Danfoss Visual Inspection System

Design Document

Team Number

Dec1704

Client

Danfoss – Radek Kornicki

Advisers

Alexander Dogandzic

Team Members/Roles

Evan Woodring – Team Lead

Nicholas Gerleman – Key Concept Holder

Joseph Elliott – Communication Lead

Cory Itzen – Webmaster

Team Email

sendesign@iastate.edu

Team Website

Dec1704.sd.ece.iastate.edu

Revised: 17 Mar 2017 / Version 1.0.0

Contents

[1 Introduction 2](#_Toc476910633)

[1.1 Project Statement 2](#_Toc476910634)

[1.2 Purpose 2](#_Toc476910635)

[1.3 Goals 2](#_Toc476910636)

[2 Deliverables 3](#_Toc476910637)

[3 Design 4](#_Toc476910638)

[3.1 System Specifications 4](#_Toc476910639)

[3.1.1 Non-functional 4](#_Toc476910640)

[3.1.2 Functional 4](#_Toc476910641)

[3.1.3 Standards 4](#_Toc476910642)

[3.2 Proposed Design/Method 4](#_Toc476910643)

[3.3 Design Analysis 4](#_Toc476910644)

[4 Testing/Development 6](#_Toc476910645)

[4.1 Interface Specifications 6](#_Toc476910646)

[4.2 Hardware/Software 6](#_Toc476910647)

[4.2 Process 6](#_Toc476910648)

[5 Results 7](#_Toc476910649)

[5.1 RealSense Camera 7](#_Toc476910650)

[5.2 Other Thing 7](#_Toc476910651)

[6 Conclusions 8](#_Toc476910652)

[7 References 8](#_Toc476910653)

[8 Appendices 9](#_Toc476910654)

# 1 Introduction

## 1.1 Project Statement

Using a depth camera and a CAD model as a reference, our solution will detect errors in products (whether the errors are improperly placed parts or incorrect configurations) on the assembly line. Our solution will then notify the correct employees that something is wrong.

## 1.2 Purpose

The driving purpose of this project is to eliminate waste. An incorrectly configured product that has been shipped off to a client will always be shipped back. This is wasteful for the sellers, consumers, and shipping companies. Providing an accurate means of automated error detection will be beneficial to all involved parties because it saves money.

## 1.3 Goals

Regarding some preliminary goals, we would like to achieve the following:

* Successfully generate accurate 3D models of products.
* Create a mesh representation of the JT visualization format
* Successfully compare two 3D models to determine the location of errors, or lack thereof.
* Build a platform to scan products without human interaction.

Overall, our goal is to have a fully functional system on a test assembly line that can accurately report errors for any given product. We envision a photo booth-like area where products can be scanned with consistent lighting and rotation. We also imagine having the software run quickly, under 30 seconds.

# 2 Deliverables

Module to Generate Useful Point Clouds from All Incoming CAD File Formats

We expect to receive JT files [3] representing the products we will be scanning. To progress, we need to convert these JT files into a mesh representation such as an OBJ file for easy point cloud to point cloud comparisons.

Module to Generate 3D Models from Scanned Products

We are using the RealSense 3D camera to scan objects [5]. The SDK for this camera allows an easy generation of OBJ files representing the scanned objects.

Module to Align Multiple Point Clouds

It is necessary to align the two point clouds before calculating error between them. This process requires finding optimal scale, translation, and rotation to compensate for differences in orientation and physical positioning.

Module to Compare Two Point Clouds

The crux of the visual inspection system is the comparison of the two point clouds. This module will ensure that any two point clouds can be compared, and how probability that an error exists in the product.

End-to-End Prototype Built from the Previous Modules

These four modules allow us to build an end-to-end prototype. The prototype will be a simple linking of the four modules, tied together behind a user-friendly interface. This prototype can be used to identify pain-points in our methodologies and reduce risk for our final design.

Prototype Testing Results

Danfoss has happily granted us a facility to test our system. We will use this facility to get accurate reports on how well our prototype is working for real-world scenarios.

Revised System

With testing follows errors. We do not expect to have a completely bug-free system when we first reach the testing phase. We do, however, expect to expel all unexpected behaviors from the system before the deadline is met.

# 3 Design

## 3.1 System Specifications

The only given specifications were that we wanted to be able to detect error in a given object by comparing it to a 3-D model; most of the project has been left to our interpretation.

### 3.1.1 Non-functional

* The system shall be able to operate for extended periods of time without failure.
* The system shall be able to determine the error status of a product in under 30 seconds.
* The system shall be able to be used by employees without specialized knowledge
* The system shall be able to operate securely. All data will remain locally to Danfoss.
* The system shall be able to reliably determine the error status of a product.

