

Eivind Torsrud Nerol

# AR Assembly Instructions based on a 3D-Camera-Projector System

Master's thesis in Robotics and Automation

Supervisor: Lars Tingelstad

October 2021

**NTNU**  
Norwegian University of Science and Technology  
Faculty of Engineering  
Department of Mechanical and Industrial Engineering



Norwegian University of  
Science and Technology





Eivind Torsrud Nerol

# **AR Assembly Instructions based on a 3D-Camera-Projector System**

Master's thesis in Robotics and Automation  
Supervisor: Lars Tingelstad  
October 2021

Norwegian University of Science and Technology  
Faculty of Engineering  
Department of Mechanical and Industrial Engineering





# Preface

This Master Thesis was written in collaboration with SINTEF Manufacturing in Trondheim during the spring of 2021 and completes my master's degree from the Department of Mechanical and Industrial Engineering at the Norwegian University of Science and Technology.

SINTEF Manufacturing has, in collaboration with Aker Solutions Verdal, NTNU, and other actors, initiated a project, Autokons. The project addresses challenges in the manufacturing procedure of jackets where the stub assembly procedure is a central part. Today's process involves multiple manual operations that are time-consuming. Manual operations are slow, which leads to a less time-efficient assembly procedure. This led me to research an augmented reality assembly marking system that marks a stub section cut for grinding.

Several people have contributed with valuable input, help, and support during this process. First, I want to thank my supervisor, Associate Professor Lars Tingelstad, for his support and guidance. Secondly, I would like to express my gratitude to Eirik Njåstad, Morten Lind, and Pål Ystgaard at Sintef Manufacturing for graciously taking the time to work with me on this project. Thirdly, I want to thank Marco Musy for his guidance related to issues in Vedo, and Martin Ingvaldsen at Zivid AS for valuable input regarding the use of their cameras. Finally, I would like to thank my family and fellow students for their valuable inputs and patience.



# Abstract

Since 1975, Aker Solutions Verdal has provided steel jacket substructures for off-shore oil and gas platforms. The yard in Verdal has a strong position in the market. Nevertheless, the global steel-price impact and the high salaries in Norway contribute to the yard being less competitive and losing contracts to foreign yards. Automation in the production would lead to a more modern yard with faster production, higher accuracy, and less staffing need to ensure more contracts in the future.

SINTEF Manufacturing has, in collaboration with Aker Solutions Verdal initiated a project, AutoKons. The project addresses challenges in the manufacturing procedure of jackets, which are often one-piece productions. Such one-piece jackets consist of large, heavy components that face challenges within geometric accuracy and time-consuming tasks such as cutting, lifting, marking, and welding.

A jacket is a structure built of round tubes and typically has four or more legs connected by a series of bracings. When assembling the bracing onto the leg, a stub is used to connect the bracing to the leg. Instead of mounting the bracing directly onto the leg, a stub makes it easier to align the bracing in the right position and angle. Before the stub is welded onto the leg, embers and welding seams must be grinded away in the area where the stub hits the leg. The stub is lifted up after its cross-section cut has been cut out in the desired configuration and marked with chalk to know where to grind. In this operation, one is dependent on cranes which are considered time-consuming and delay other tasks in the manufacturing area.

This thesis investigates creating a system consisting of a 3d camera with a projector based on avoiding the above-mentioned crane operation for marking. Instead of lifting and marking manually, the stub's cross section cut is marked with a projector. The projected image will highlight the area on the leg's surface on which the sheet metal worker/welder should grind. An integration of this in the current procedure can lead to fewer crane operations, and thus a more efficient installation.

Various experiments are performed with synthetic and real data where stubs in

different configurations are transformed from the defined reference system. With certain assumptions, such as the reference system and the projected cross-section not being compared to a real cross-section in the same configuration, it is seen that results are obtained close to the position tolerances for tubular nodes in the standard NORSOK-M101, *Structural steel fabrication*.

The implemented method also has the possibility of being extended to other applications within the current stub assembly procedure. The intersection path between the meshes in the implemented method can also be used in automation of cutting and welding by the use of robotic systems with hand-eye calibration.

# Sammendrag

Siden 1975 har Aker Solutions Verdal produsert stål “jacket” understell for offshore olje- og gassplattformer. Verftet i Verdal har en sterk posisjon i markedet, men på grunn av den globale høye stålprisen og de høye lønningene i Norge, så fører det til at verftet er mindre konkurransedyktig og mister kontrakter til andre utenlandske verft. Automasjon i produksjonen av jacketer kan føre til et mer moderne verft med raskere produksjon, bedre nøyaktighet og mindre bemanning som sikrer flere kontrakter inn i fremtiden.

SINTEF Manufacturing har initiert et prosjekt, AutoKons, sammen med Aker Solutions Verdal som undersøker problemer i fabrikkasjonsprosedyren av jacketer. Slike konstruksjoner er ofte av kategorien “et stykk” produksjon og består av tunge dimensjoner som fører til problemer innen geometrisk nøyaktighet med tidskrevende operasjoner som kutting, løfting, markering og sveising.

En jacket er en struktur som er bygd av runde rør og har typisk fire “leger” som knyttes sammen av en rekke avstivere, “bracings”. Når bracingen skal monteres på legen kommer stub montasjen inn. Istedenfor å montere bracingen direkte på legen, kommer stuben inn som et lite koblet rørstykke mellom leg og bracing. Stubens oppgave er å gjøre det enklere å justere bracingen slik at den kommer i riktig posisjon og vinkel. Før stuben sveises på legen, så må glødeskall og sveisesømmer fjernes med sliping i området der tverrsnittet til stuben treffer legens overflate. For å vite hvor man skal slipe løftes stuben opp etter at dens tverrsnittet er kuttet ut i ønsket konfigurasjon, for å så markere med kritt. I denne operasjonen er man avhengig av kran, noe som blir ansett som tidskrevende og sinker andre oppgaver i fabrikkasjonsområdet.

Denne masteroppgaven har undersøkt å lage et system bestående av et 3D kamera og en projektor med utgangspunkt i å unngå overnevnte kranoperasjon for markering. Istedenfor å løfte opp og markere manuelt, så markeres stubens tverrsnitt med en projektor. Bildet som deretter blir projisert igjennom projektorens bildeplan vil fremheve området på legens overflate, der platearbeideren/sveiseren skal slipe istedenfor å løfte opp og manuelt markere. En integrasjon av dette i dagens prosedyre kan føre til færre kranoperasjoner, og dermed en mer effektiv montering.

Det er gjennomført ulike tester med både syntetisk og ekte data der stuber er definert i ulike konfigureringer med en transformasjon fra det definerte referansesystemet. Med gitte antagelser som at referansesystemet og det projiserte tverrsnittet ikke er sammenlignet mot et virkelig tverrsnitt i samme konfigurasjon, så ser man at resultatene er nær posisjonstoleransene for rørknutepunkt i standarden NORSOK-M101, *Structural steel fabrication*.

Skjæringskurvaturen som blir definert av skjæring mellom meshene i den implementerte metoden kan også nyttes inn i andre applikasjoner i dagens stubmontasje som kutting og sveising ved å flytte kurvatures koordinater over i et robotkoordinatsystem ved hjelp av en hånd-øye kalibrering.



# Contents

<b>Preface</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Sammendrag</b>	<b>v</b>
<b>1. Introduction</b>	<b>1</b>
1.1. Background . . . . .	1
1.2. Problem description and objectives . . . . .	2
1.3. Previous work . . . . .	4
1.3.1. SINTEF Manufacturing . . . . .	4
1.3.2. Specialization project . . . . .	4
1.4. Limitations . . . . .	5
1.5. Outline . . . . .	6
<b>2. Stub assembly for jacket Fabrication</b>	<b>7</b>
2.1. Preliminary . . . . .	7
2.2. Fabrication and dimensions . . . . .	9
2.2.1. Tolerances . . . . .	10
2.3. Stub assembly procedure . . . . .	11
2.3.1. Stub cross-section cut . . . . .	12
2.3.2. Lifting up and marking . . . . .	13
2.3.3. Lifting down and grinding . . . . .	15
2.3.4. Installation and welding . . . . .	15
<b>3. Preliminaries</b>	<b>17</b>
3.1. Computer vision . . . . .	17
3.1.1. Rotation matrices . . . . .	17
3.1.2. Homogeneous transformation matrices . . . . .	18
3.1.3. Pinhole camera model . . . . .	18
3.1.4. Corner detection . . . . .	22
3.1.5. Camera calibration . . . . .	22
3.1.6. OpenCV . . . . .	23

3.1.7. Zhang’s method . . . . .	23
3.2. Point clouds . . . . .	26
3.2.1. Iterate closest point . . . . .	27
3.2.2. Rotate point clouds using normals . . . . .	27
3.2.3. Covariance matrix . . . . .	28
3.3. Mesh . . . . .	29
3.3.1. Delauney triangulation . . . . .	29
3.3.2. Mesh intersection . . . . .	30
3.3.3. Signed distance . . . . .	31
<b>4. Method</b>	<b>33</b>
4.1. Dependencies . . . . .	34
4.1.1. Hardware . . . . .	34
4.1.2. Software . . . . .	37
4.2. Capture point clouds . . . . .	38
4.2.1. Blender point cloud . . . . .	38
4.2.2. Zivid point cloud . . . . .	39
4.3. Pointcloud processing . . . . .	41
4.3.1. Downsampling and outlier removal . . . . .	42
4.3.2. Cylinder fitting . . . . .	44
4.3.3. Reference system . . . . .	45
4.4. Mesh processing . . . . .	49
4.4.1. Mesh intersection . . . . .	50
4.5. Image projection . . . . .	51
4.5.1. Calibration . . . . .	51
4.5.2. Create projected image . . . . .	52
4.6. Evaluation . . . . .	54
<b>5. System</b>	<b>59</b>
5.1. Blender . . . . .	59
5.1.1. Camera projector setup . . . . .	60
5.1.2. Scene . . . . .	62
5.1.3. Calibration . . . . .	62
5.2. Lab . . . . .	63
5.2.1. Zivid Two-Acer Predator system . . . . .	65
5.2.2. Calibration . . . . .	67
<b>6. Experiment</b>	<b>69</b>
6.1. Synthetic . . . . .	69
6.2. Lab . . . . .	73
6.2.1. Pose 1 . . . . .	73
6.2.2. Pose 2 . . . . .	83

<b>7. Discussion</b>	<b>89</b>
7.1. Pipeline	89
7.1.1. Capture point clouds	89
7.1.2. Point cloud processing	91
7.1.3. Mesh processing	92
7.1.4. Image projection	93
7.1.5. Evaluation	94
7.2. Requirements	95
7.2.1. Position and reference system	95
7.2.2. Size	96
7.2.3. Transforming	96
7.3. Experiment	96
7.4. Scaling and today’s procedure	98
7.4.1. Stub section cut	98
7.4.2. Lifting up and down	99
7.4.3. Scaling	101
<b>8. Conclusion</b>	<b>103</b>
8.1. Further Work	104
<b>A. Fabrication Tolerances</b>	<b>109</b>
A.1. Tubulars	109
A.2. Tubular nodes	111
<b>B. Zivid and Projector Pair Calibration</b>	<b>113</b>
B.1. Projector calibration	113
B.1.1. Generate checkerboard image	114
B.1.2. Capture zivid image frames	115
B.1.3. Rotate 3D points using eigenvectors	116
B.1.4. Projector intrinsics and Zivid depth camera extrinsics	119
<b>C. Additional Figures</b>	<b>121</b>
<b>D. Mathematics</b>	<b>123</b>
D.1. Centering matrix	123
<b>E. Data-Sheet</b>	<b>127</b>
E.1. Zivid Two	127
E.2. Acer Predator Z650	141



# List of Figures

1.1. System Overview . . . . .	3
2.1. Jacket parts . . . . .	7
2.2. Jacket Detail . . . . .	8
2.3. Leg Section . . . . .	9
2.4. Welding Seams . . . . .	9
2.5. Tolerances . . . . .	10
2.6. Fabrication area [5] . . . . .	11
2.7. CNC cutting machine . . . . .	12
2.8. Section Cuts . . . . .	13
2.9. Lift stub onto leg . . . . .	14
2.10. Grinding area . . . . .	14
2.11. Stub lifted down . . . . .	15
3.1. Pinhole camera model . . . . .	19
3.2. Camera and world coordinate systems . . . . .	20
3.3. Object Planes . . . . .	24
3.4. ICP . . . . .	27
3.5. Delauney Triangulation . . . . .	29
3.6. Mesh intersection . . . . .	31
3.7. Signed distance . . . . .	31
4.1. Software Pipeline Diagram . . . . .	33
4.2. Hardware Pipeline . . . . .	34
4.3. Camera and projector parameters . . . . .	35
4.4. Rig setup example . . . . .	36
4.5. 3D-camera-projector setup . . . . .	36
4.6. Capture point cloud in Blender API . . . . .	38
4.7. Blender Capture . . . . .	39
4.8. Point cloud . . . . .	39
4.9. Pixel correspondence Zivid . . . . .	40
4.10. Capture Point Cloud Zivid . . . . .	40
4.11. Zivid Capture . . . . .	41

4.12. Point cloud processing . . . . .	42
4.13. Downsampling Zivid . . . . .	42
4.14. Zivid Aruco marker . . . . .	43
4.15. Cylinder fitting . . . . .	44
4.16. Reference system . . . . .	45
4.17. System variables . . . . .	46
4.18. Initial stub configuration . . . . .	47
4.19. Stub rotation . . . . .	47
4.20. Stub rotation . . . . .	48
4.21. Stub Multiple configuration . . . . .	48
4.22. Stub translation configuration . . . . .	48
4.23. Mesh processing . . . . .	49
4.24. Mesh generation . . . . .	49
4.25. Mesh Intersection in $\{r\}$ . . . . .	50
4.26. Intersection lines . . . . .	51
4.27. Image projection . . . . .	51
4.28. Mapping coordinates to projector image plane . . . . .	52
4.29. Creation of the projection image . . . . .	53
4.30. Projector image on leg's surface . . . . .	53
4.31. Pixels of $A_g$ after projection . . . . .	54
4.32. Grinding Area points . . . . .	55
4.33. Difference in grinding Area points . . . . .	56
4.34. Difference in grinding Area points . . . . .	56
4.35. Difference in grinding Area points . . . . .	57
5.1. Blender System . . . . .	59
5.2. Blender 3D-camera-projector setup . . . . .	61
5.3. Rig setup example . . . . .	62
5.4. Leg lab specimen . . . . .	63
5.5. Lab setup . . . . .	64
5.6. Lab system . . . . .	65
5.7. Zivid-Projector system . . . . .	67
6.1. Stub Configuration Synthetic . . . . .	70
6.2. Projector images Pose 1 . . . . .	70
6.3. Threshold image Pose 1 . . . . .	70
6.4. Difference in predicted and projected Pose 1 . . . . .	70
6.5. Detail view in predicted and projected Pose 1 . . . . .	71
6.6. Signed Distance . . . . .	71
6.7. $A_{gap}$ aligned with ICP to $A_g$ . . . . .	71
6.8. Projected image in Blender . . . . .	72
6.9. Histogram Signed distance . . . . .	72

6.10. Poses lab . . . . .	73
6.11. Stub Configuration Pose 1 Experiment 1 . . . . .	74
6.12. Projected image pose 1 . . . . .	74
6.13. Threshold image Pose 1 . . . . .	74
6.14. Difference in predicted and projected Pose 1 . . . . .	75
6.15. Detail view in predicted and projected Pose 1 . . . . .	75
6.16. Signed distance between $A_g$ and projected $A_{gap}$ Pose 1 . . . . .	75
6.17. $A_{gap}$ aligned with ICP to $A_g$ . . . . .	76
6.18. Distribution of signed distance between $A_{gap}$ and $A_g$ . . . . .	76
6.19. Stub Configuration Pose 1 Experiment 2 . . . . .	77
6.20. Projected image pose 1 . . . . .	77
6.21. Threshold image Pose 1 . . . . .	77
6.22. Difference in predicted and projected Pose 1 . . . . .	78
6.23. Detail view in predicted and projected Pose 1 . . . . .	78
6.24. Signed distance between $A_g$ and projected $A_{gap}$ Pose 1 . . . . .	78
6.25. $A_{gap}$ aligned with ICP to $A_g$ . . . . .	78
6.26. Distribution of signed distance between $A_{gap}$ and $A_g$ . . . . .	79
6.27. Stub Configuration Pose Experiment 1 . . . . .	80
6.28. Projected image pose 1 . . . . .	80
6.29. Threshold image Pose 1 . . . . .	80
6.30. Difference in predicted and projected Pose 1 . . . . .	81
6.31. Detail view in predicted and projected Pose 1 . . . . .	81
6.32. Signed distance between $A_g$ and projected $A_{gap}$ Pose 1 . . . . .	81
6.33. $A_{gap}$ aligned with ICP to $A_g$ . . . . .	82
6.34. Distribution of signed distance between $A_{gap}$ and $A_g$ . . . . .	82
6.35. Stub Configuration Pose 2 Experiment 1 . . . . .	83
6.36. Projected image pose 1 . . . . .	83
6.37. Threshold image Pose 1 . . . . .	84
6.38. Difference in predicted and projected Pose 1 . . . . .	84
6.39. Detail view in predicted and projected Pose 1 . . . . .	84
6.40. Signed distance between $A_g$ and projected $A_{gap}$ Pose 1 . . . . .	85
6.41. $A_{gap}$ aligned with ICP to $A_g$ . . . . .	85
6.42. Distribution of signed distance between $A_{gap}$ and $A_g$ . . . . .	86
6.43. Stub Configuration Pose 2 Experiment 1 . . . . .	86
6.44. Projected image pose 1 . . . . .	86
6.45. Threshold image Pose 1 . . . . .	87
6.46. Difference in predicted and projected Pose 1 . . . . .	87
6.47. Detail view in predicted and projected Pose 1 . . . . .	87
6.48. Signed distance between $A_g$ and projected $A_{gap}$ Pose 1 . . . . .	88
6.49. $A_{gap}$ aligned with ICP to $A_g$ . . . . .	88
6.50. Distribution of signed distance between $A_{gap}$ and $A_g$ . . . . .	88

7.1. Boundaries . . . . .	94
7.2. System requirements . . . . .	95
7.3. Intersection path with the cut . . . . .	98
7.4. Interpolation . . . . .	99
7.5. Projection in lab . . . . .	100
7.6. Stub configuration larger than the captured point cloud. . . . .	102
7.7. Interpolation . . . . .	102
A.1. Tubulars - Circumferential tolerance . . . . .	109
A.2. Ovality . . . . .	110
A.3. Node stub location . . . . .	111
B.1. Checkerboard pattern . . . . .	114
B.2. Checkerboard poses . . . . .	115
B.3. Checkerboard coordinates . . . . .	116
B.4. Checkerboard coordinates centroid . . . . .	117
B.5. Eigenvectors . . . . .	118
B.6. Checkerboard coordinates aligned with the XY-Plane . . . . .	118
B.7. Frames . . . . .	120
C.1. Chessboard 11x17, square size 60 pixels . . . . .	121



# List of Tables

5.1. Zivid Two Technical Specifications . . . . .	65
5.2. Acer Predator Z650 Technincal Specifications . . . . .	66
6.1. Synthetic Experiment . . . . .	69
6.2. Synthetic Transformation . . . . .	72
6.3. Pose 1 lab setup experiment . . . . .	73
6.4. Lab pose 1 transformation experiment 1 . . . . .	76
6.5. Lab pose 1 transformation experiment 2 . . . . .	79
6.6. Lab pose 1 transformation experiment 3 . . . . .	82
6.7. Pose 2 Experiment . . . . .	83
6.8. Lab pose 2 transformation experiment 1 . . . . .	85
6.9. Lab pose 2 transformation experiment 2 . . . . .	88



# Glossary

**bracing** Stay tubes that form trusses between the legs. [1](#)

**can** Can's amplifies the node, this is just one pipes with greater material thickness that connect a set of bracings together.. [8](#)

**circular welding seam** Large round tubes are produced from rolled plates welded together into a round tube; the weld seam is a longitudinal seam. [9](#)

**jacket** Bottom fixed offshore construction of round tubes in trusses, also called steels substructures or just chassis. The name jacket may come from the construction became built as a jacket around the bottom attachment, i.e., the poles, because wooden poles became too short in deep water. [1](#)

**leg** Round tubes with varying thickness and diameter, which operate as load-bearing on a jacket. A jacket has 4 to 8 legs depending on the size and the purpose of the jacket. The leg's connection to the seabed is via clusters and piles. [1](#)

**longitudinal welding seam** Large round tubes are produced from rolled plates welded together into a round tube; the weld seam is a longitudinal seam. [9](#)

**stub** Transition coupling between the bracing and the leg. [1](#)



# Chapter 1.

## Introduction

This chapter presents a brief background to the existing assembly problem and the studied objective. Thenceforth, the purpose of this master thesis is formulated, followed by the given limitations. Finally, an outline of the thesis concludes the chapter.

Some of the content in this chapter is based on the author's specialization project written as part of TPK4560 at NTNU [1].

### 1.1. Background

Since the onset of commercial-size crude oil recovery in the late 19th century, the global oil demand has led to an increase in offshore construction and fabrication. In connection with oil extraction, one is dependent on large offshore platforms. Offshore platforms are large steel structures with facilities for well drilling to explore, extract, store, and process petroleum and natural gas that lies in rock formations beneath the seabed [2]. Such extensive steel substructures vary in size, depending on the oil field location.

The fabrication process for manufacturing large steel substructures involves lifting, grinding, cutting, welding, and assembling heavy steel components. Multiple of these tasks are manually executed and lead to time-consuming operations that can be made more efficient by the increased development of production technology within robotic vision.

