL).Name = [];
        DB(DBL).Tag = [];
        DB(DBL).Hog = H;
TrainSet = zeros(81, PicNum*DBL);
for t = 0: DBL-1,
    TrainSet(:, (PicNum*t)+1:(PicNum*t)+PicNum) = DB(t+1).Hog;
end

% Retrain/train networks
for n = 0: DBL-1,
    Targets = zeros(1, PicNum * DBL);
    Targets((n*PicNum)+1:(n*PicNum)+PicNum) = ones(1, PicNum);
    DB(n+1).SVM = svmtrain(TrainSet', Targets);
end
save ('DB.mat', 'DB');
end
else
    TrainSet = H;
    Targets = ones(1, PicNum);
    DB(1).SVM = svmtrain(TrainSet', Targets');
    DB(1).Name = [];
    DB(1).Tag = [];
    DB(1).Hog = H;

    save ('DB.mat', 'DB');
    id = 0;
end

AddToDB.m

function [ error ] = AddToDB(Name, Tag, Hog, Img)
cd /home/regen/Dropbox/matlab/Real2

% Loads old database if there exists one
if exist ('DB.mat')
    load ('DB.mat');
    n = length (DB);
else
    DB = [];
    save ('DB.mat', 'DB');
    n = 0;
end
end

pause (3);

% check to see if the object is not already in the DB
for m = 1: n,
    if strcmp(Name, DB(m).Name) || strcmp(Tag, DB(m).Tag)
        fprintf('Already in the Database, aborting.
');
        error = 1;
        return;
    end
end

DB(n).id = n;
DB(n).Name = Name;
DB(n).Tag = Tag;
DB(n).Mesh = ['mesh/ ' Tag 'MESH.stl '];
DB(n).Img = Img;

save('DB.mat', 'DB');

fprintf('Successfully added object to Database
');
error = 0;
end

DomeSpiral.m

function [ P ] = DomeSpiral(height, radius, dist, step)
    % DomeSpiral

    % minimum distance to table
    minH = 200;

    % Maximum Radius of Spiral (Needs to be investigated)
    MaxR = 1000;

    % Controlling size of sphere–spiral
    if radius > MaxR
        radius = MaxR;
    end

    % initial values
    P = [0 0 max(minH, height)];
% Number of rings
maxS = 20;

% Creating the spiral shaped dome
for t = 0: step: maxS-step,

    n = length(P(:, 1)) + 1;

    % Cartesian Coords
    % the ^ (1/4) needs to be investigated...
    P(n, 1) = (t/maxS)^(1/4) * radius * cos(t/dist);
    P(n, 2) = (t/maxS)^(1/4) * radius * sin(t/dist);

    if height > minH
        P(n, 3) = (height-minH) * sqrt(1 - (P(n, 1)/radius)^2 - (P(n, 2)/radius)^2) + minH;
    else
        P(n, 3) = minH;
    end

    P(n, 3) = real(P(n, 3));% Sometimes have a imaginary part :( 
end

% Creating the extra circle around the bottom.
tmp = t/dist; % just to make the circle start at the place % the spiral ended at
for o = t: step: 2*pi + tmp,
    n = length(P(:, 1)) + 1;

    P(n, 1) = radius * cos(o);
    P(n, 2) = radius * sin(o);
    P(n, 3) = minH;
end

P(1, :) = [];
end
Appendix D
Manual
References