### 3.1.2 Functional

* The system shall be able to use a CAD model as a reference object
* The system shall be able to scan an object of the size 3’x3’x3’
* The system shall be able to determine whether a product is generally defective
* The system shall be able to determine the area of a defect
* The system shall be able to generate 3D models available for later viewing.

### 3.1.3 Standards

Standard industry practices will be followed to reduce defects in code as well as improve overall quality. All changes made to the production system must be reviewed by other members of the team. Automated tests on these components will be created in the form of unit and integration tests. Regression testing will be performed to prevent adding defects to already working components. A strict set of coding conventions has been laid out to increase understandability of the code and prevent common types of errors.

## 

## 3.2 Proposed Design/Method

The process to achieve our goals can be broken up into several discrete steps. The first phase involves obtaining point clouds from both the physical object and CAD model. These point clouds are then compared to determine optimal alignment. We use these aligned point clouds to determine possible error in the system. The scan is repeated if there is low certainty of the existence of an error. The system finally alerts the user to either the absence of error or shows where it believes an error might have occurred.

The included SDK for the Intel RealSense camera provides support for three dimensional scanning of objects [5]. This process will generate a mesh file used for later consumption. This mesh represents a composite of depth captures from different angles around the object. Vertices from this mesh directly translate to a point cloud representing the object surface [5].

Creating a point cloud from a JT file first requires converting the JT file to a mesh. This has been possible using commercial tooling [6] but has proved challenging to do programmatically. This leads to our current approach of using a tool to export the file as a mesh before feeding it to our system. Based on client communication this seems to be an adequate solution.

Vertices from this mesh are not directly translatable to a surface point cloud. The vertices in the mesh are sparse and only placed where required to create ideal geometry. Density may be arbitrarily added to this mesh using a process known as tessellation [7]. In our instance we are able to calculate the area of each triangular face to determine if it should be split. The single triangular face may be split into three which meet in the centroid of the original face. This process will allow us to create a dense surface suitable for error detection and alignment.

The point clouds representing the physical and ideal objects must be accurately scaled and aligned to detect error. A rudimentary method to do this is ensuring alignment during capture and manually accounting for the offset in distance and model sizes. This method lacks robustness and increases the manual work necessary to use the system. This process is instead done algorithmically using the Iterative Closest Point method [2]. The method approximates an optimal linear transformation to minimize mean squared error between point clouds. It has many pre-existing implementations, reducing difficulty in implementation.

The heuristic used for localized error detection will require large amount of testing and tweaking. Our current approach centers on detecting clusters of points in the scanned object that have no correspondence to points in the ideal object. The points from the ideal object will first be placed into a data structure known as a KD Tree [1]. This allows us to quickly determine the closest point in the actual object to a point existing in the capture.

The distance from each point in the scan to its closest object in the ideal model is recorded. The standard deviation from these distances is recorded as a metric to differentiate noise in the camera from actual error. Capture points more than a standard deviation away from the closest ideal point are marked as potential error. A bounding box or convex hull is created around tight clusters of these points to indicate areas where consistent error is detected.

## 3.3 Design Analysis

The process for converting a JT file to a mesh has been tested. Commercial tools are able to accurately create a mesh from a given file.

We have begun development of the module to scan a physical object into a point cloud. This module utilizes the RealSense camera to generate an OBJ file for consumption by a later module. From our testing, we’ve noticed some problems with the RealSense camera. First, the camera generates very noisy point cloud representations of our real-world objects. This has problem cumbersome in testing as we can’t get a detailed representation to compare with the CAD models. We have so far only been able to retrieve low-accuracy 3D captures.

Several plans exists to mitigate potential issues posed by the camera. We are first wanting to test the camera in a more controlled environment. Over the next couple of weeks we plan to build a controlled environment suitable for testing the camera in ideal conditions. The results of these tests will have a large impact on our future plans. We have a second camera, the Occipital Structure Camera, available for testing. If the RealSense camera proves to be insufficient for our needs it provides a clear path for experimentation. A third alternative exists in changing the algorithm used for matching and error detection. Our current algorithms operate on a full scan of a 3D object. This algorithm can be modified to instead work on single captures from a given angle. This introduces extra complexity into our project but side-steps the issue of low accuracy stitching from the camera.

# 4 Testing/Development

## 4.1 Interface Specifications

The Intel RealSense camera is the only hardware that will be interacting with the software (aside from the main computer).

## 4.2 Hardware/Software

Autodesk Inventor is currently being used for JT file conversion. It is used to select configurations from a JT file and convert it to a more usable format, such as OBJ or STL. Though this is being used for testing, we likely will be able to use this for the final product as well.