When a significant steel substructure, a [jacket](#), is settled for fabrication, [legs](#), [bracings](#), [stubs](#), and other components are essential for constructing a strong truss structure. The jacket usually has four legs with several stubs in each elevation that connect the bracing to the leg. In this master thesis, an assembling procedure developed by Aker Solutions is considered. Essentially, the assembly procedure

consists of assembling two components, the stub and the leg. The reader is referred to [chapter 2](#), *Stub Assembly for Jacket Manufacturing*, for a more detailed description and further illustrations of this procedure.

## 1.2. Problem description and objectives

Since 1975, Aker Solutions Verdal has provided steel jacket substructures for offshore oil and gas platforms. The yard in Verdal has a strong position in the market. Nevertheless, the global steel-price impact and the high salaries in Norway contribute to the yard being less competitive and losing contracts to foreign yards. Automation in the production would lead to a more modern yard with faster production, higher accuracy, and less staffing need to ensure more contracts in the future.

Sintef Manufacturing has, in collaboration with Aker Solutions Verdal initiated a project, AutoKons. The project address challenges in the manufacturing procedure of jackets, which are often one-piece productions. Such one-piece jackets consist of large, heavy components that face challenges within geometric accuracy and time-consuming tasks such as cutting, lifting, marking, and welding.

Assembling the stub onto the leg is today a central part of the jacket fabrication. However, fitting tubular joints during the prefabrication of steel substructures for offshore structures can present difficulties for the sheet metal worker/welder, as it does not always fit properly. The stub's cross-section cut is circular, ideally defined without considering any tolerances within ovality, straightness, and circularity. Such imperfections lead to the stub section not always fitting, causing time-consuming problems for the sheet metal worker/welder as adaptation and rework occurs.

The installation involves repeatedly lifting with a traveling overhead crane to mark for manual operations such as grinding and welding. Lifting operations are slow, which makes the procedure inefficient and costly.

Therefore, the AutoKons project researches applications within vision systems that can solve today's procedure in a more time-efficient way. Regarding time-efficient, so does both accuracy and manual operations make an impact.

In collaboration with SINTEF Manufacturing, this master thesis investigates creating a proof of concept of a 3D-camera-projector system that can project a stub section cut onto the leg surface to reduce the number of lifting operations. The proof of concept contains a structured light Zivid Two camera with an external projector in a stereo setup as in [Figure 1.1](#).

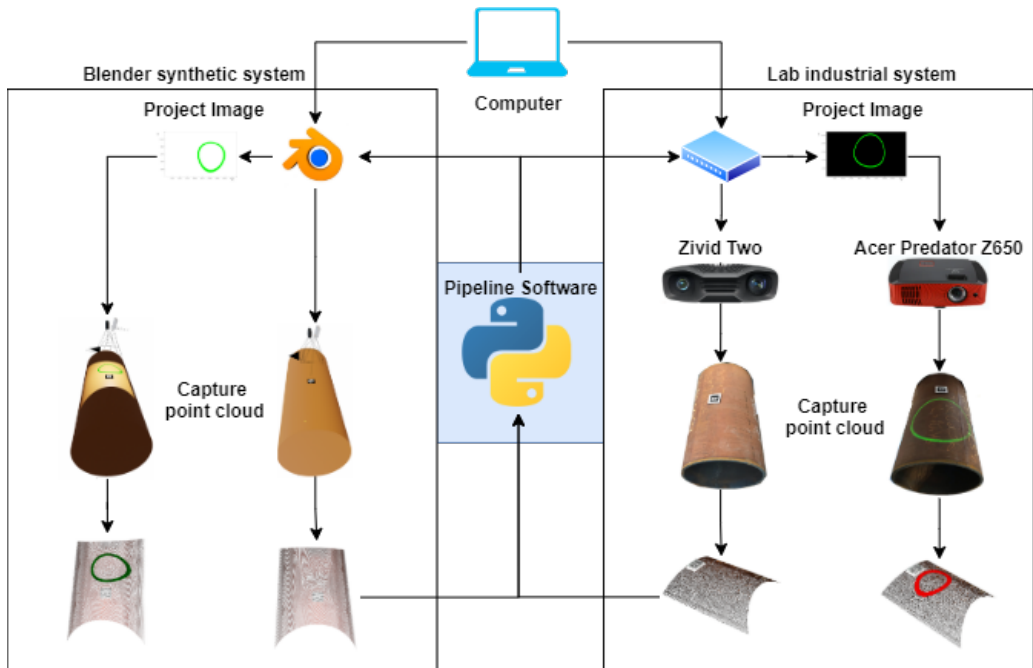


Figure 1.1.: System Overview

This master thesis' primary objective is to develop a method for projecting a stub cross-section cut using a 3D-camera-projector system. The secondary objectives are stated as:

1. Investigate the implemented method's further possibilities into today's assembling procedure.
2. Evaluate the developed method against requirements in *NORSOK-M101 Structural steel fabrication* [3].
3. Illuminate the method's limitations and identify further work for making the implemented method more robust.

## 1.3. Previous work

### 1.3.1. SINTEF Manufacturing

SINTEF has, before this master thesis was initiated, worked with a procedure for getting the intersection path between two measured meshes. The article, *Mesh-based tool path calculations for tubular joints* [4], presents a mesh intersection procedure to get the intersection path in the following sequences:

1. Performs an optional smoothing; interpolates the smoothed path to specified resolution.
2. Estimates the two surface normal vectors and the two surface tangents in the plane spanned by the normal's at each interpolation point.
3. Calculates the cutting tool and welding tool approach directions for obtaining the specified welding groove geometry at each interpolation point and stores all the data parameterized by the interpolation angle.

The result from the article present illustrations with both synthetic, representative meshes and meshes obtained from a hand 3D-scanner of actual tubes from the shop-floor manufacturer. The reference implementation for the developed software tool is based on Python and uses the mesh modeller from the 3D creation suite Blender as the platform.

### 1.3.2. Specialization project

In the author's specialization project [1], a flexible 3D-camera-projector method was implemented. This method allowed calibration of a 3D camera with a projector without printing of a conventional checkerboard. Instead, the checkerboard was projected through the projector, and the 3D camera defines the 3D points. This makes it easier to calibrate fast with different fields of view where you usually need printed checkerboards in various sizes. The method presented an acceptable result in form of a proof of concept. A further description of this method is presented in [Appendix B](#). The same technique is used in the implemented pipeline in this master thesis.



## 1.4. Limitations

Rather than producing an implemented method that tests every sub-process through the pipeline, this master thesis aims to establish a functional approach creating a good starting point. For making the method more robust, it is advantageous to test different methods/algorithms in each sub-process. The main focus has been to provide a pipeline with open-source libraries and few dependencies, thus all theories for all algorithms are not presented.

SINTEF Manufacturing already had the Zivid Two camera. Therefore, structured light is used as the geometric measuring method, hence obligating all other details on the proof of concept to conform to it. In other words, different measurement methods are not investigated in this report. The KUKA robot used in the lab setup is only used to move the 3D camera and projector around in the scene, and therefore no theory or detail on the KUKA robot is provided.

## 1.5. Outline

The report is structured around the implemented method presented in [chapter 4](#), of the cause so the method can be used for different hardware that are used in this thesis.

### **chapter 1 - Introduction**

Necessary background information, problem description, objectives, and limitations of the thesis are presented.

### **chapter 2 - Stub Assembly for Jacket Manufacturing**

The main parts and components of Aker Solutions' current assembly procedure are presented concurrently with a description of the procedure, followed by general acceptance criteria.

### **chapter 3 - Preliminaries**

The necessary initial theory within computer vision, point clouds and meshes.

### **chapter 4 - Pipeline**

This chapter describes the common pipeline, for either a synthetic or an industrial lab system with the connected dependencies. The pipeline is divided into four sub-pipelines that describe the processes for creating an Augmented Reality (AR) assembly instruction system, and for grinding the section-cut for tubular joints. The evaluation method of the pipeline concludes the chapter.

### **chapter 5 - Pipeline**

Description of the hardware used in the experiment followed by a walk-through of the steps included in the calibration method.

### **chapter 6 - Experiment**

An experiment is conducted to evaluate some of the processes in the implemented method. The goal and method is presented followed by the results.

### **chapter 7 - Discussion**

Evaluation of the implemented method followed by an interpretation of the results.

### **chapter 8 - Conclusion**

The conclusion of the master thesis is given followed by further work.

### **Appendix A - Appendix**

The appendix presents the tolerance requirements in [3], the 3d-camera-projector calibration method and the hardware data sheets.

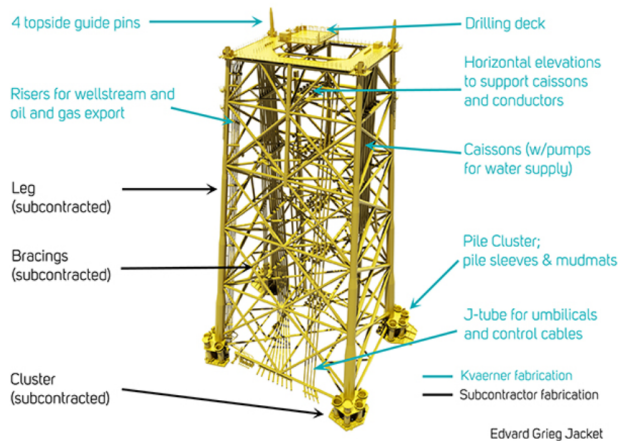
## Chapter 2.

# Stub assembly for jacket Fabrication

This chapter presents the today considered assembly procedure; assembly of tubular joints during jacket fabrication. The main parts and components are given concurrently with a description of the assembly procedure. The presented information within the assembly procedure is from chapter 2 in [5].

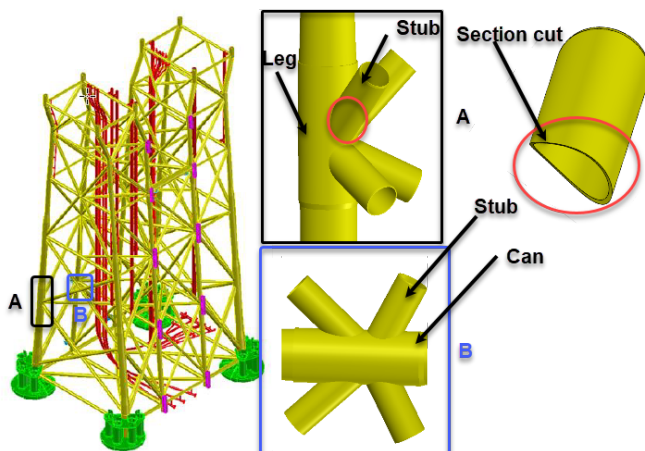
Some of the content in this chapter is based on the author's specialization project written as part of TPK4560 at NTNU [1].

### 2.1. Preliminary



**Figure 2.1.:** The various parts of a jacket explained [6]

The Yard at Verdal specializes in constructing steel substructures for offshore installations, so-called jackets, and has delivered multiple jackets since 1975. Steel jacket substructures are truss platforms used in the petroleum and wind turbine industries that are permanently attached to the seabed for prolonged use [7]. The platform is composed of a deck supported by the jacket attached to the seabed with piles or suction anchors.



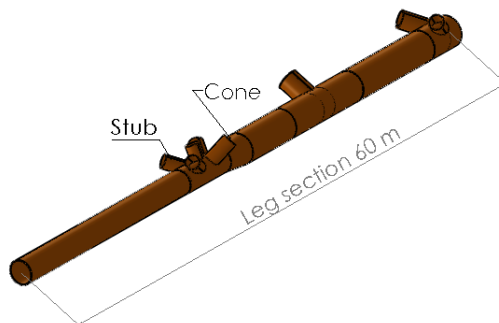
**Figure 2.2.:** Jacket with detail [8]

**View A:** Node of a stub mounted on leg with detail of section cut.

**View B:** Stub mounted on a can where to bracings intersect.

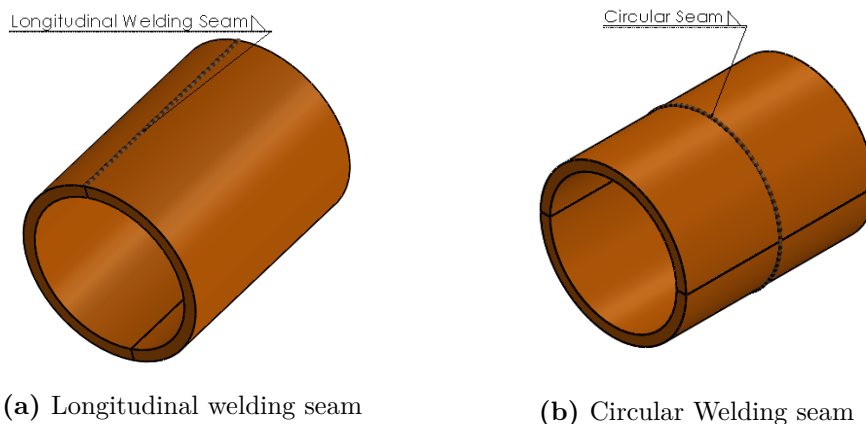
The jacket is a structure build of round tubes and have typically four or more legs connected by a series of bracings. When assembling the bracing onto the leg, a stub is used to connect the bracing to the leg. The stub is typically a 2 meter long round tube either mounted on to a leg (View A in Figure 2.2) or a can (View B in Figure 2.2) between two bracings. Instead of mounting the bracing direct onto the leg, a stub in each node makes it easier to align and adjust the bracing in the right position and angle.

## 2.2. Fabrication and dimensions



**Figure 2.3.:** Leg Section

Fabrication of large round tubes are done from cold-rolled plates and are welded together with [longitudinal welding seams](#) as in [Figure 2.4a](#). The length of the tube is dependent on the rolling machine. Thus, it is needed to connect the round tubes with [circular welding seams](#) as in [Figure 2.4b](#). The legs are split into sections up to 60 meters and finished with longitudinal- and round welding seams, cones, and cans before the stub assembly procedure starts. One leg section typically has only one cone [Figure 2.3](#).



(a) Longitudinal welding seam

(b) Circular Welding seam

**Figure 2.4.:** Welding Seams

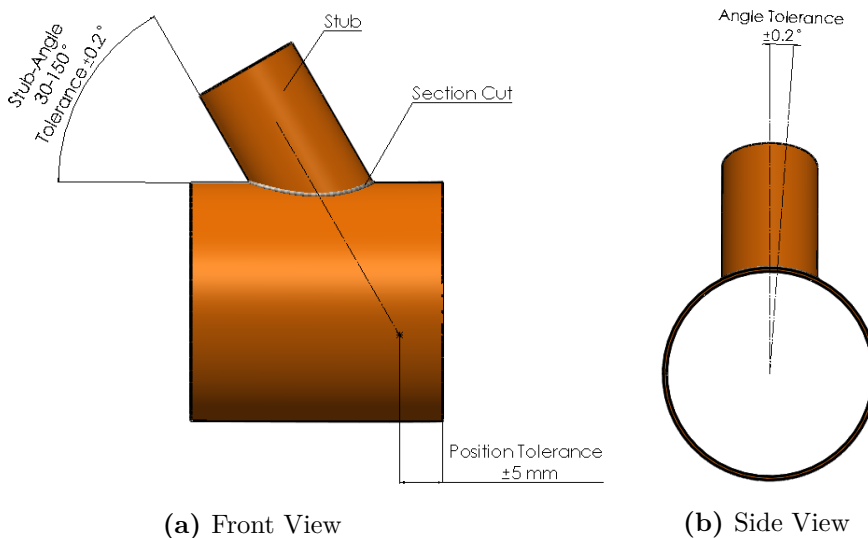
The legs diameter varies in size from 2-6 meters depending on the steel structure. Generally, the stub diameter is half of the leg; however, many stubs have the same diameter as the tube it is should be on. This applies mainly to stubs between bracings, legs without clusters, and the leg approach at the top of the jacket.

### 2.2.1. Tolerances

As mentioned, the geometric and measurement fabrications tolerances for tubular [section A.1](#)) and tubular nodes [section A.2](#) impact today a lot how efficient the stub installation turn out. High deviations like this can lead to time-consuming work and, in the worst case, rework for the sheet metal worker and welder. The tolerances that are essential for the stub assembly manufacturing is according to NORSOK-M-101 [3] standard. The three most important are:

1. Circumference 1
2. Out of roundness(ovality) 2
3. Straightness 4

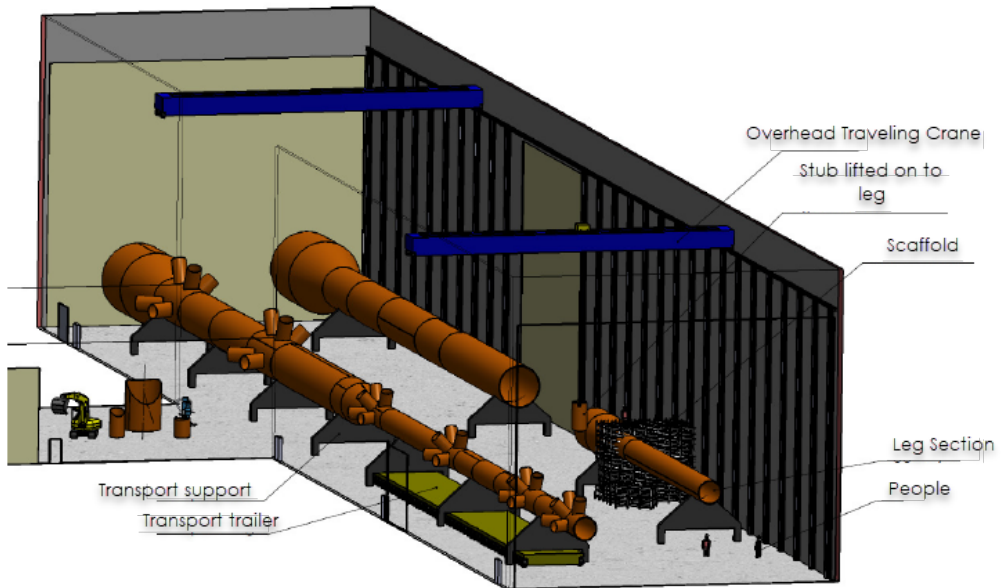
The reader is referred to [Appendix A](#) in the appendix for a more detailed explanation.



**Figure 2.5.:** Tolerances

The assembly must be accurate, so the tubular fits in the desired position later in the installation. All preparation work is so the welder could make a good weld. A sharp, even welding groove improves the conditions for a good weld. Factors such as visibility, access, and working position are also crucial. If a welding defect is detected through the inspection, the weld around the area with the defect must be grouted away and welded again.

## 2.3. Stub assembly procedure



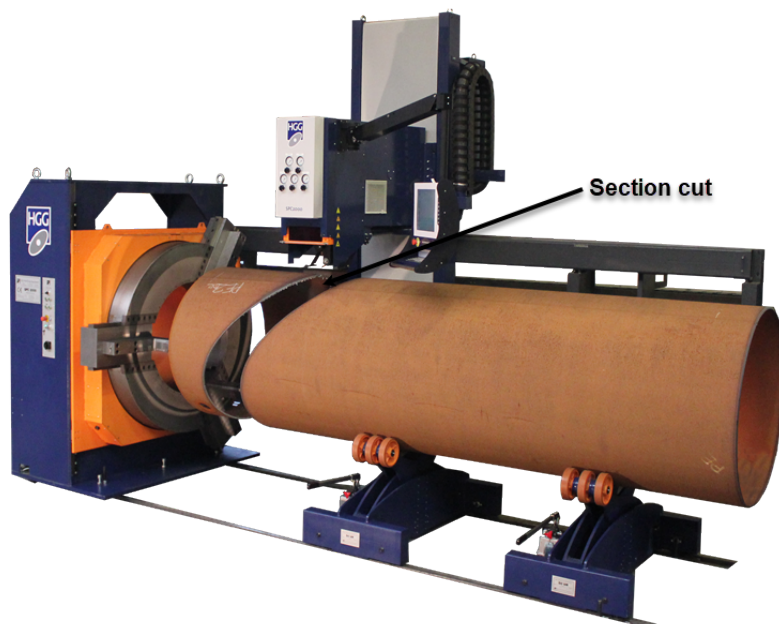
**Figure 2.6.:** Fabrication area for stub assembly [5]

The stub assembly procedure can be summarized into the following steps:

1. Cut the cross-section cut of the stub.
2. The legs are leveled and rotated, so the stub is mounted on the leg's highest point.
3. Lifting the stub onto the leg with overhead travelling crane. Marking where to grind away embers and welding seams.
4. Lifting the stub down and start grinding.
5. Installation of the stub on wedges and welding.

### 2.3.1. Stub cross-section cut

A CNC plasma cutter, [Figure 2.7](#), defines the cutting path for the stub section-cut. The stub and the leg's nominal diameter define the path, and all geometry deviations would impact the section cut accuracy. The cut accuracy depends on the tube's geometry within circularity, ovality, and straightness. Also, by applying the cut, much heat is added that can change the tube geometry. After the cut, embers, longitudinal seams, and circular seams need to be removed in the area where the stub should be welded to the leg.

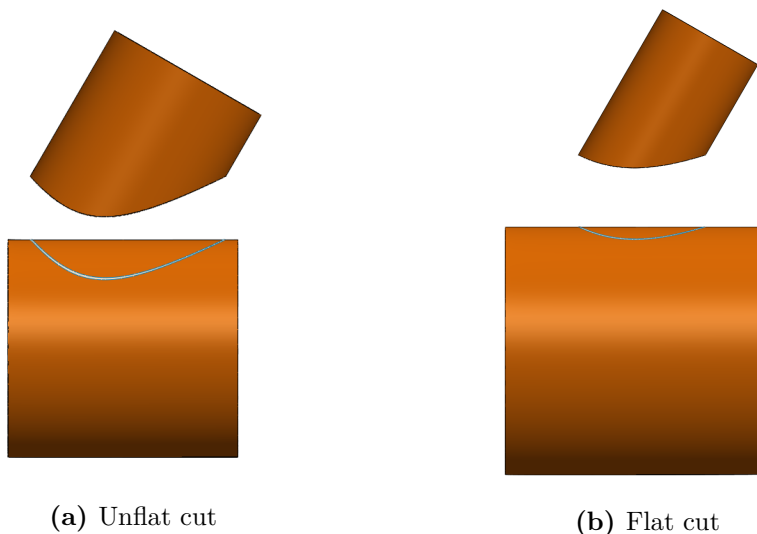


**Figure 2.7.:** SPC 1500-3000 PT, CNC Pipe Cutting machine for vessels and offshore [9]

Even if the manufactured parts do not entirely match the drawing, the result must be accurate to stay within the tolerances, especially for opposing construction components. Mainly so that the forces are distributed as in the done calculations. If the installation is inaccurate, it can give many extra hours filling with welding and a greater probability of welding errors. Welding of the stub is the most time-consuming part and takes almost twice the stub assembly time. Typical values are a maximum of 15 hours for the assembly and welding at least 35 hours. The angle, length, and thickness of the stub vary depending on the jacket's placement. The welding method depends on the angle, type of welding groove, and where the operator is welding on the section cut.