MeshLab is an open-source tool that allows visualization and manipulation of 3D mesh data [8]. This tool has allowed us to quickly visualize the quality of 3D captures generated by the RealSense camera as well as the mesh generated from a sample JT file.

3D Builder is a 3D object creation tool built into Windows 10. It has been used to create simple meshes for testing purposes.

The Prusa i3 is an open hardware 3D printer. It has allowed us to build reference objects from our own meshes to obtain real data from synthetic objects.

## 4.2 Process

To start testing our project, it made sense to start with testing the cameras at our disposal. Until recently, the only camera we had to work with was the Intel RealSense camera. Intel was gracious enough to provide sample code as part of their SDK, so we were able to start scanning right away. We started with scanning anything and everything that was considered an "around the house" item. The scans included books, computer monitors, and even people.

From this, we noticed a few things:

* The infra-red light coming from the camera would pass through any transparent panels (for example, on a signal generator's display).
* The camera could pick up details as small as wrinkles on a t-shirt, provided the wrinkles didn't move as the scanning commenced.
* It was impossible to scan underneath the object, for obvious reasons.
* The objects scanned frequently had bumpy surfaces instead of smooth.
* It was very easy to misalign the camera during the scanning process (largely due to human error).

We then moved on to the files we will be comparing our scans to. For JT conversion testing, we tried many different programs, including JT Open and JT Assistant. Sadly, most of the programs we tested were either not able to open the file due to the size, or were unable to export in a usable format. The only one we found that worked was Autodesk Inventor, which we are currently using. We likely will not need to do much more, but could look into creating a program which does the selection and exporting for us.

Synthetic 3D models resembling parts were created for testing. These models were printed in order to obtain real data from the 3D camera corresponding to a 3D model.

It was decided early on that a “test bench” should be created to validate our algorithms for reconstruction, tessellation, alignment, and error detection. This tool is designed with several goals in mind. The tool is able to visualize point clouds and meshes at different stages of the transformation pipeline. The ability to visualize and manipulate these structures allows for visual debugging and manipulation of the mentioned algorithms. The tool allows algorithms and their parameters to be switched at runtime. This allows for rapid improvement, validation, and comparison of approaches for different areas of the problem. The tool finally provides simple abstractions to allow us to develop these algorithms more quickly. This tool is nearing completion and should be finished within the next 1-3 weeks.

# 5 Results

### 5.1 RealSense Camera

The Intel RealSense camera has shown its fair share of difficulties. To start, the SDK and documentation have been lacking in critical areas. There have been multiple times where *crucial code* has been missing from either the samples or documentation. This has drastically slowed down the development process. After much trial and error we have been able use the SDK to programmatically scan an object.

The RealSense camera performed poorly when tested against our printed models. In the generated model, there were gaps, alignment issues, incorrect origin placements, and odd sections of the platform were picked up. This lead to an extremely noisy point cloud, unfit for error detection.

The results obtained were done so in an uncontrolled environment. To work with this camera, we plan on making a center for consistent testing. Right now, the camera may be performing poorly because we don’t have an exact platform for the camera and object to rest on, so there’s noise generated from human movement. This is the best course of action to remedy this camera. If this doesn’t work, we may have to switch development time to the Occipital Structure camera.

### 

# 6 Conclusions

Our goal with the project plan is to show that it’s possible to compare a real-world object to a 3D model. The plan revolves around generating a proof of concept. We can achieve this by the following plan, laid out in detail in the Deliverables section. To give a quick summary, we will do the following:

* Show that we can make a point cloud out of the provided CAD models.
* Show that we can make point clouds out of real world objects.
* Show that we can compare the previous two point clouds to determine the presence of errors.

This plan allows us to achieve the proof of concept we are striving for. It further allows us to pursue a production-quality system. Right now, we’ve completed very experimental prototypes of the modules. Each of these are ready for robust testing. Once we’ve tested, we will be able to progress in combining our modules for a full solution.

# 7 References

[1] "KD Tree". Data structure used for spatial indexing.

<https://en.wikipedia.org/wiki/K-d_tree>

[2] "Iterative Closest Point". Algorithm developed for point set registration.

<https://en.wikipedia.org/wiki/Iterative_closest_point>

[3] "JT file format". File format used for object visualization.

<https://www.freecadweb.org/tracker/file_download.php?file_id=503&type=bug>

[4] "PCL – Point Cloud Library". Contains documentation on the modular point cloud libraries, software installations, and a comprehensive list of tutorials covering various aspects of the PCL.

[5] "RealSense SDK". Documentation pertaining to the camera we plan to use.