The larger the difference between the diameter, the easier it is to assemble the stub, as the section-cut becomes a more flatted cut, showed in [Figure 2.8](#). All the round tubes are thick-walled and robust, and difficult to modify if they are out of roundness. Components with out of roundness greater than the required tolerance in [Appendix A](#) are often replaced because of vital dimensions.

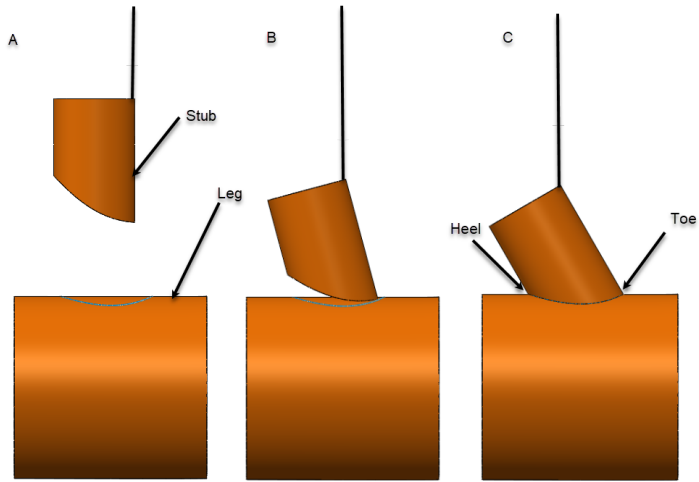


**Figure 2.8.:** Section Cuts

### 2.3.2. Lifting up and marking

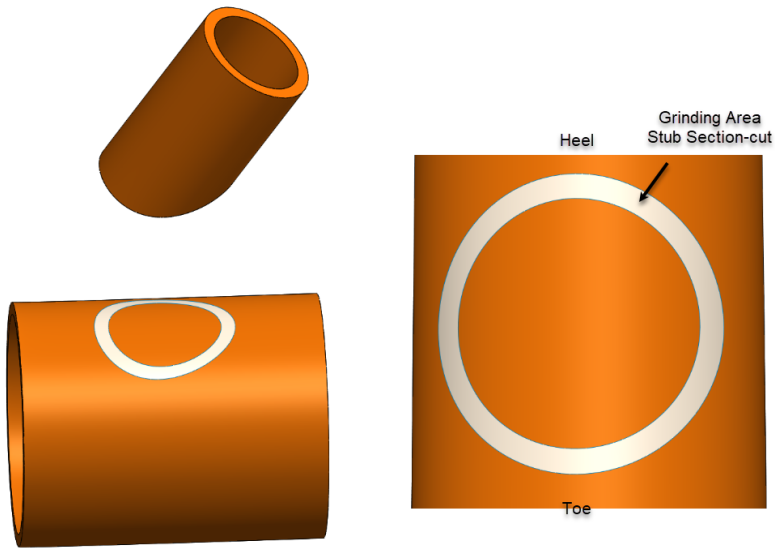
Before the stub assembly begins, the legs are leveled and rotated, so the stub is mounted on the leg's highest point. The legs' rotation is often done with two overhead traveling cranes simultaneously and is a time-consuming job.

The stub is lifted with a traveling overhead crane and lowered onto the leg. In the [Figure 2.9](#), the stub hangs a little crooked due to the center of gravity. The cut of the stub makes it land approximately at the correct angle.



**Figure 2.9.:** Overhead traveling crane lift the stub onto the leg

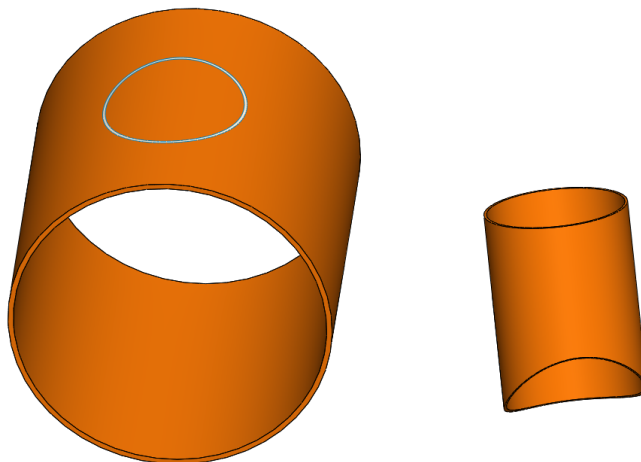
When the stub is in approximate position, the stub's section is marked on the leg to see the area where one needs to grind away surface rust and embers.



**Figure 2.10.:** Marked Grinding zone for stub section-cut

### 2.3.3. Lifting down and grinding

The stub is lifted down again to be able to grind. Then the stub is lifted one more time for assembly. This time the stub is landed on wedges to adjust the joint opening and fine-tune the angle. A sledgehammer is used to adjust, and the stub is offloaded on only two wedges and hanging in the crane. The stub would then be difficult to handle and unstable.



**Figure 2.11.:** Lifted down for grinding

### 2.3.4. Installation and welding

The stub is in position, and the toe can be spot welded when the position and opening in the toe are correct. This helps a lot with the stability of the rest of the installation. Small pieces are used under welding for extra control of the welding. These need to be removed afterward. The pieces may not be preheated before welded because it is unwieldy and slowly, and the material around can be damaged somewhat by this. This could lead to problems.

The time spent assembling is associated with a good deal of coincidence when one gets the stub in an accurate position but is relatively easy for the most experienced sheet metal workers. One does not know entirely if the cuts fit ( for example, if it has been burned correctly) before the installation is almost finished, so that all work may prove in vain.

During the entire installation, the overhead traveling crane will secure the stub and limit other manufacturing areas' activity. Other overhead traveling cranes can not pass the one in use. If something is to be lifted past, one must wait or use

other transport methods. The number of stubs installed simultaneously for a day depends on the number of cranes and the number of stubs pointing in the same direction( number of stubs on the top of the leg when rotated). Typical is three stubs of 6 sheet metal workers. The stubs are welded before a new installation because you do not have to rotate the leg many times.

# Chapter 3.

## Preliminaries

This chapter provides the reader with necessary theoretical knowledge regarding computer vision system in 3D space. The presented theory concepts is gathered from several textbooks as well as papers published in widely accepted journals [10]–[12].

Some of the content in this chapter is based on the author’s specialization project written as part of TPK4560 at NTNU [1].

### 3.1. Computer vision

#### 3.1.1. Rotation matrices

Rotation matrices are used to represent the orientation difference of a coordinate system  $\{c\}$  to a rotated coordinate system  $\{o\}$ . The coordinate frames are a representation in World coordinates(3D) and represented as a  $3 \times 3$  matrices with each column being equal to a unit vector. The reference system axes can be e.g be represented as the identity columns vectors in  $\mathbb{R}^{3 \times 3}$ .

$$\{c\} = \begin{bmatrix} x_c & y_c & z_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

To obtain the frame  $\{o\}$ , a linear transformation  $R_{co}$  is applied to represent the rotation from frame  $\{c\}$  to frame  $\{o\}$ , denoted as  $R_c$

$$R_c = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (3.2)$$

Each matrix column representing coordinate the frame  $\{o\}$  are the directional unit vectors of the the axes of frame  $\{o\}$  given in the coordinates of frame  $\{c\}$ .  $SO(3)$  has the following definition [10]:

$$R \in SO(3) = \left\{ R \in \mathbb{R}^{3 \times 3}, R^T R = I^{3 \times 3}, \det(R) = 1 \right\} \quad (3.3)$$

Thus we can denote,

$$R^T = R^{-1} \quad (3.4)$$

### 3.1.2. Homogeneous transformation matrices

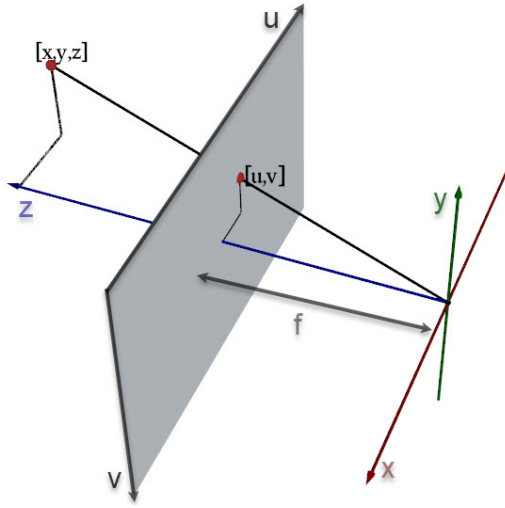
Transformation matrices are representations for combining orientation and position of a rigid body. Normally, a natural choice will be to use a rotation matrix as in (3.2) to represent a rotation of a body frame  $\{b\}$  in the fixed frame  $\{s\}$  and a vector  $t \in \mathbb{R}^3$  to represent the origin of  $\{b\}$  in  $\{s\}$ . Rather than identifying  $R$  and  $t$  separately, we add them both into a single matrix as follows [10].

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

where  $R \in SO(3)$  and  $t \in \mathbb{R}^3$  is a column vector.

### 3.1.3. Pinhole camera model

The pinhole camera model is commonly used in modeling a real camera for computer vision applications. The model is designed so the light is supposed to go through the optical center, which is the origin of the camera frame  $c$ , and then be projected on to the retinal plane where the image sensors are located [12]. The retinal plane is parallel to the focal plane and located the negative focal length  $f$  distance in the  $z_c$  direction. A simplification done in agreement with the thin-lens theory of optics introduces the virtual image plane as the plane in the front of the camera frame, parallel to the focal plane, and located the focal length  $f$  in the positive  $z_c$  direction.



**Figure 3.1.:** Pinhole camera model

Let the homogeneous coordinates in the camera frame  $c$  be represented as:

$$\tilde{r}_{cp}^c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \quad (3.6)$$

The same point in the world coordinate frame  $w$  be defines as:

$$\tilde{r}_{wp}^w = \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (3.7)$$

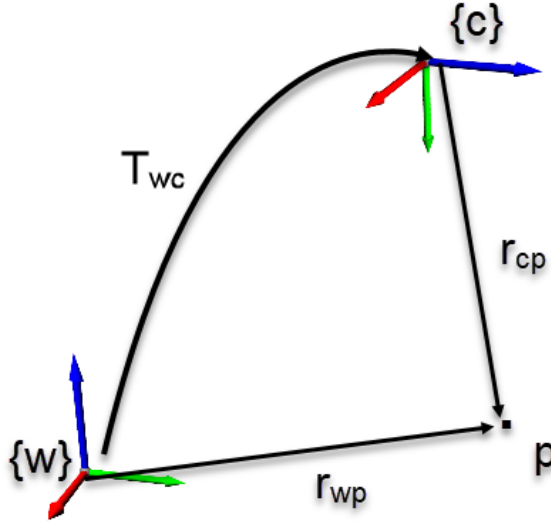
To be denoted in the figure below:

The transformation between frames for this point is given by

$$\tilde{r}_{cp}^c = T_{cw} \tilde{r}_{wp}^w \quad (3.8)$$

where  $T_{cw}$  is homogeneous transformation matrix from  $\{c\}$  to  $\{w\}$ .

The normalized coordinates of the point is represented by  $r_{cp}^c$  projected onto the



**Figure 3.2.:** Camera and world coordinate systems

image plane is given by:

$$\tilde{s} = \begin{bmatrix} s_x \\ s_y \\ 1 \end{bmatrix} = \frac{1}{z_c} r_{cp}^c = \frac{1}{z_c} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} \frac{x_c}{z_c} \\ \frac{y_c}{z_c} \\ 1 \end{bmatrix} \quad (3.9)$$

The conversion between homogeneous and euclidean vector coordinates can be done by

$$r = \Pi \tilde{r} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \quad (3.10)$$

The normalized image coordinates can now be calculated by

$$\tilde{s} = \frac{1}{z_c} \Pi_{cp}^{cc} = \frac{1}{z_c} \Pi T_{cw} \hat{r}_{up}^w \quad (3.11)$$

The corresponding pixel coordinates values can be calculated by doing the following

$$u = \frac{f}{\rho_w} s_x + u_0 \quad (3.12)$$



$$v = \frac{f}{\rho_h} s_x + v_0 \quad (3.13)$$

Where  $f$  is the focal length,  $\rho_w$  is the horizontal width off a pixel,  $\rho_h$ , is the vertical height of a pixel,  $u_0$  and  $v_0$  are the pixel coordinates of the  $z_c$  axis, which is the center of the image plane [12]. The pixel coordinates is defines so the point of origin is in the upper left corner of the image plane. The pixel coordinates vector form are given the form as

$$p = \begin{bmatrix} u \\ v \end{bmatrix}, \quad \tilde{p} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (3.14)$$

Now it is possible to describe the transformation from the normalized image coordinates to the pixel coordinates as a linear transformation by using the homogeneous vectors  $\tilde{\mathbf{p}}$  and  $\tilde{\mathbf{s}}$ . The transformation is expressed as:

$$\tilde{p} = K\tilde{s} \quad (3.15)$$

where  $\tilde{p} = (u, v, 1)^T$  is the pixel coordinate vector  $\tilde{s}$ ,  $\tilde{s} = (s_x, s_y, 1)^T$  is the normalized image vector and

$$K = \begin{bmatrix} \frac{f}{\rho_w} & 0 & u_0 \\ 0 & \frac{f}{\rho_h} & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.16)$$

is the camera parameter matrix. The elements of  $\mathbf{K}$  are the intrinsic  $f$ ,  $\rho_h$ ,  $\rho_w$ ,  $u_0$  and  $v_0$  that are specific to the camera. The equations (3.13) and (3.16) can be combined into represent the projective camera transformation as

$$z_c \tilde{p} = K \Pi T_{cw} \tilde{r}_{wp}^w \quad (3.17)$$

This representation can be used to calculate the pixel coordinates for a corresponding to a point  $\tilde{r}_{wp}^w$ , given the relative transformation between  $\{c\}$  and  $\{w\}$ . This formulation is the is the basis for solving the inverse of a problem where the pixel values of a point is known, and the position in the coordinates of the fixed world frame  $\{w\}$  is desired.

### 3.1.4. Corner detection

In computer vision systems detection of corners in variety of angles, light and orientation is important within applications such as object detection, pose estimation and camera calibration.

Corner detection is a approach used to find the regions in an image where is a sharp change in intensity or a sharp change color as e.q. a corner. A corner can be defined as the intersection between two edges. The sobel operator convolved in use with the original image is used to calculate the approximations of derivatas, representing the gradients of the image in x- and y-direction as in (3.18) and (3.19) [13]. This operator is used in a variety of corner detection algorithm such Harris [14], curvature scale-space (CSS) [15], or Susan [16].

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad (3.18)$$

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3.19)$$

### 3.1.5. Camera calibration

The camera calibration aims to establish the geometric parameters such as intrinsic and extrinsic parameters. This is a crucial step in many computer vision applications where metric information in the scene is required. The intrinsic parameters obtained from a calibration are the parameters in (3.16) The pin hole model in subsection 3.1.3 is simplified due to the sense that the light is going straight through the optical center and hits the image sensor. In a real camera, lenses focus the light in a way that it doesn't behave as in the pin hole model and need to be corrected by a set of distortion coefficients. Two major distortion are radial distortion and tangential distortion. Due to radial distortion, straight lines will appear curved. This effects is more visible as we move from the center of the image. Tangential distortion occurs because the image taking lens is not aligned perfectly parallel to the imaging plane. So some areas in image may look nearer than expected. In short to correct the image five distortion coefficients is given by:

$$\text{Distortion coefficients} = \left( k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3 \right) \quad (3.20)$$

Where the  $k_1$ ,  $k_2$  and  $k_3$  correct for the radial distortion, and  $p_1$  and  $p_2$  for the tangential distortion [17].

The extrinsic parameters corresponds to rotation and translation vectors which translates a coordinates of a world point to a coordinate system.

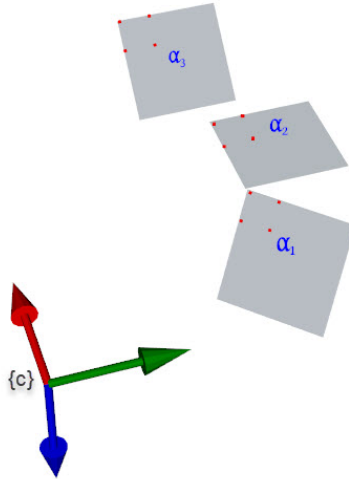
### 3.1.6. OpenCV

In OpenCV the integrated method for *Calibratecamera* is from the Zhang's and Bouget's Method. The Zhang's [18] method is the most used calibration technique. The technique only requires the camera to observe a planar pattern from a few (at least two) different orientations without knowing the distance. OpenCV uses a printed checkerboard with a set size of the squares and number of squares in the row- column direction(as in the figure). To calibrate the camera, a set of 3D points and the corresponding camera 2D image coordinates is needed. The inner corners points in the checkerboard is defined as the 3D points(objectpoints) and the correspondent camera image coordinates are found by corner detection using the function *foundchessboardcorners*(imagepoints). Since the checkerboard is required to be attaced to a plane(wall) , the Z-coordinate of the objectpoints is set equal to zero. When comparing the set of objectpoints and imagepoints the camera matrix, the distortion coeffisents, rodriguez vectors, and translation vectors can be obtained.

### 3.1.7. Zhang's method

This section presents Zhang's method for calibration [18] where the camera parameter matrix,  $\mathbf{K}$  is computed from Pnp in three planes where the unit orthogonal vectors of at least three checkerboard plane poses are found, and the absolute conic  $B = K^{-T}K^{-1}$  is found from the 2 constraints for each plane related to these orthogonal unit vectors [12].

Consider 3 object planes  $\alpha_1$ ,  $\alpha_2$  and  $\alpha_3$ . Define an object frame 1 so that the  $z$  coordinate is zero in the object plane  $\alpha_1$ ,  $\alpha_2$  and  $\alpha_3$ . As in the [Figure 3.3](#) below:



**Figure 3.3.:** Three planes,  $\alpha_1, \alpha_2$  and  $\alpha_3$ , with four points in each plane.

In each object plane  $\alpha_j$  there are 4 points with coordinates in the object frame  $\{j\}$  given by  $\bar{r}_{ji}^j = [x_{ji}, y_{ji}, 0]^T$ . The corresponding homogeneous vector of the position in the  $xy$  plane of frame

$$\bar{\mathbf{x}}_{ji} = \begin{bmatrix} x_{ji} \\ y_{ji} \\ 1 \end{bmatrix} \quad (3.21)$$

The normalized image coordinates of the point  $\bar{r}_{ji}^j$  are given by

$$\bar{s}_{ji} = \mathbf{\Pi} \bar{r}_{ji}^j = \begin{bmatrix} \mathbf{R}_j & \mathbf{t}_j \end{bmatrix} \bar{r}_{ji}^j = \mathbf{M}_j \bar{\mathbf{x}}_{ji} \quad (3.22)$$

where,

$$\mathbf{M}_j = \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \quad (3.23)$$

The pixel coordinates are then

$$\bar{p}_{ji} = \mathbf{K} \bar{s}_{ji} = \mathbf{K} \mathbf{M}_j \bar{\mathbf{x}}_{ji} \quad (3.24)$$

This defines a homography  $\mathbf{H}_j = \mathbf{K} \mathbf{M}_j$  for the plane  $\alpha_j$  so that

$$\bar{p}_{ji} = \mathbf{H}_j \bar{\mathbf{x}}_{ji} \quad (3.25)$$

The homography  $\mathbf{H}_j = \mathbf{K} \mathbf{M}_j$  for the plane  $\alpha_j$  can be found from the four point mappings  $(\bar{p}_{ji}, \bar{\mathbf{x}}_{ji})$  where  $\bar{\mathbf{x}}_{ji}$  is a point on the plane  $\alpha_j$ . Suppose that the homog-

raphy  $\mathbf{H}_j = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}$  is found for 3 different planes  $\alpha_j, j = 1, 2, 3$ . Then for each plane

$$\begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \end{bmatrix} = \begin{bmatrix} \mathbf{K}\mathbf{r}_1 & \mathbf{K}\mathbf{r}_2 & \mathbf{K}\mathbf{t} \end{bmatrix} \quad (3.26)$$

and follows that

$$\mathbf{r}_1 = \mathbf{K}^{-1}\mathbf{h}_1 \quad \text{and} \quad \mathbf{r}_2 = \mathbf{K}^{-1}\mathbf{h}_2 \quad (3.27)$$

The orthogonality of the rotation matrix implies that

$$\begin{aligned} \mathbf{r}_1^T \mathbf{r}_2 &= \mathbf{0} \\ \mathbf{r}_1^T \mathbf{r}_1 &= \mathbf{r}_2^T \mathbf{r}_2 \end{aligned} \quad (3.28)$$

This gives the conditions

$$\begin{aligned} \mathbf{h}_1^T \mathbf{B} \mathbf{h}_2 &= 0 \\ \mathbf{h}_1^T \mathbf{B} \mathbf{h}_1 &= \mathbf{h}_2^T \mathbf{B} \mathbf{h}_2 \end{aligned} \quad (3.29)$$

on the image of the absolute conic  $\mathbf{B} = \mathbf{K}^{-T} \mathbf{K}^{-1} = (\mathbf{K} \mathbf{K}^T)^{-1}$ , which is a symmetric matrix. Ensuring the symmetry of  $B$  the elements of the matrix is written in terms of 6 independent variables  $b_1, \dots, b_6$  as

$$\mathbf{B} = \begin{bmatrix} b_1 & b_2 & b_4 \\ b_2 & b_3 & b_5 \\ b_4 & b_5 & b_6 \end{bmatrix} \quad (3.30)$$

The elements are stacked in the vector as

$$\mathbf{b} = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{bmatrix}^T \quad (3.31)$$

The two conditions for each plane can then be reformulated as  $a_1 \mathbf{b} = 0$  and  $a_2 \mathbf{b} = 0$ . The first condition can be formulates as

$$\begin{aligned} \mathbf{u}^T \mathbf{B} \mathbf{v} &= u_1 v_1 b_1 + (u_1 v_2 + u_2 v_1) b_2 + u_2 v_2 b_3 \\ &+ (u_1 v_3 + u_3 v_1) b_4 + (u_2 v_3 + u_3 v_2) b_5 + u_3 v_3 b_6 \end{aligned} \quad (3.32)$$

and the second is

$$\begin{aligned} \mathbf{u}^T \mathbf{B} \mathbf{u} - \mathbf{v}^T \mathbf{B} \mathbf{v} &= (u_1^2 - v_1^2) b_1 + 2(u_1 u_2 - v_1 v_2) b_2 + (u_2^2 - v_2^2) b_3 \\ &+ 2(u_1 u_3 - v_1 v_3) b_4 + 2(u_2 u_3 - v_2 v_3) b_5 + (u_3^2 - v_3^2) b_6 \end{aligned} \quad (3.33)$$

where  $\mathbf{h}_1 = \mathbf{u} = [u_1, u_2, u_3]^T$  and  $\mathbf{h}_2 = \mathbf{v} = [v_1, v_2, v_3]^T$  is used to simplify

notation. This gives

$$\begin{aligned} \mathbf{a}_1 &= \begin{bmatrix} u_1v_1 & u_1v_2 + u_2v_1 & u_2v_2 & u_1v_3 + u_3v_1 & u_2v_3 + u_3v_2 & u_3v_3 \end{bmatrix} \\ \mathbf{a}_2 &= \begin{bmatrix} u_1^2 - v_1^2 & 2(u_1u_2 - v_1v_2) & u_2^2 - v_2^2 & 2(u_1u_3 - v_1v_3) & 2(u_2u_3 - v_2v_3) & u_3^2 - v_3^2 \end{bmatrix} \end{aligned}$$

There will be 2 conditions, and 3 planes gives 6 conditions. This leads to the expression

$$Ab = 0 \quad (3.34)$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_{11} \\ \mathbf{a}_{12} \\ \mathbf{a}_{21} \\ \mathbf{a}_{22} \\ \mathbf{a}_{31} \\ \mathbf{a}_{32} \end{bmatrix} \quad (3.35)$$

A solution for  $\mathbf{b}$  is then found with a singular value decomposition of  $\mathbf{A}$  as

$$\mathbf{A} = \sum_{i=1}^6 \sigma_i \mathbf{u}_i \mathbf{v}_i^T, \quad \mathbf{u}_i \in \mathbb{R}^{2n}, \mathbf{v}_i \in \mathbb{R}^6 \quad (3.36)$$

which gives  $\mathbf{b} = kv_6$ . Then the matrix  $\mathbf{B}$  is found from the elements of  $\mathbf{b}$ . Now, the camera matrix  $\mathbf{K}l$  can be found from  $\mathbf{B}$  using Cholesky decomposition. There are different variants of the Cholesky decomposition. The Cholesky decomposition of a symmetric positive definite matrix  $\mathbf{B}$  as  $\mathbf{B} = \mathbf{L}\mathbf{L}^T$  where  $\mathbf{L}$  is a lower triangular matrix, that is, a matrix where all the elements above the diagonal are zero. However, in the standard MATLAB and OpenCV function the Cholesky decomposition as  $\mathbf{B} = \mathbf{G}^T\mathbf{G}$  where  $\mathbf{G}$  is upper triangular, which is a matrix where all elements below the diagonal are zeros. This is appropriate for this problem because  $\mathbf{K}$  and  $\mathbf{K}^{-1}$  are upper triangular as in (3.16). Cholesky decomposition based on  $\mathbf{B} = \mathbf{G}^T\mathbf{G}$  where  $\mathbf{G}$  is upper triangular then gives  $\mathbf{K}^{-1}$ , and  $\mathbf{K}$  is finally found by matrix inversion.

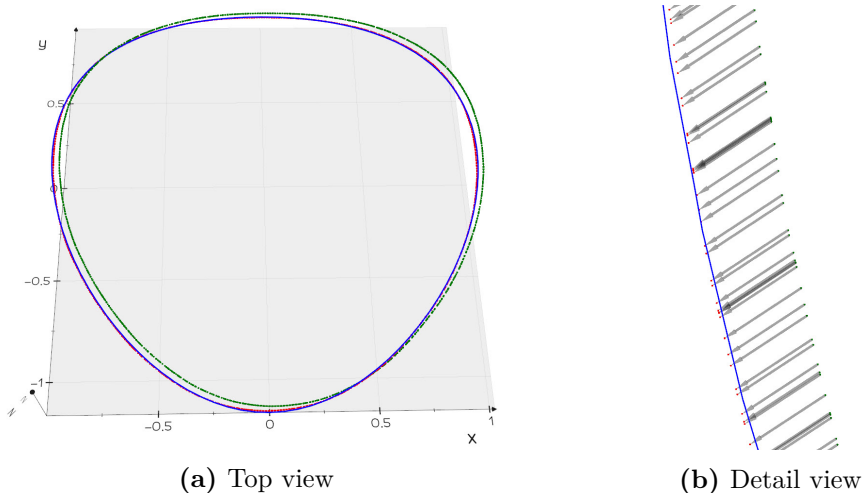
## 3.2. Point clouds

Point clouds are a set of data points in space which either can represent a 3D shape or an object. Each point can be represented as  $p_j = [x_j, y_j, z_j]^T$ . To generate a point cloud the accuracy of the depth value along the optical axis  $z_c$ , is crucial to calculate  $p_j$  by using (3.17) [12]. The depth value can be calculated using structured light, time of flight finders or stereo vision. A point cloud is often organized in combination with RGB values for each point. The point cloud

can be used in applications such as geometric measurement, visualization, object recognition and classification.

### 3.2.1. Iterate closest point

The iterates closest point(ICP) is an algorithm designed to minimize the difference between two point clouds by finding the best fit transformation in terms of rotation and translation. The algorithm is compromised by every iteration of ICP as a least



**Figure 3.4.:** Uses ICP to align the green point cloud to the blue. The red point cloud is after ICP.

squares minimization problem. The function that wants to be minimized is the squared sum of distances between a point cloud  $\mathbf{P} = \{p_i\}$ , to another point cloud  $\mathbf{Q} = \{q_i\}$ :

$$E = \sum_i [\mathbf{R}p_i + \mathbf{t} - q_i]^2 \rightarrow \min \quad (3.37)$$

To minimize this function the pose in form rotation  $\mathbf{R}$  and translation  $\mathbf{t}$  need to align the  $\mathbf{P}$  to  $\mathbf{Q}$ . It is a non-linear function because of the rotation.

### 3.2.2. Rotate point clouds using normals

The computation of surface normals from a point in a point cloud,  $p_j = [x_j, y_j]$ , is essential in many 3D vision applications. This can be done in different ways such as:

1. Pick three non-collinear points lying in a plane,  $p_j$ , to calculate the plane normal Z-axis, Y-axis and X-axis.

2. Use all the points in  $p_j$  to fit a plane by minimizing the least squares error, this will give us plane normal(Z-direction). Then choose two points in  $p_j$  to calculate the X or Y axis direction. Either by the cross product of the normal and X or y, i.e.  $y = x \times z$
3. Use the eigenvectors of  $p_j$  covariance matrix as the plane's xyz axes.

From these three methods there are some pros and cons. In method 1, which three points should we use to estimate the plane? The same question is relevant for method 2, which two points should we use to estimate the X and Y axis? The preferred method is 3 since there we use all the variances of all the points to calculate the xyz axis for the plane [19].

### 3.2.3. Covariance matrix

By looking into the covariance matrix, a quick look at the difference between the variance and the covariance will make things more clear. Variance measures the variation of a single variable (the values along the x-axis example), whereas covariance for a point cloud is a measure of how much three random variables vary together (like the values along the X, Y and Z axis) [20]. The covariance matrix for a 3-Dimensional case can be expressed as

$$C = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X}) (X_i - \bar{X})^T \quad (3.38)$$

where  $C \in \mathbb{R}^{3 \times 3}$ ,  $X_i = [x_i, y_i, z_i]$  and  $\bar{X}$  is the sample mean for all axes and can be expressed as the centroid of the coordinates.  $X \in \mathbb{R}^{n \times 3}$  where  $n$  is the number of points in the point cloud. An interesting approach is that the expression

$$X_i - \bar{X} = C_n \cdot X_i \quad (3.39)$$

where  $C_n$  is the centering matrix [21], see in [section D.1](#) for a closer look. This expression (3.39) defines the set of coordinates relative to the origin set as the centroid. Then we can express the covariance matrix (3.38) as

$$C = \frac{1}{n-1} \sum_{i=1}^n (C_n \cdot X_i) (C_n \cdot X_i)^T \quad (3.40)$$

To find the set of the points XYZ axis we need to find the eigenvectors of the covariance matrix. This can be applied either by eigen decomposition of the covariance matrix or singular value decomposition (SVD) of the expression in (3.39) as

$$(C_n \cdot X) = U \Sigma V^T \quad (3.41)$$



Where each column of  $U$  represent the XYZ axis of the  $X$  matrix. To orientate  $X$  the XY plane of the checkerboard coordinates we need to left multiply the eigenvector  $U^{-1} = U^T$  to the expression in (3.39) as

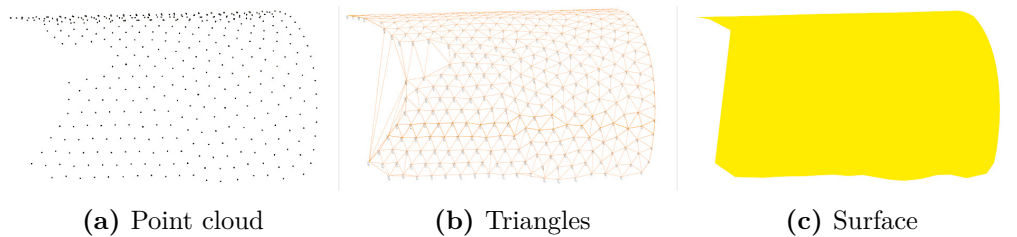
$$X_r = U^T \cdot (C_n \cdot X) \quad (3.42)$$

Where  $X_r$  is the vector where a set of points are aligned with XY plane given that  $X$  coordinates are collinear along the relative Z-axis for  $X$ .

### 3.3. Mesh

#### 3.3.1. Delauney triangulation

Delauney triangulation is used to construct topology from unstructured point data. In two dimensions we create generate triangles from an unstructured grid or polygonal dataset, while in three dimensions we generate a tetradal. In the [Figure 3.5](#) we see how to create triangles in 3D from a field of points.



**Figure 3.5.:** Delauney Triangulation

These points are captured by a 3D-camera and are down sampled since it isn't necessary to create triangles from all the points. Since all the points in [Figure 3.5a](#) can be projected onto a plane. The triangle indices are created in 2D, and then the same indices in 3D are found afterwards to generate the triangles. The Delauney triangulation meet the following properties [22]:

1. In Delauney there will be no other points(vertex) within the circumcircle of any triangle.
2. A triangle is formed by the nearest points, and each line segment do not intersect.
3. No matter where the area starts from, the final result will be consistent.
4. If the diagonals of the convex quadrilateral formed by any two adjacent triangles are interchangeable, then the smallest angle among the six internal

angles of the two triangles will not become larger.

5. If the smallest angle of each triangle in the triangulation is arranged in ascending order, the arrangement of the Delaunay triangulation will get the largest value.
6. Adding, deleting, or moving a vertex will only affect the adjacent triangle.
7. The outermost boundary of the triangular mesh forms a convex polygon shell.

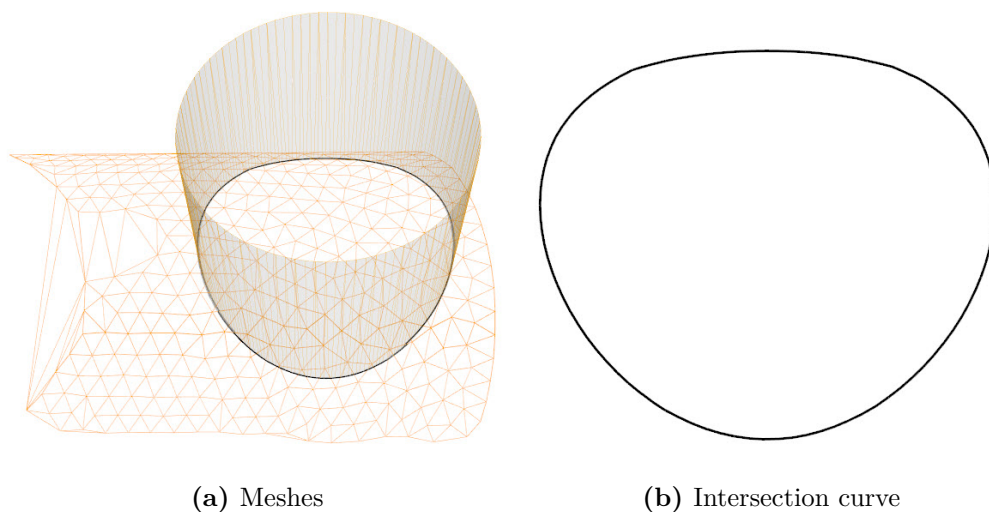
One important concern regarding the Delaunay triangulation is that the process is numerically sensitive. Creating triangles with too few points can cause the algorithm to break down. And if you have a large number of points to triangulate, you may consider to randomize the order of the points.

### 3.3.2. Mesh intersection

Mesh intersections within VTK [22] are within boolean operations and the algorithm in VTK is from [23]. The possible Boolean operations between two objects are their union ( $A \cup B$ ), intersection ( $A \cap B$ ), and difference ( $A - B$  or  $B - A$ ). A boolean procedure takes in two objects and outputs a combination of these objects. These objects are assumed to be triangulated surface meshes. To explain the boolean procedure, a couple definitions are required to be explained,

**Intersection loops:** Define where one surface crosses the other and **Sub-surfaces:** are the portions of each surface that are separated by the intersection loops. The computational steps in the Boolean process for discrete polygonal surfaces can be summarized into:

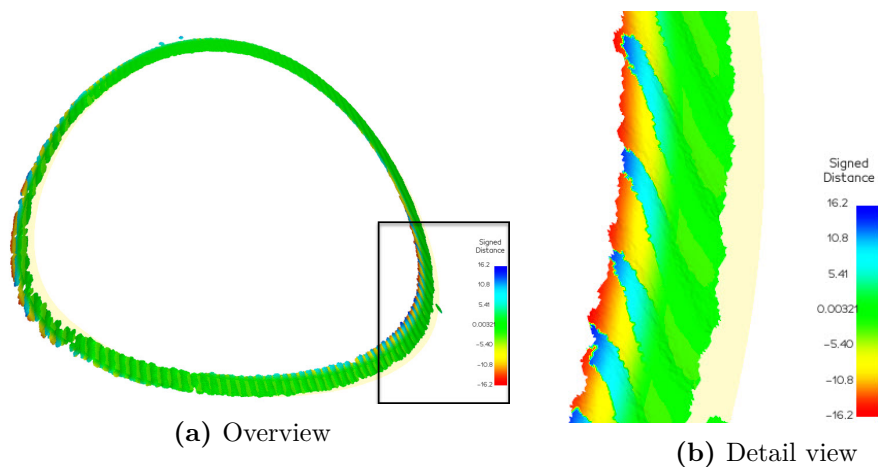
1. **Intersection:** Determine where the input surfaces intersect in space. This step creates the intersection loops that are used for re-triangulation and sub surface determination.
2. **Re-triangulation:** Re-triangulate each surface near the intersection loops. The intersection loops are comprised of intersection points and lines on each surface, and each surface is re-triangulated separately.
3. **Boolean:** Determine the correct combination of sub surfaces for output. The Sub-surfaces are extracted based on their orientation relative to the intersection loops incident on the surfaces.



**Figure 3.6.:** Mesh intersection between two coarse meshes

### 3.3.3. Signed distance

A signed distance field is point-associated field that gives the distance from each point in a data set to some location or object. In the case in [Figure 3.7](#), the signed distance are computed as the nearest point on the yellow polygonal mesh. The sign of distance is determined by which side of the yellow mesh the points are located. Points inside the yellow mesh or in the opposite of the surface normal direction will have a negative distance while points outside will have positive distance. Points on the surface will be equal to a distance of zero.



**Figure 3.7.:** Signed distance



# Chapter 4.

## Method

This chapter describes the common pipeline for either a synthetic or an industrial lab system with the connected dependencies. The pipeline is divided into four sub-pipelines that describe creating an AR assembly instruction system for grinding the section cut for tubular joints. The evaluation of the pipeline concludes the chapter. Each sub-pipeline is explained with figures and diagrams. The evaluation method in the last section concludes the chapter.

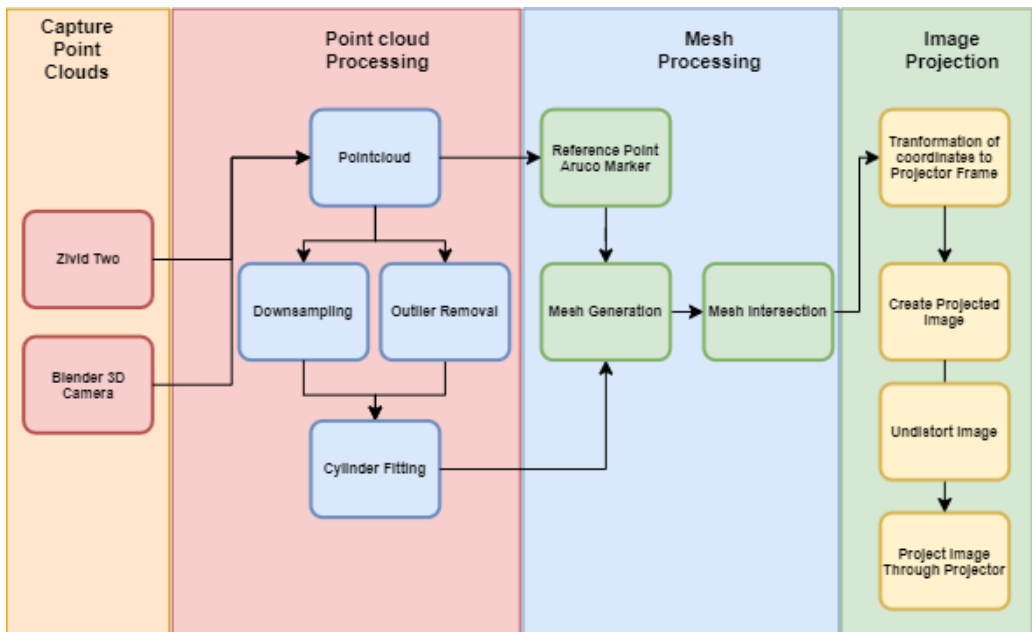
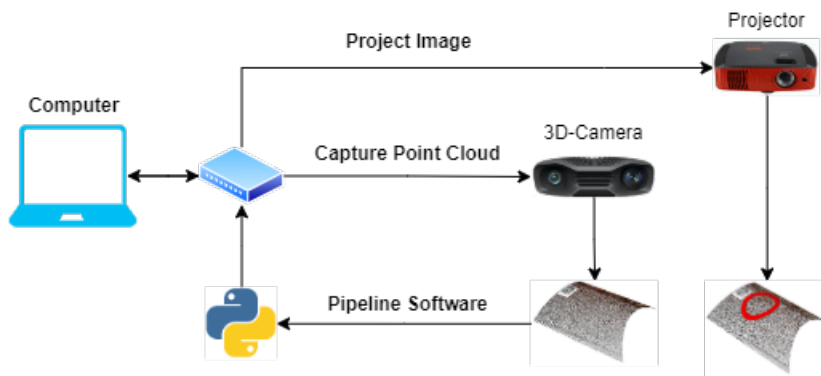


Figure 4.1.: Pipeline Diagram

## 4.1. Dependencies

This section presents the dependencies and assumptions for the hardware and software connected to the implemented pipeline.