<https://software.intel.com/en-us/intel-realsense-sdk-support>

[6] "Autodesk Inventor". CAD application for creating 3D digital prototypes

<http://www.autodesk.com/products/inventor/overview>

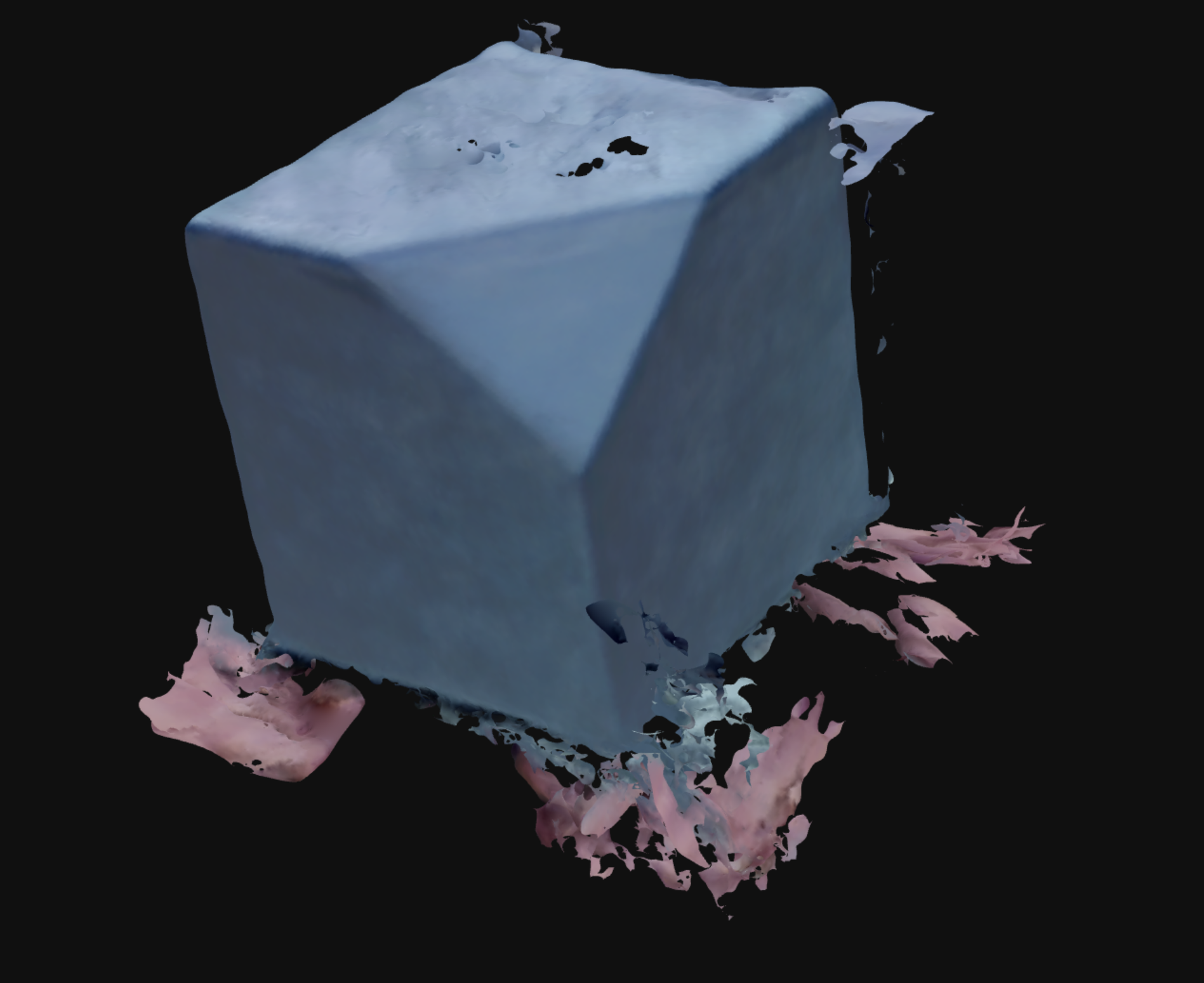
[7] "Delaunay Tessellation Field Estimator". Mathematical Tool for reconstructing a volume-covering from a discrete point set.

<https://en.wikipedia.org/wiki/Delaunay_tessellation_field_estimator>

[8] "Meshlab". Open source software for viewing 3D models.

<http://www.meshlab.net/>

# 8 Appendices



An attempted scan during testing. The original object is a cube with a corner sliced off. This attempted scan showed the errors humans introduce to the scanning process. On the bottom, the skin-colored blobs are actual hands that were unintentionally captured. Furthermore, the top right shows an extra chunk of gray. This was from the camera losing alignment and attempting to readjust itself. Lastly, notice how the surfaces aren't completely smooth, and even have bits missing at the top.