### 4.1.1. Hardware



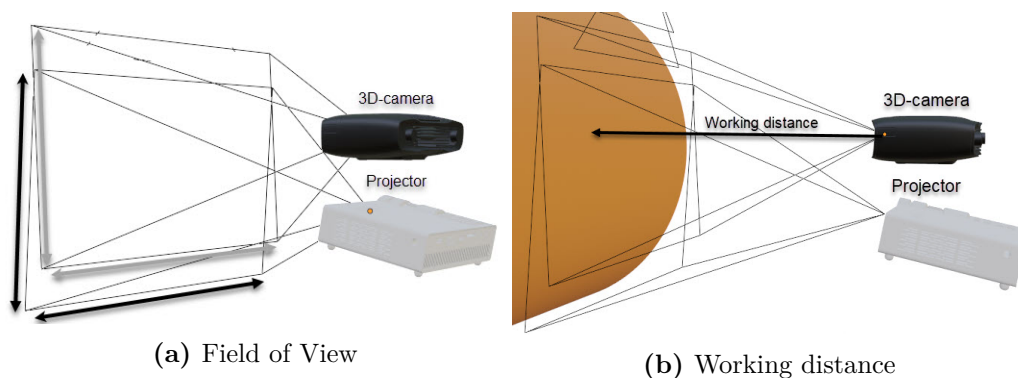
**Figure 4.2.:** Hardware Pipeline

The pipeline requires that the following hardware is available:

1. Computer - a standard computer with an external graphics card supports OpenGL and runs either Windows or a Linux platform. The computer must have connectivity options for Ethernet and HDMI or use a USB hub with these options.
2. Capture Point Clouds- An industrial or a synthetic 3D camera that captures high resolution point clouds with good precision. An advantage is to have a 3D camera that can capture a large field of view(FOV) within a short working distance.
3. Projector - a projector that can produce a large image that overlaps with the FOV of the 3D camera. An advantage is to have a short-throw projector with a short focal length, so the projector translation relative to the 3D camera is small. The projector can either be synthetically or a real-desk projector.
4. A setup that makes the 3D camera and projector rigid so a transformation can be found from a stereo calibration. This setup can either be a rig, a robot, or another kind of setup.

## Impact factors

Hardware has properties that will affect whether it fits the application or not. For example, if you look at a computer's connection to the pipeline, its task is to get the application running as quickly as possible. In any case, it has not been taken into account that the system will work in real-time for this pipeline, so how fast the code goes has been irrelevant when implementing the system.



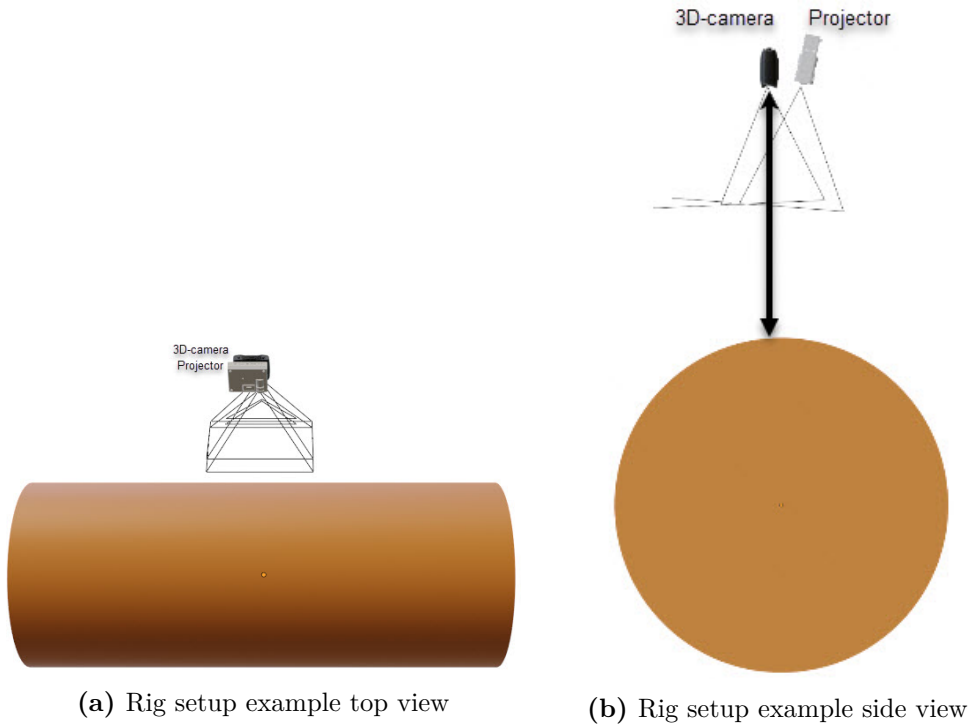
**Figure 4.3.:** Camera and projector parameters

Many factors impact a 3D-camera point of view on how the capturing result of a point cloud would be. There are many different technologies in the field of machine vision. The main factors that apply when choosing the feasible technology are factors as:

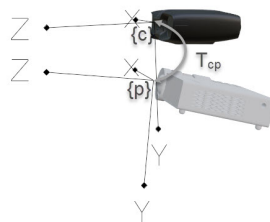
1. Working Distance - The distance from the 3D camera to the object scene.
2. Field of view - The view the 3D camera sees and is dependent on the working distance.
3. Resolution - The number of pixels in a image plane, dependent on the image size in height and width.
4. Accuracy: The accuracy compared to real measurement.

The same factors apply when choosing a projector as a 3D camera. The [Figure 4.3](#) shows the system is configured so the projector and the 3D Camera overlap in the FOV. Traditional projectors, typical desk projectors, have a much longer focal length than 3D-camera. As stated earlier, it is expedient to have a projector with a focal length in the same area as the focal length of the 3D Camera. Therefore, it is feasible for this application to have a short-throw projector with a short focal length that can throw a large image on a short distance.

The projector and the 3D camera need to be on a rigid setup that makes it easy to calibrate transformation from the camera frame to the projector frame. In addition, the setup needs to see the leg's surface from a top view as in the [Figure 4.4a](#).



**Figure 4.4.:** Rig setup example



**Figure 4.5.:** Rigid 3D-camera-projector with transformation  $T_{cp}$



### 4.1.2. Software

The software for this system is developed using python 3, and use these pip packages:

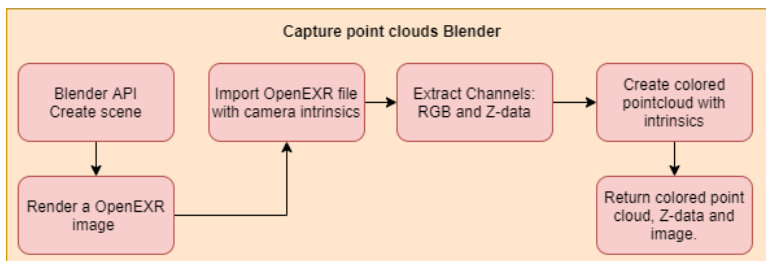
1. **Numpy**- Numpy [24] is the fundamental package for scientific computing in Python. It is a Python library that provides multidimensional array object, various derived objects(such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selection, I/O, discrete Fourier transform, basic linear algebra, basic statistical operations, random simulation and much more.
2. **Visualization ToolKit(VTK)** - VTK [22] is open source software for manipulation and displaying scientific data. It comes with the state-of-the-art tools for 3D rendering, a suite of widgets for 3D interaction, and extensive 2D Plotting capability.
3. **Vedo** - Vedo [25] is a lightweight and powerful python module for scientific analysis and visualization of 3D objects. Vedo is based on VTK and numpy with no other dependencies.
4. **OpenCV** - OpenCV [26] (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.
5. **Open3D** - Open3D [27] is an open-source library that supports rapid development of software that deals with 3D data. The Open3D frontend exposes a set of carefully selected data structures and algorithms in both C++ and Python. The backend is highly optimized and is set up for parallelization.
6. **Matplotlib** - Matplotlib [28] is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.
7. **Scipy** - SciPy [29] a scientific computation library that uses NumPy underneath. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing.
8. **OpenEXR** - OpenEXR provides the specification and reference implementation of the EXR file format. The purpose of format is to accurately and efficiently represent high-dynamic-range scene-linear image data and associated metadata, with strong support for multi-part, multi-channel use cases. The library is widely used in host application software where accuracy is critical, such as photorealistic rendering, , image compositing, and deep compositing.

## 4.2. Capture point clouds

You get an array of ordered points after capturing a point cloud, either synthetic or from real data. How the array is organized is dependent on which 3D camera you are using. The most common formats for 3D graphics are ASCII points (\*.xyz), PLY file (\*.ply), and Point Cloud Data file (\*.pcd). An advantage when working with point clouds is to have the points organized with correspondence between the pixels and the 3D points. This makes it easier to extract pixels and the coordinate for that coordinate instead of computing the 3D point each time it is needed by using the camera intrinsic and z-data obtained through triangulation.

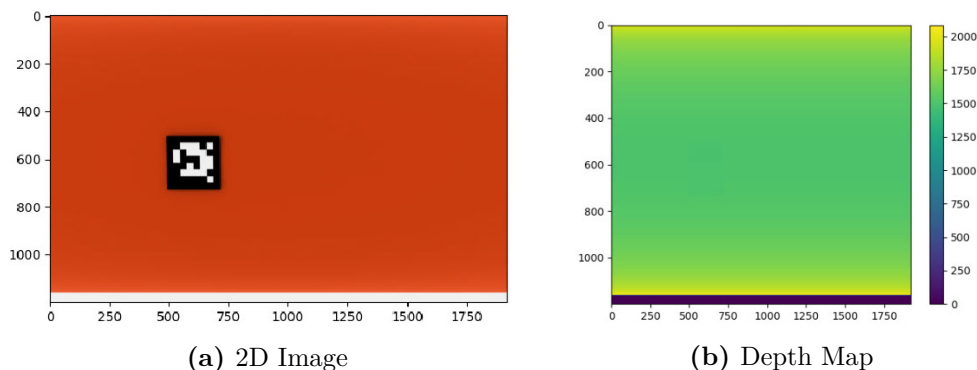
### 4.2.1. Blender point cloud

The blender point cloud is created from rendering through a scene in Blender with the format OpenEXR. The OpenEXR format allows extracting the Z-data for each pixel value in the rendered image with the RGB values. Thus, you can construct an industrial environment in Blender as your intended application. This is very practical in earlier phases when you want to test your principle. For example, you can create a 3D camera with all the properties you wish to have. It is often much easier to start with a "perfect" environment first before adding all the noise coming from a real environment.



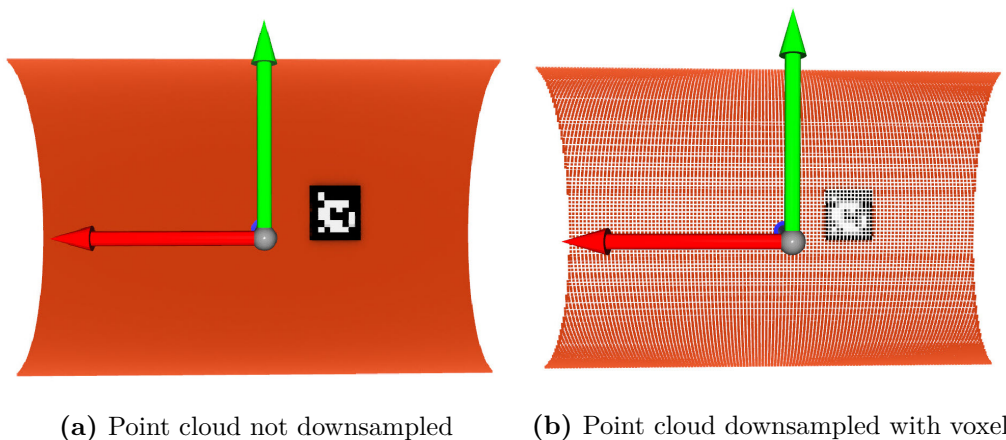
**Figure 4.6.:** Capture point cloud in Blender API

When you have rendered a multichannel image with a (.exr) format, you can import the file into a python environment with the package OpenEXR. Now, you can decompose the multichannel format to extract out the RGB-Values and Z-data. To create a point cloud now, you need the intrinsic parameters and the distortion coefficients. However, the designed camera in Blender for this application has no distortion. So the only thing required is the intrinsic parameters. By running a python [script](#) in Blender, you can extract the intrinsic parameters of all your cameras in the scene and save them to your wanted location. Below, in [Figure 4.8](#), you can see a general point cloud rendered in Blender.



**Figure 4.7.:** Blender Capture

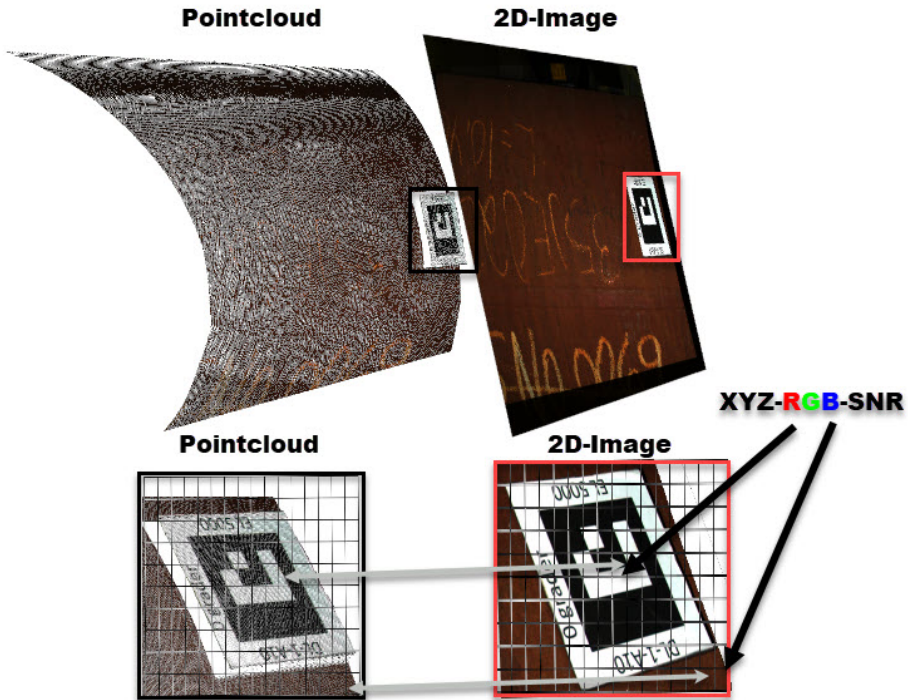
By combining the 2D image, camera intrinsic and the depth image you can create a point cloud:



**Figure 4.8.:** Point cloud captured in Blender

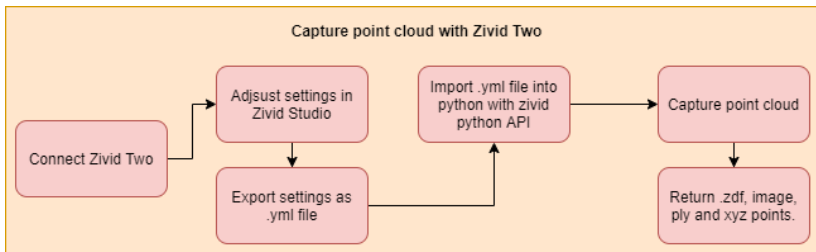
#### 4.2.2. Zivid point cloud

The Zivid Point Cloud is organized as the format Zivid Data File (.zdf), a format that has ordered the points with a 1:1 correlation between pixels in the 2D image (color and depth) and the X, Y, Z points in the point cloud. This means that the neighbor pixels in the 2D image are the neighbor points in the point cloud. The orderliness of this array speeds up the computation and lowering the cost of dealing with noisy data.



**Figure 4.9.:** Pixel correspondence of the Zivid data file

The Zivid two camera uses a sensor with 2.3 MP(1944x1200) to capture point clouds of a scene. Because of the 1:1 correlation between pixels and points, the generated point cloud consists of approximately 2.3 million points. XYZ (mm), RGB(8-bit), and SNR are provided for every pixel, where the SNR signal is the Signal-to-noise-ratio. The data is stored as an array of shapes [1944, 1200, 7].



**Figure 4.10.:** Capture Point Cloud Zivid Two

When running the pipeline, the capturing setting for the Zivid camera is set through Zivid studio, where you can adjust settings for capturing a better point cloud. However, the Zivid studio has an assisted mode that adjusts the settings to the best for the captured scene. A .yaml file is exported from Zivid studio and saved to a location where the Zivid python API can load the same settings. Then this file is used to capture with running a Python script. After capturing a point cloud image, it returns an xyz array, 2d image, and a ply-file loaded into Vedo. Also, every time a scene is captured, a .zdf is saved to easily import the data later if you want to adjust and recreate the scene and print other results/plots.

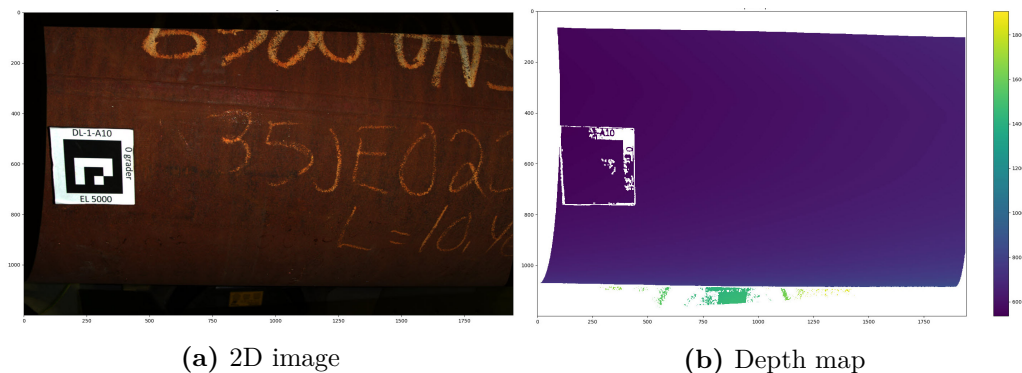
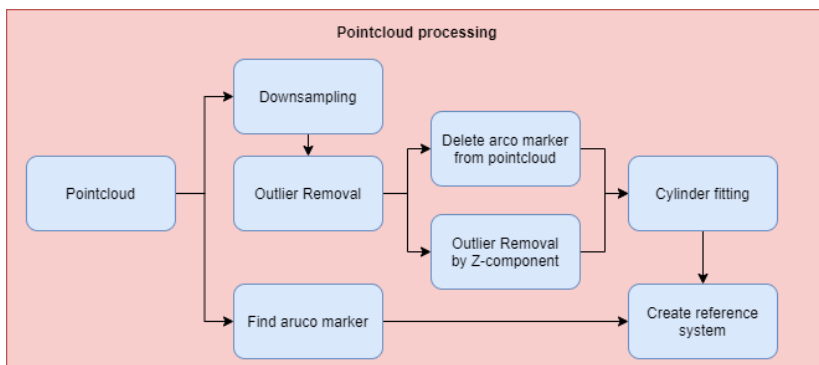


Figure 4.11.: Zivid captured point cloud

### 4.3. Pointcloud processing

Now, a point cloud is either captured synthetically or from a 3D industrial camera. The common format for both Open3d and Vedo is a ".ply" format. So when we have an xyz array of .ply and image array, it is easy to work with both Open3d and Vedo. However, point clouds, either captured in Blender or by a 3D camera, contain millions of points. Working with arrays with millions of points requires high computation time, and often it isn't necessary to have so many points. Therefore, downsampling is done to shorten computation time. Usually, downsampling and outlier removal/noise is made parallel to get a more workable point cloud.

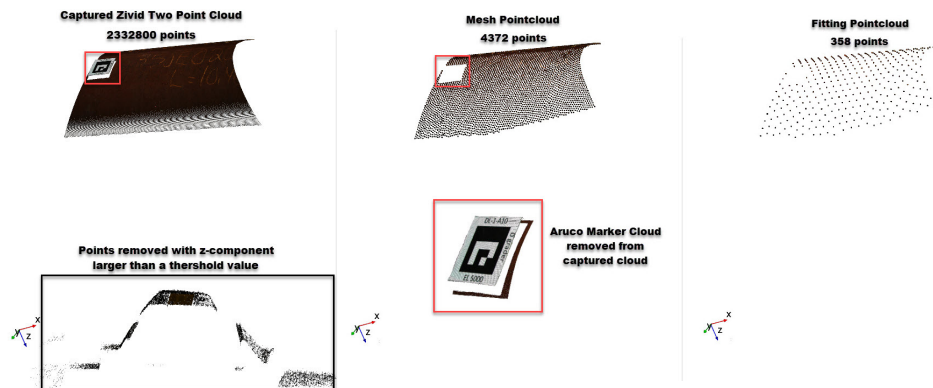
In Vedo, all objects imported into the library are converted into a poly data object. This object can be a surface mesh structure that can hold data arrays in points, cells, or in the dataset itself. Open3d converts all objects into a point cloud data file (.pcd) invented by the pointcloud.org library. The data file format can store xyz data, xyz data + colors, xyz- surface normals, and moment invariants.



**Figure 4.12.:** Point cloud processing

### 4.3.1. Downsampling and outlier removal

Downsampling and outlier removal of a point cloud can be done in many different ways depending on which approach you want to have. However, mainly computation time, density, and distance between points define the compression of a high-resolution point cloud. In this pipeline, the main functions for downsampling have been downsampling through voxels, radius outlier removal, and removing points that are close or too far from the scene/camera to an object.



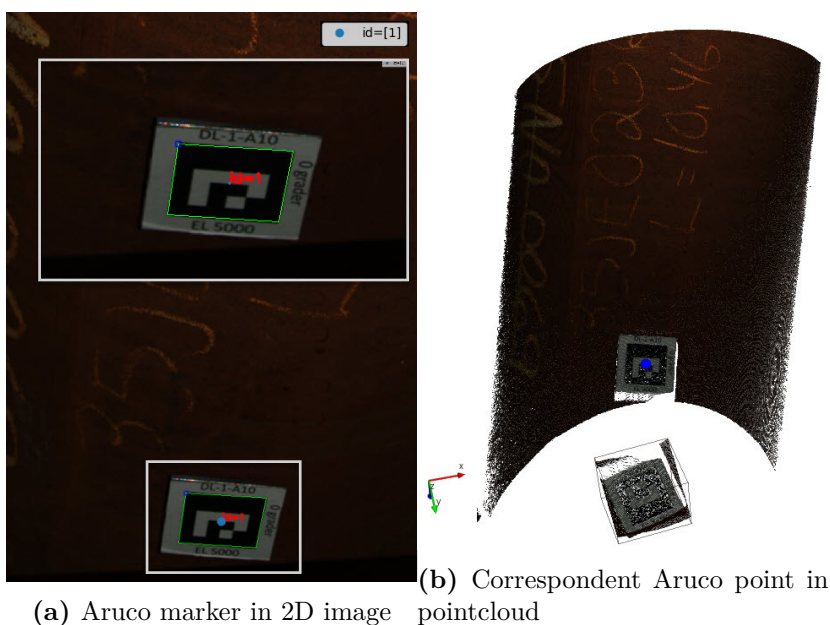
**(a)** Delete points within a Z-value  
**(b)** Downsample and delete the Aruco marker  
**(c)** Downsampled for cylinder fitting

**Figure 4.13.:** Downsampling the Zivid point cloud

In vedo an option is to clean a pointcloud with a function that create a bounding box around a point with a tolerance value that defines how far the points should be from each other in terms of fraction of the bounding box length. The tolerance value is set in terms of the unit used in the pointcloud So the value is tuned

for desired result. In the [Figure 4.13](#) you see a Zivid captured point cloud that is downsampled for both creating a mesh and for fitting. In Open3d you can downsample within a voxel filter that defines 3D voxel grid(think about a voxel grid as set of tiny 3D boxes in space). Then, all points in each voxel(3D box) will be downsampled to the approximated centroid in each voxel. After the pointcloud is downsampled to a less concentrated point cloud it is easier to remove outliers with a nearest neighbour search since the outlier clusters arent so concentrated as it was. Vedo can remove outliers from a cloud of points within a specified radius search, and can define how many neighbour points that should be deleted from the point cloud. However, sometimes it isn't so easy to delete all outliers/noise from a industrial 3D camera. So for this scene some points are deleted by the range of the z-component using Numpy to find all indices that have a z-component that is larger or smaller than a threshold value.

The Aruco marker in the figure is defined as noise for the point cloud for either mesh or cylinder fitting. To delete the marker from the scene, the four corners in the image below are found in pixel values with the OpenCV function for finding an Aruco marker. Then the correspondent points in the point cloud can be found to define a bounding box to delete all the Aruco marker points as in the [Figure 4.14](#).



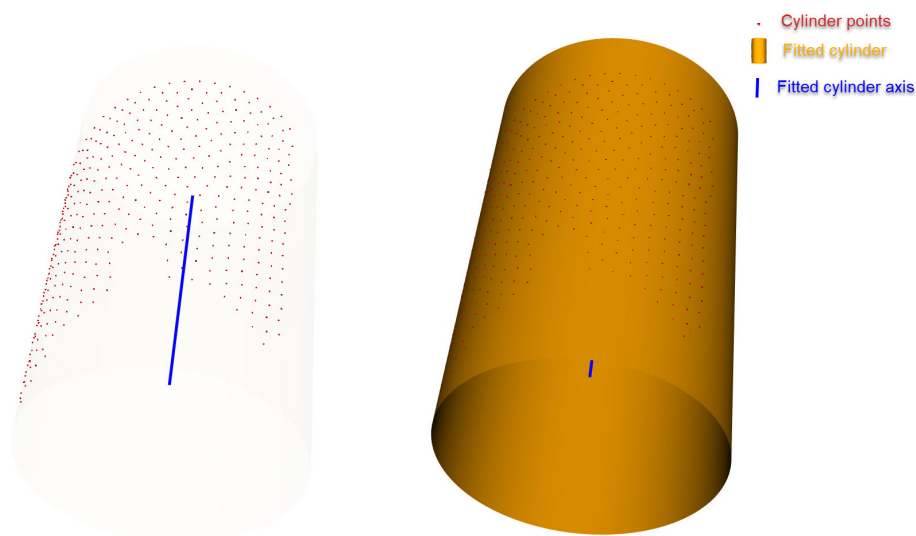
**Figure 4.14.:** Zivid Aruco marker



### 4.3.2. Cylinder fitting

A point cloud is now downsampled, so the number of points shows a fair distribution of points that show how the leg lay in the scene. To define a reference system along the leg, defined axes are needed to set a configuration between the leg and the stub. A cylinder fitting is set to define a common direction axis for the points in the [Figure 4.15a](#).

The algorithm for this is from David Eberly's paper [30], *Fitting 3D Data with a Cylinder*. Xingjie Pan implements the algorithm in Python from the following repo. The algorithm returns a direction axis for the points, a radius, and a point along the direction axis. A reference system can be created from the direction axis and the Aruco marker to move points with a set translation along the direction axis. When running the algorithm, we get a direction vector(axis) as in the [Figure 4.15a](#).



(a) Fitted cylinder axis to cylinder points

(b) Fitted cylinder

**Figure 4.15.:** Cylinder fitting

Since the axis to the cylinder and the Aruco reference point is now known, a reference system can be defined from the axis and the Aruco point as described in the next section.



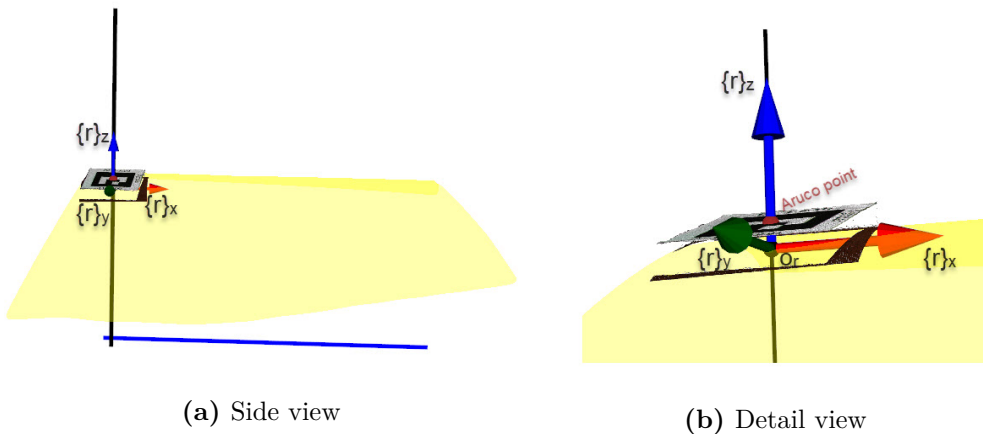
### 4.3.3. Reference system

A point cloud has been created, the data has been cleaned of noise, a reference point from the Aruco marker has been found, and a common axis for the leg's surface points is defined. The next step is to define a reference system to determine a stub configuration that intersects with the leg's surface. With the reference system, the idea is to rotate the stub around reference system axes and add a desired translation along the leg's axis.

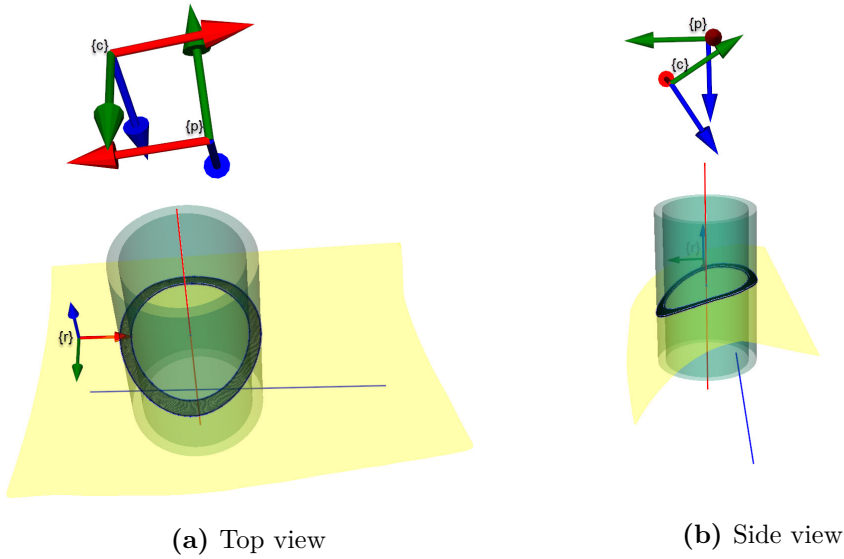
A lot of geometry is defined in the pipeline, and in order not to confuse the different coordinate systems, axes, points, and surfaces with each other, all defined names are introduced before it is explained up against the pipeline in [Figure 4.17](#).

Before the reference frame  $\{r\}$  can be defined, the leg mesh,  $L_m$ , surface need to be created. A more detailed description of this is in [section 4.4](#). The  $\{r\}$  is defined as following in [Figure 4.16](#):


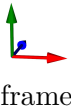











1. The origin point,  $O_r$  of  $\{r\}$  is defined by intersecting a line, the perpendicular vector of the  $L_a$  going through the Aruco point in the point cloud with the  $L_m$ .
2.  $\{r\}_x$ , is the leg axis,  $L_a$ , vector return from the cylinder fitting in [subsection 4.3.2](#).
3.  $\{r\}_y$ , the cross product between  $\{r\}_x$  and  $\{r\}_z$ .
4.  $\{r\}_z$ , is the perpendicular vector to  $L_a$  and is coincident with black axis in [Figure 4.16](#).



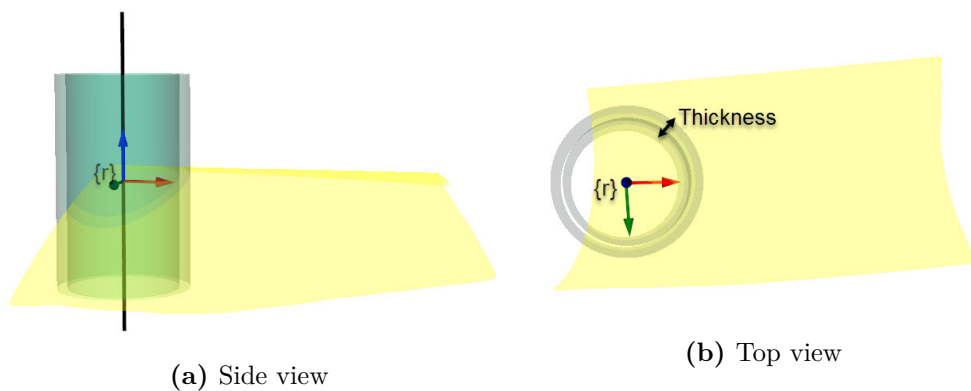
**Figure 4.16.:** Reference system frame



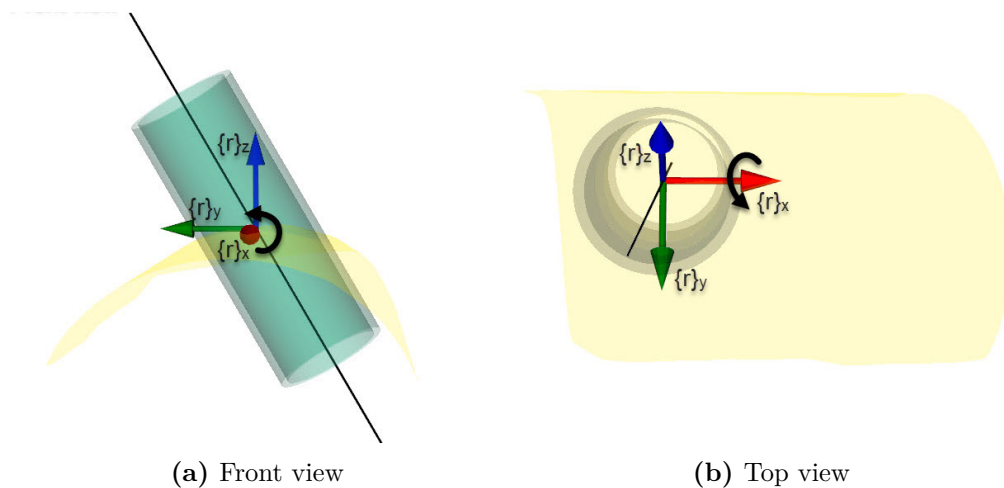
**Figure 4.17.:** System variables

- |  |   |
|--|---|
| a)  $L_m$ - Leg mesh  | h)  $\{r\}$ - Reference system frame                       |
| b)  $L_a$ - Leg axis  | i)  $O_r$ - Origin point of $\{r\}$                       |
| c)  $S_m$ - Stub mesh  | j)  $\{c\}$ - Camera frame                               |
| d)  $S_{cm}$ - Stub concentric mesh                           | k)  $\{p\}$ - Projector frame                            |
| e)  $S_a$ - $S_m$ and $S_{cm}$ has same axis                  | l)  $A_g$ - Area grinding                                |
| f)  $I_{sm}$ - Intersection curve between $S_m$ and $L_m$     | m)  $I_p$ - Intersection point between $S_a$ and $L_m$ . |
| g)  $I_{scm}$ - Intersection curve between $S_{cm}$ and $L_m$ |   |

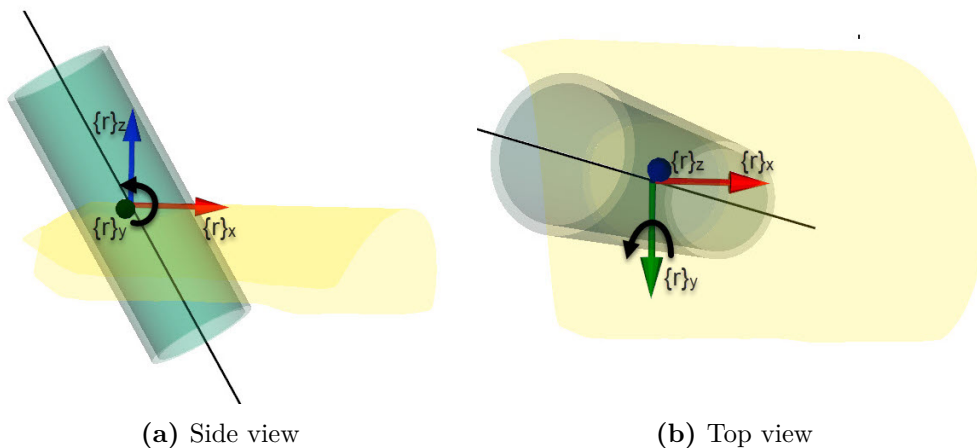
A positional reference system,  $\{r\}$ , is established for creating different configurations for a stub orientation and translation relative to  $\{r\}$ . When a configuration of a stub,  $S_m$  and  $S_{cm}$  are made. The two cylinders are defined, so the  $S_a$  is coincident with  $\{r\}_z$ -axis. Then the orientation configuration can be applied by rotating around one or multiple of the axes of  $\{r\}$ . The difference between the  $S_m$  and  $S_{cm}$  define the thickness of the stub. Figure 4.18-Figure 4.22 show examples of different configuration with orientation and translation.



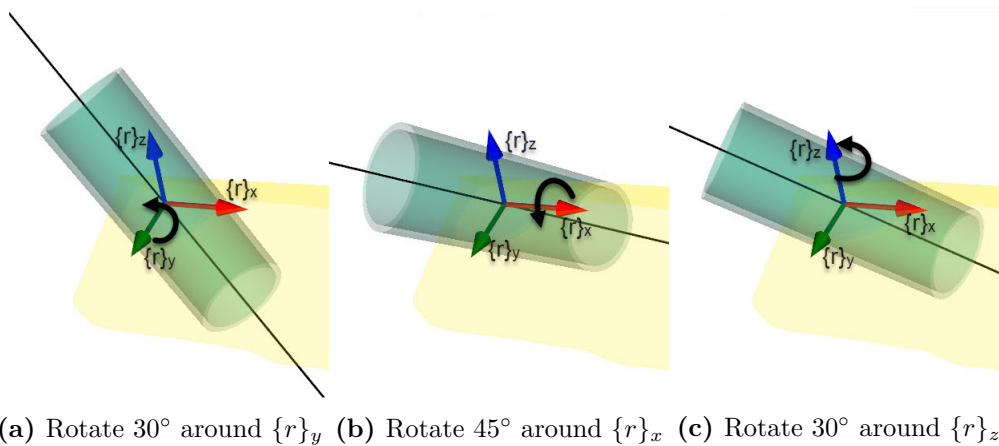
**Figure 4.18.:** Initial stub configuration



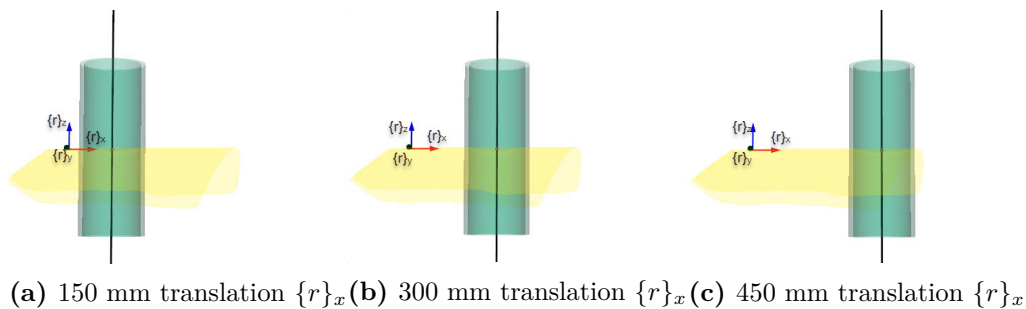
**Figure 4.19.:** Stub configuration with  $30^\circ$  rotation around  $\{r\}_x$



**Figure 4.20.:** Stub configuration with  $30^\circ$  rotation around  $\{r\}_y$

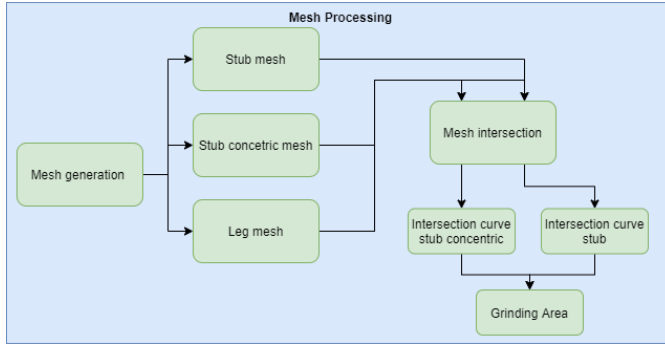


**Figure 4.21.:** Stub configuration with rotation  $\{r\}_y$ - $\{r\}_x$ - $\{r\}_z$ .



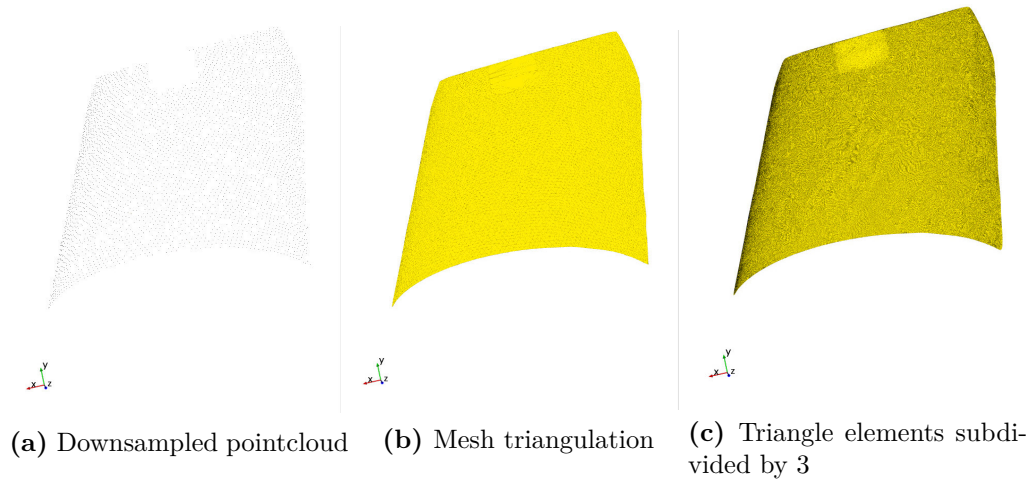
**Figure 4.22.:** Stub translation configuraion

## 4.4. Mesh processing



**Figure 4.23.:** Mesh processing

A mesh is normally computed of a collection of vertices, edges, and faces that defines the shape of a polyhedral object. The faces typically consist of triangle elements. The polygon mesh is created using Delaunay triangulation by projected the  $x$ - $y$  points from [Figure 4.24a](#) onto a plane and find the best indices of triangle elements. It is then matching the triangle polygons between the same indices between  $xy$ -Plane and the real point cloud, [Figure 4.24b](#). An advantage here is that the Zivid captured point cloud is a 1:1 correspondent pixel point cloud makes it much faster to find the triangle vertices. Since intersection between two meshes is planned to do, a subdivision filter is added to all the elements in [Figure 4.24c](#) to produce more elements. This means subdividing all triangle polygons by 3, which means making three new triangles inside one triangle.



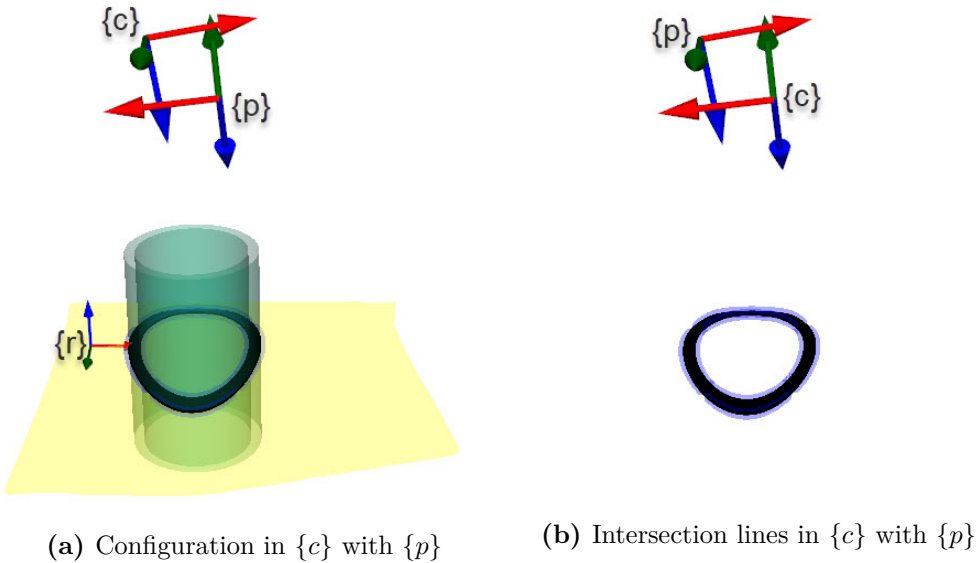
**Figure 4.24.:** Mesh generation of  $L_m$

The leg mesh,  $L_m$  is now created, and to make a configuration of stub meshes, two synthetic cylinders need to be defined. In Vedo [25] there are functions for creating different shapes, such as cylinders. Therefore, two synthetic cylinders can be created with a defined radius, axis, and resolution to define  $S_m$  and  $S_{cm}$ . As stated earlier in subsection 4.3.3, the stub configuration is created in  $\{r\}$  where the  $S_a$  is coincident with the  $\{r\}_z$ -axis as in Figure 4.18. From here, the  $S_m$  and  $S_{cm}$  can be defined with orientation and translation within  $\{r\}$  limits.

#### 4.4.1. Mesh intersection

A stub configuration is set with a defined orientation and translation relative to  $\{r\}$ . Then the next is to intersect the  $S_m$  and  $S_{cm}$  with  $L_m$  to define  $I_{sm}$  and  $I_{scm}$ . The difference in radius between  $S_m$  and  $S_{cm}$  will define the thickness of the tubular stub and the  $A_g$ .

Through Vedo [25] all mesh objects are defined as a VtkPolydata object. This object can represent geometric structures such as vertices, lines, polygons, and/or triangle strips. Point and cell attribute values(e.g., scalars, vectors, etc.) also are represented. However, to intersect two meshes, the meshes need to be defined as such an object. Intersecting is done through the function `intersectionWith` in Vedo, that uses the `vtkIntersectionPolyDataFilter` in VTK [22]. This filter computes the intersection between two `vtkPolyData` objects and returns a line/curve. In this case this function return  $I_{sm}$  and  $I_{scm}$ .



**Figure 4.25.:** Mesh Intersection in  $\{r\}$

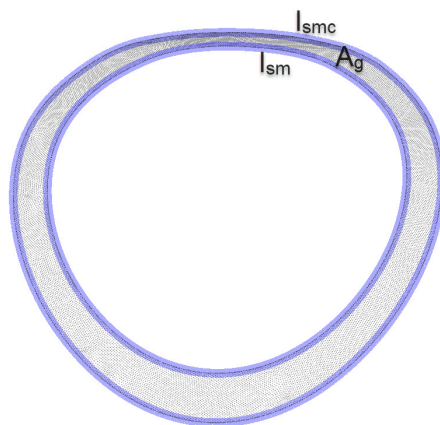


Figure 4.26.: Intersection lines,  $I_{sm}$  and  $I_{scm}$ , and grinding area  $A_g$ .

## 4.5. Image projection

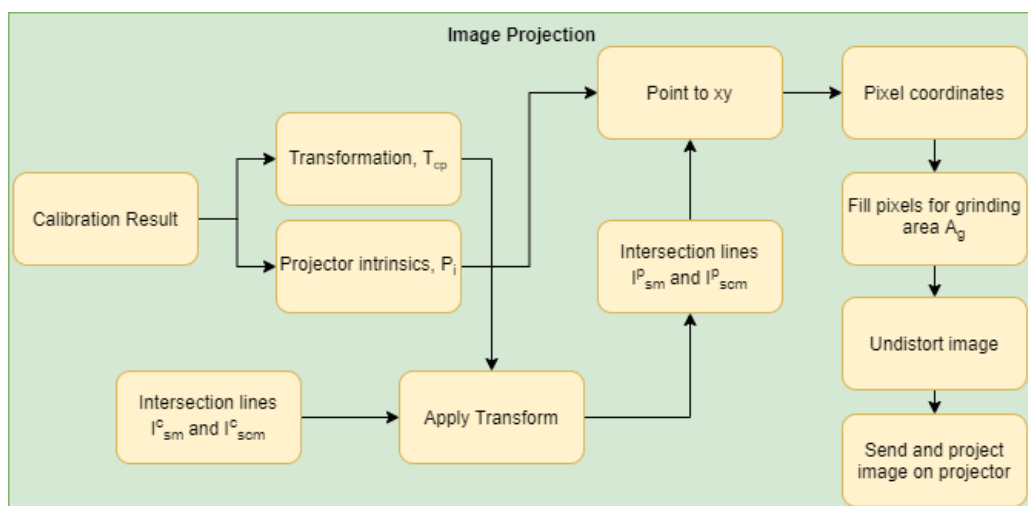


Figure 4.27.: Image projection

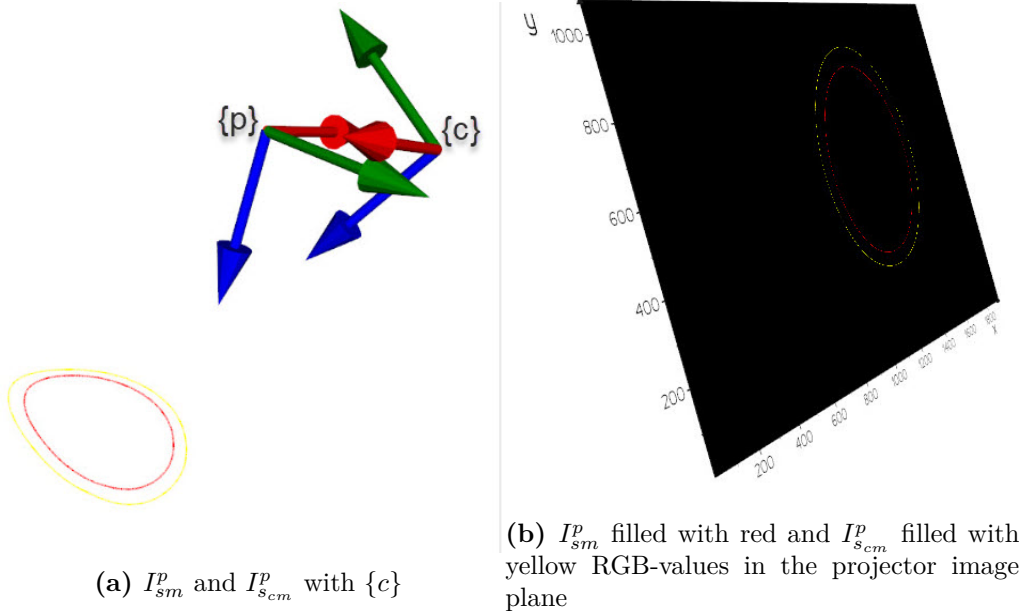
### 4.5.1. Calibration

Before the  $A_g$  can be highlighted onto the leg's surface, a projector intrinsic and a stereo calibration need to be done in order to transform the  $I_{sm}$  and  $I_{scm}$  in to the  $\{p\}$ - frame. The reader is referred to [Appendix B](#), Zivid and Projector Pair Calibration, for a more detailed description and further illustrations of this

calibration method. The checkerboard used for calibration is a 11x17, row by columns checkerboard in [Figure C.1](#), the returned projector intrinsic, distortion and the transformation between  $\{c\}$  and  $\{p\}$  is denoted as  $P_i$ ,  $P_d$  and  $T_{cp}$ .  $P_i$  and  $T_{cp}$  is as [\(3.16\)](#) and [\(3.5\)](#). In [Figure 4.17](#), you can see a transformation  $T_{cp}$  from  $\{c\}$  to  $\{p\}$  frame.

#### 4.5.2. Create projected image

From mesh intersection, the  $I_{sm}^c$  and  $I_{scm}^c$  is defined to be in the  $\{c\}$ - frame. Before a image can be created with  $P_i$ , the coordinates of  $I_{sm}^c$  and  $I_{scm}^c$  must be transformed into  $\{p\}$ - frame as  $I_{sm}^p$  and  $I_{scm}^p$ . This transformation is done as in described in [subsection 3.1.2](#). When the coordinates are relative to the  $\{p\}$  frame, the coordinates of  $I_{sm}^p$  and  $I_{scm}^p$  correspondent pixels coordinates in the projector image plane can be found with [\(3.16\)](#). Then each pixel coordinate is assigned with a RGB-value in the projector image plane as shown in [Figure 4.28](#) below.

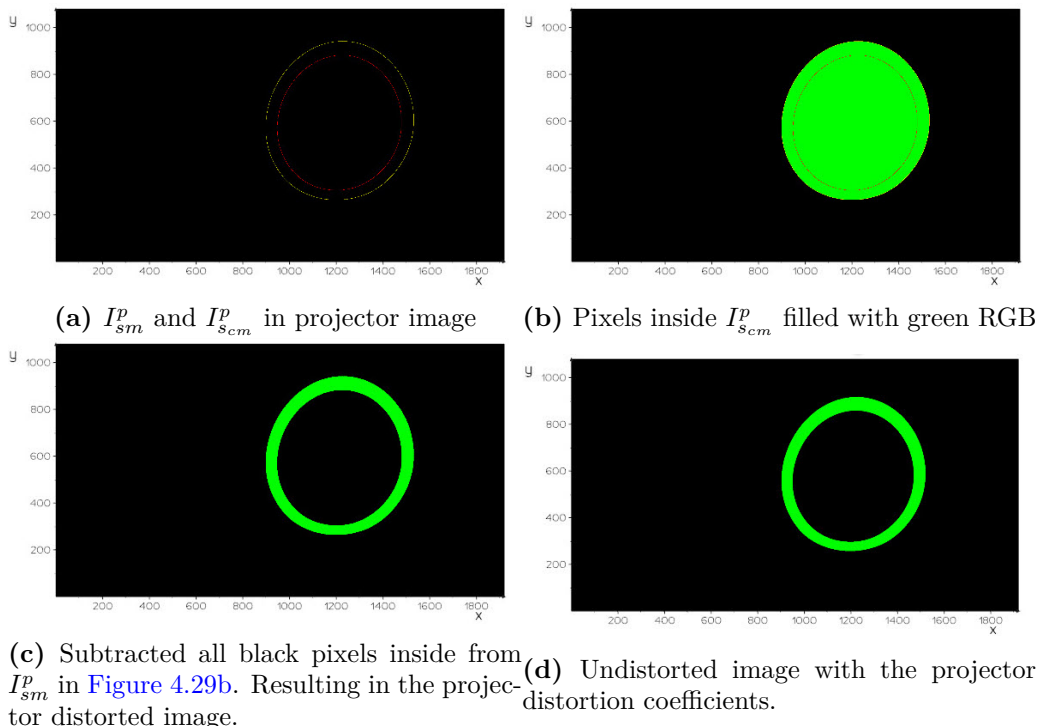


**Figure 4.28.:**  $I_{sm}^p$  and  $I_{scm}^p$  mapped to the projector image plane.

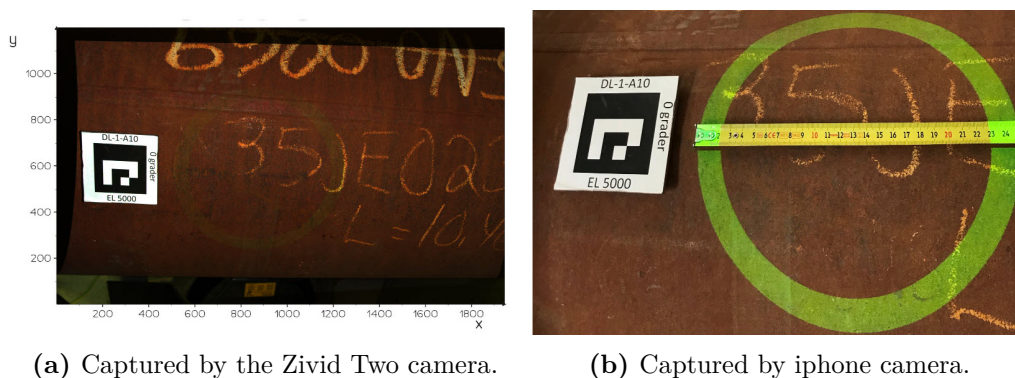
The contours of the intersection lines  $I_{sm}^p$  and  $I_{scm}^p$  are now defined in the image plane. Instead of iterating of all pixels values between the two contours in [Figure 4.29a](#), the function `cv2.fillPoly()` can fill all the pixels with a RGB-value inside that contour. Therefore, all pixels inside the yellow contour are first filled with green RGB-values as in [Figure 4.29b](#). Then all the pixels inside the red contour



are filled with black RGB-values and subtracted from the [Figure 4.29b](#), and the result is the distorted projector image in [Figure 4.29c](#). The only thing left now is to undistort the image with the projector distortion coefficients,  $P_d$ , and send the image to the projector. When the image is sent to the projector, it gives us an image as in [Figure 4.30](#).



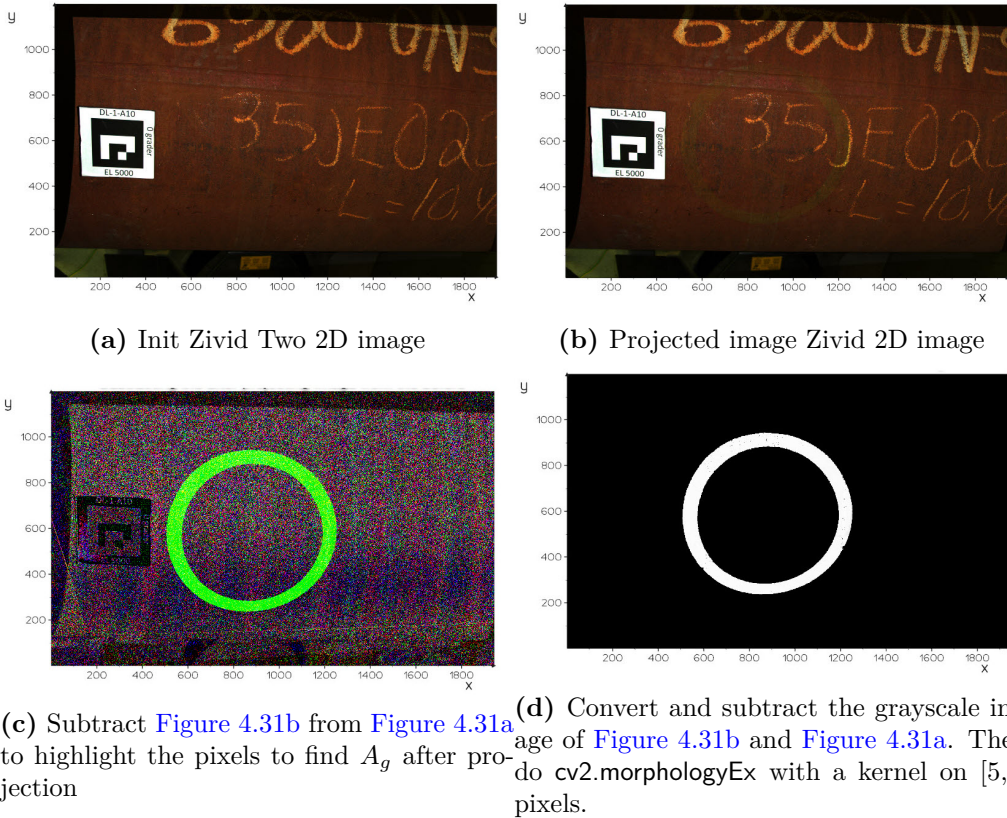
**Figure 4.29.:** Creation of the projection image



**Figure 4.30.:** Image [Figure 4.29d](#) projected onto the leg surface

## 4.6. Evaluation

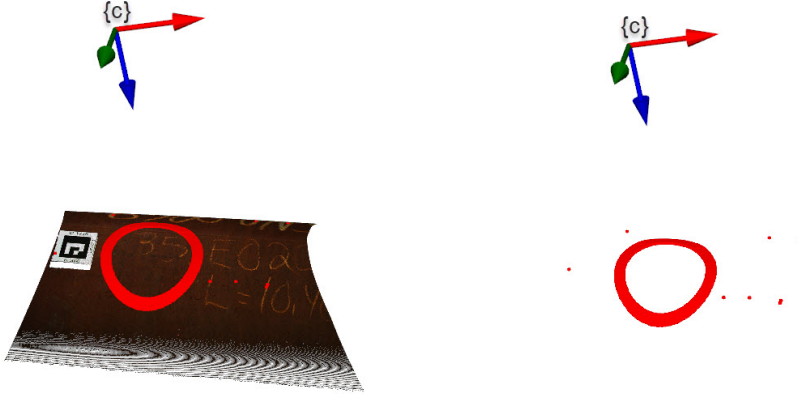
After the  $A_g$  is projected onto the leg surface, it is important to somehow validate if  $A_g$  its the same before and after projection from the projector. As seen in [Figure 4.30a](#) it difficult to spot the green  $A_g$  in the Zivid image plane. Therefore, in order to highlight and somehow extract the pixels that corresponds to  $A_g$  in [Figure 4.30a](#), a new point cloud, [Figure 4.31b](#), with 2D-image are captured to compare it with the initial point cloud in [Figure 4.31a](#) that defined the  $L_m$ .



**Figure 4.31.:** Pixels of  $A_g$  after projection

When the new 2D image in [Figure 4.31b](#) is subtracted from the init image in [Figure 4.31a](#), you can see that the green pixels in the image are visible. However, it is a lot of noise in the image that needs to be cleaned and deleted. In [Figure 4.31d](#) the grayscale image of [Figure 4.31b](#) is subtracted from [Figure 4.31a](#) and after the function `cv2.morphologyEx` is executed for that image with a defined kernel. The size of the kernel decides whether which cluster of points is either noise or the pixels that you want to keep. Now, the pixels from [Figure 4.31d](#) can be found in

the Zivid point cloud array as shown as the red highlighted point in [Figure 4.32b](#). These pixels are denoted as grinding area after projection  $A_{gap}$ . Although after the morphology was applied, it is still some noise in the point cloud. The noise is deleted by downsampling the cloud and set a radius outlier removal as described in [subsection 4.3.1](#).



(a)  $A_{gap}$  pixels from [Figure 4.31d](#) in Zivid point cloud.

(b)  $A_{gap}$  points in  $\{c\}$ - frame.

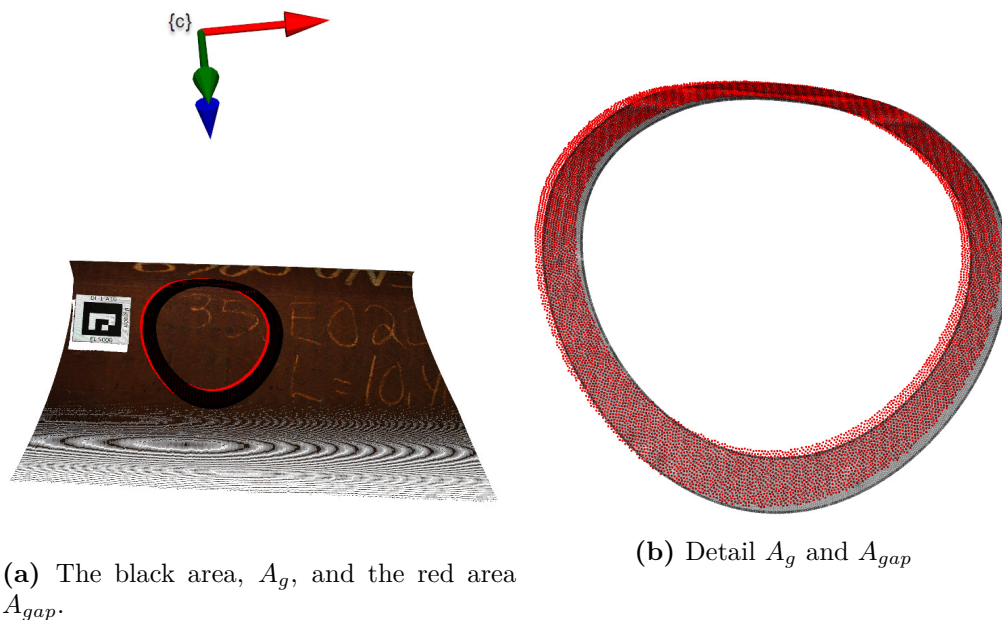
**Figure 4.32.:** Grinding area points after projection

Whether the actual cut cross-section from the current procedure will be equal to the defined initial grinding area in the system depends on the Aruco marker location on the leg's surface and the cylinder fitting that defines the defined axis system for rotation and translation. It is difficult to evaluate whether the cross-section calculated in the pipeline would look like this in real life, as you do not have a stub lying around with which you could have compared it. A more detailed assessment of this is discussed in [section 7.2](#).

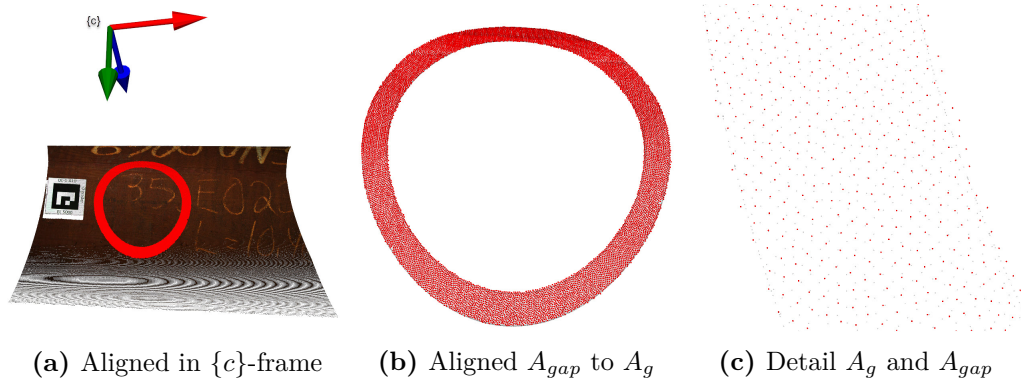
To evaluate the system deviation for the calculated  $A_g$  with the projected cross-section of the doctor  $A_{gap}$ , one looks at two factors to define a deviation within:

1. Transformation: the transformation between the  $A_g$  and  $A_{gap}$ . Iterate closest point(ICP) are applied to the  $A_{gap}$  to find the transformation matrix needed to be applied to align  $A_g$  with  $A_{gap}$ . So the transformation to move the red belt in [Figure 4.33b](#) to align with the black belt in [Figure 4.33b](#). This return the aligning transformation matrix,  $T_a$ .
2. Signed distance: Looking at the signed distance between the mesh of  $A_g$

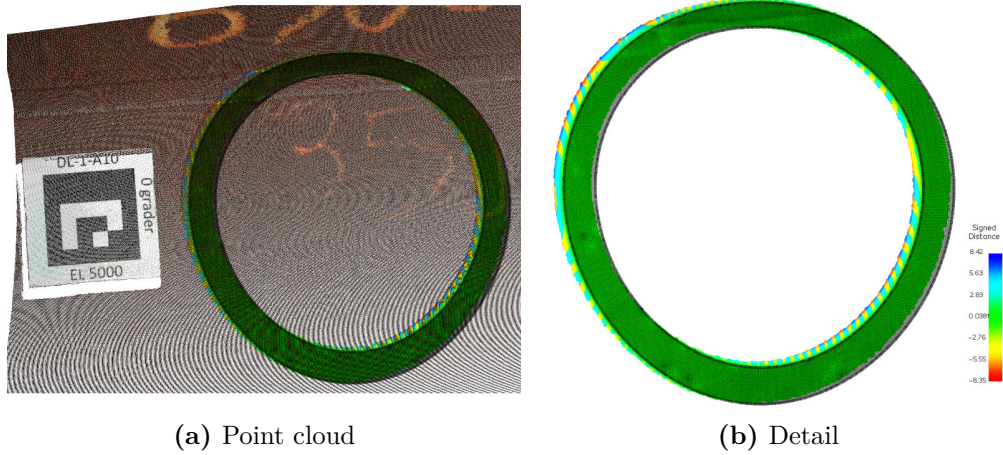
and  $A_{gap}$ . And looking on the average signed distance.



**Figure 4.33.:** Grinding area before and after projection



**Figure 4.34.:**  $A_{gap}$  aligned to  $A_g$  with ICP



**Figure 4.35.:** Signed Distance between  $A_{gap}$  and  $A_g$

From [Figure 4.34c](#) the  $A_{gap}$  has been aligned with  $A_g$ , and a transformation,  $T_a$  is estimated which gives us the rotation and translation needed to align the two point clouds with each other. Also in figure [Figure 4.35b](#) the signed distance are plotted for the surface of  $A_{gap}$  and  $A_g$ .



# Chapter 5.

## System

This chapter describes the synthetic blender and the laboratory setup to capture and project a grinding area onto the leg's surface. The setups are explained within how the setup is modeled and which hardware is used with the implemented method in the [chapter 4](#). Finally, calibration results used in [chapter 6](#) are described and presented for each system. All source code for the two systems are in this repository, [Github repository](#).

### 5.1. Blender

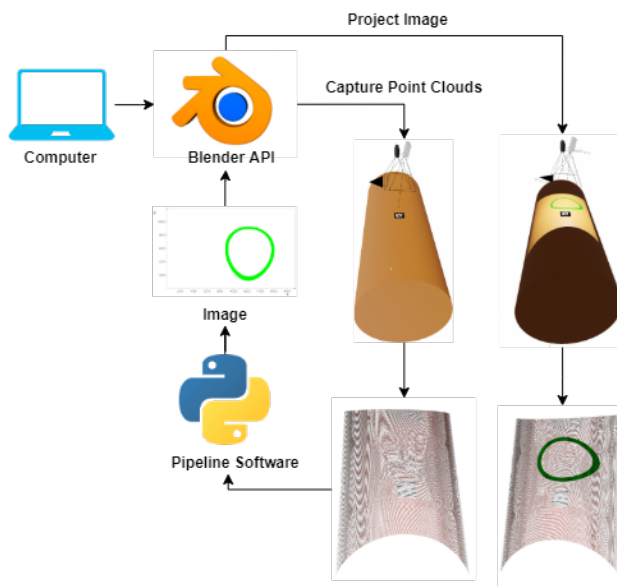


Figure 5.1.: Blender System



The synthetic setup in Blender is modeled with a stereo setup of a camera and projector with a defined transformation between the  $\{c\}$ - frame and the  $\{p\}$ -frame,  $T_{cp}$ . The advantage of the usage of Blender is that data can be captured/rendered fast. As described in the [subsection 4.2.1](#), capture point clouds blender, a point cloud can be created with the extraction of the depth(Z-buffer) from the .exr format with the intrinsic parameters and the 2D-image through functions in Open3D. This is a great tool when you don't have set up your lab environment, within this case, a projector and a 3D camera. Therefore, at the start of the project the method was initiated in Blender to capture data fast instead of waiting for a lab environment.

### 5.1.1. Camera projector setup

A camera in Blender can be set in any transformation relative to the world coordinate system of the Blender scene. The camera is modelled from the pinhole model, and has in this case no distortion. The camera's focal length, sensor height and width, and resolution can easily be tuned into desired parameters. Adjusting these parameters would give you the possibility to change the camera's FOV and the working distance to a object in a scene. The modelled blender 3D-camera intrinsic parameters is denoted as the  $K_{bc}$ . In Blender, projectors are not integrated into the API. However, there is a workaround for projecting an image in a scene with defined intrinsic parameters of the projector. For example, a projector in Blender can be modeled as a spotlight with specified light power, with normal coordinates divided by the Z-component, the image can be projected in the scene. The projector intrinsic can be extracted by modeling a camera at the same point as the projector and assuming the projector as an inverse camera with the pinhole camera model. Then, constraining the projector image plane to match the camera image plane, the scaling can be applied by scaling the projector x-y ratio of the projector image plane by the (5.1).

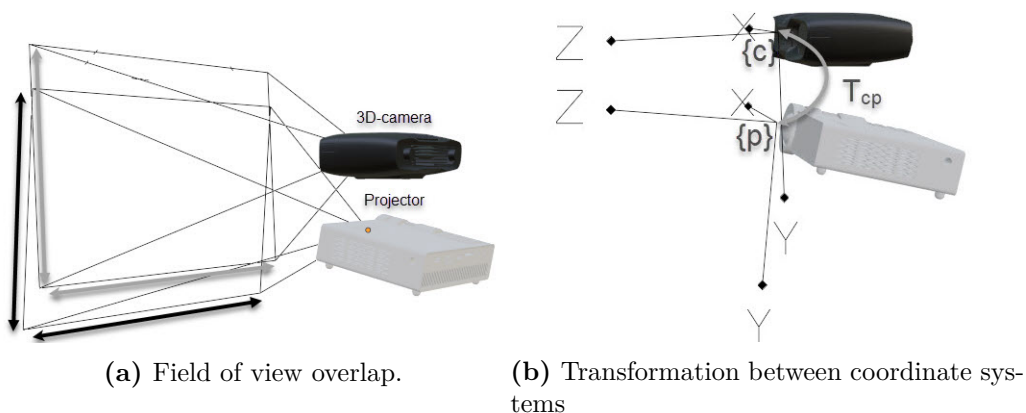
$$S_x = \frac{F_k}{S_w} \quad S_y = \frac{F_k}{S_w} \cdot \frac{r_x}{r_y} \quad (5.1)$$

Where  $S_x$  and  $S_y$  are a mapping scaling in Blender relative to the Blender world coordinate system.  $F_k$  is the focal length of the constrained camera,  $S_w$  is the sensor width, and  $r_x$  and  $r_y$  are the resolution of the camera. Since the camera and projector image planes now match, the camera's intrinsic parameters are equal to the projector in the Blender. From here, the intrinsic parameters can be extracted by running the following [script](#) in the created scene. The script gives the  $K_{bc}$ ,  $K_{bp}$



with the transformation between the camera and projector  $T_{cp}$  as :

$$K_{bc} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad K_{bp} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad T_{cp} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.2)$$

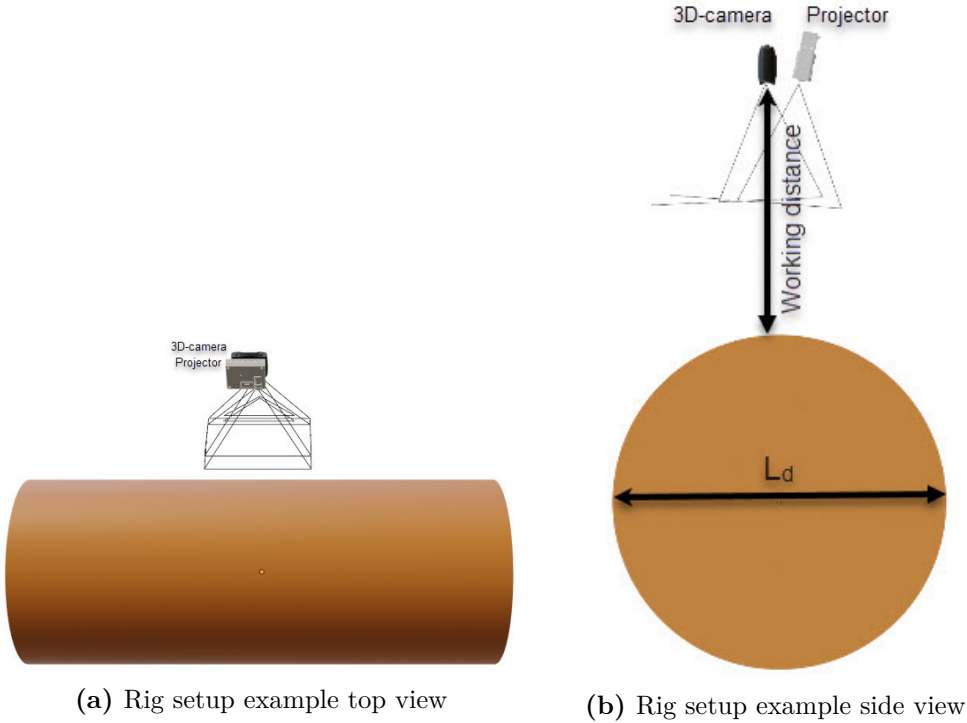


**Figure 5.2.:** Blender 3D-camera-projector setup

These properties is denoted as the Blender setup properties. As mentioned with Blender is that the camera and the projector's properties can easily be tuned or changed, and you only need to re-run the [script](#), and you get the new blender properties. Now the setup can look like in figure [Figure 5.2](#).

### 5.1.2. Scene

Since the 3d-camera-projector setup is set, the system can be moved around in the created scene. As stated, the setup is used to capture the leg's surface for visualizing the grinding area with a projector. Therefore, a synthetic leg can easily be defined in the Blender scene as a cylinder with a set diameter and placed at any location related to the 3d-camera-projector system.



**Figure 5.3.:** Rig setup example

Here the leg diameter  $L_d$  can be modified into different diameters with a wanted working distance.

### 5.1.3. Calibration

This section present the properties for the system in the conducted experiment in [chapter 6](#). The set properties for the camera-projector is these properties:

$$K_{bc} = \begin{bmatrix} 1333.3 & 0 & 960 \\ 0 & 1250.0 & 600 \\ 0 & 0 & 1 \end{bmatrix} K_{bp} = \begin{bmatrix} 1386.6 & 0 & 960 \\ 0 & 1300.0 & 600 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.3)$$

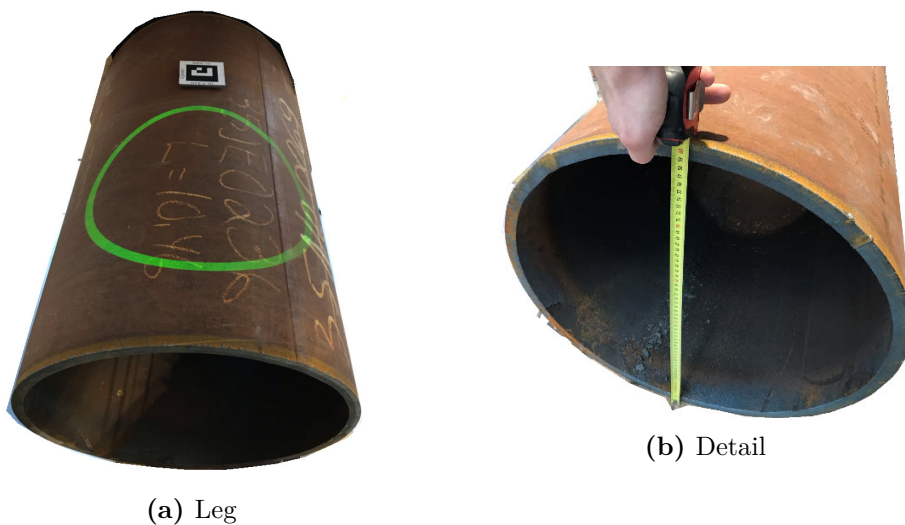
$$T_{cp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.99 & 0.12 & -0.1995 \\ 0 & -0.12 & 0.99 & 0.0139 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.4)$$

The  $L_d = 4m$  for the conducted experiment in [section 6.1](#)

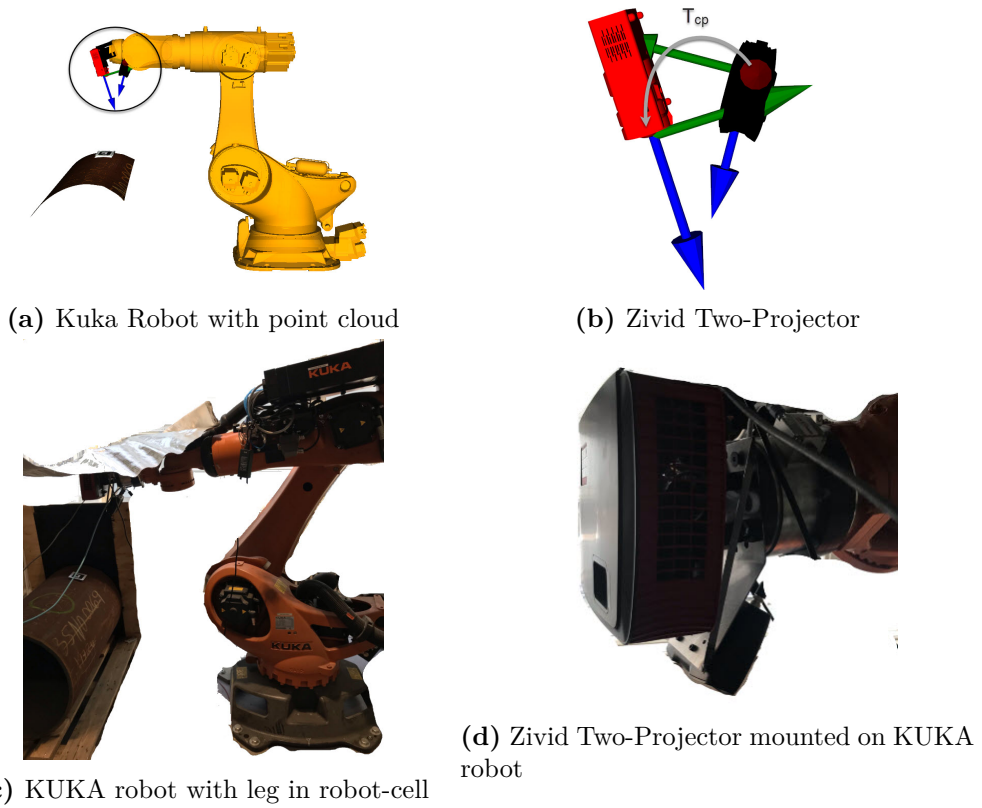
## 5.2. Lab

The lab setup is set up at the Manulab at NTNU-Gløshaugen in a robot-cell designed for SINTEF Manufacturing's projects upon the AutoKons project, as stated in [section 1.2](#). The robot-cell has two KUKA KR 100 Titan robots, which are setup for a cutting-welding task related to the AutoKons project. However, in this system, only one of these robots is used to transform the 3D-Camera-Projector around in the robot cell.

In the robot cell scene, it is a round tube, represented as a down-scaled leg. The leg's diameter dimension is approximate  $L_d \approx 600mm$  laying on the floor.



**Figure 5.4.:** Leg lab specimen



**Figure 5.5.:** Lab setup

### 5.2.1. Zivid Two-Acer Predator system

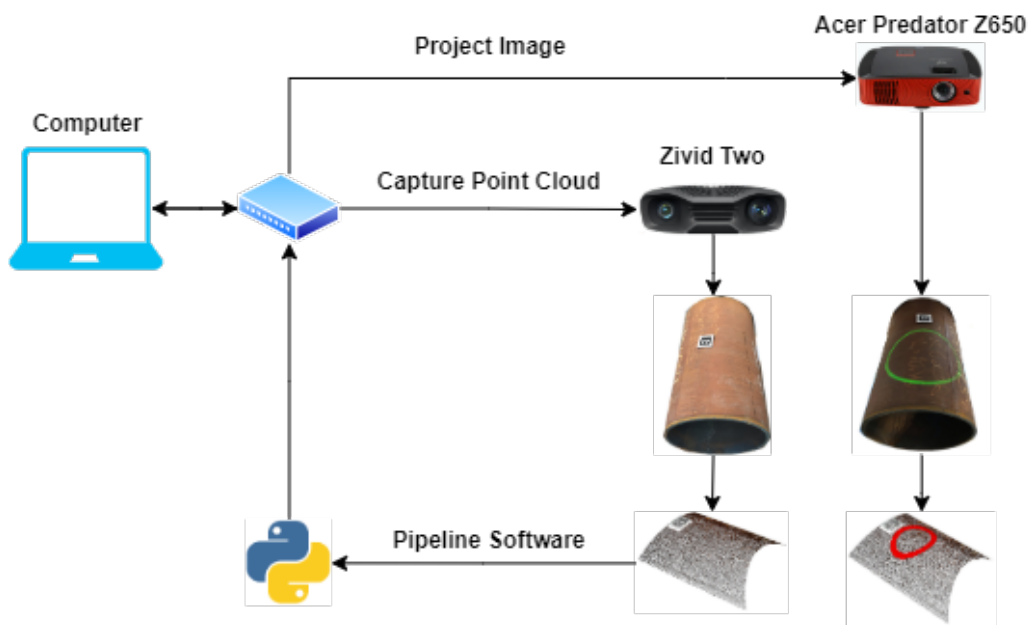


Figure 5.6.: Lab system

The Zivid two camera is a timed structure light 3D scanner that is used for capturing high-resolution point cloud in the presented pipeline in [subsection 4.2.2](#). The camera produces fast and accurate 3D point clouds, fast acquisitions, and capture a point cloud down to 60 *ms*.

Table 5.1.: Zivid Two Technical Specifications

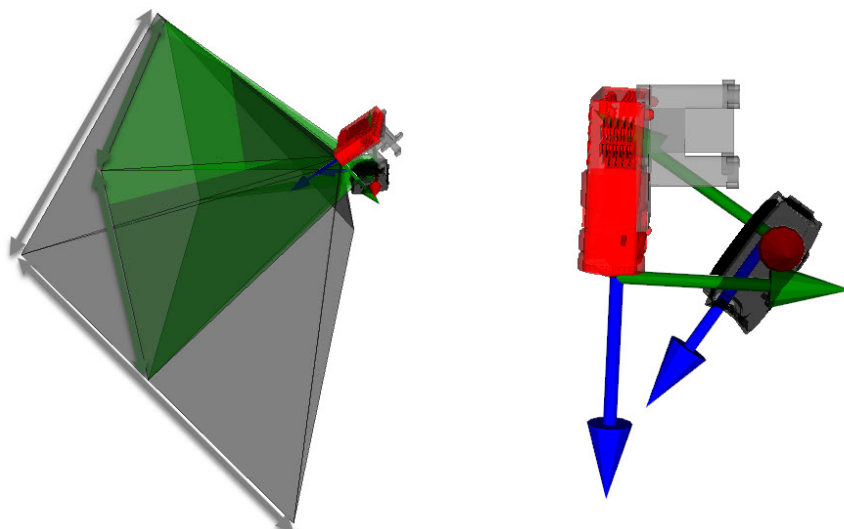
Properties	Specifications
Resolution (px)	1944 x 1200 (2.332Mp)
Point cloud Output	3D (XYZ) + Color (RGB) + SNR
Apeture	f/1.8 to f/32
Shutter (S)	1/600 s to 1/10 s
Gain (G)	1x to 16x
Projector Brightness (B)	1/4x to 1.8x 1x = 360 lumens
Recommed Working Distance	400 to 1200
Focus Distance	700
Field of view (mm)	754 x 449 at 700
Spatial resolution (mm)	0.39 at 700
Point Precision	60 $\mu$ m
Size (mm)	169 mm x 56 mm x 122 mm
Weight (mm)	880 g

The accuracy within the focus distance to the Zivid Two camera is set to  $60 \mu m$ . The HDR feature is one of the Zivid's unique capability, and make it possible to capture "difficult and shiny" objects. The HDR mode in Zivid works with multiple settings in Zivid studio, including multiple apertures in various frames. The camera creates acquisitions with a specified f-number and then combines these acquisitions into one high-quality frame.

The projector used in the system is an Acer Predator Z650 short-throw projector with a resolution of 1920x1080. This makes it possible to make a large image within a short working distance and is advantageous in this application since the Zivid two camera's working distance ranges between 400-1200 *mm*. Therefore, it is essential that the projector FOV overlap within the Zivid Two Camera.

**Table 5.2.:** Acer Predator Z650 Technincal Specifications

<b>Properties</b>	<b>Specifications</b>
Resolution(px)	1920x1080
Aperture	F/2.6 to F/2.78
Focal length(mm)	10.20-11.22
Projection distance(mm)	900-4600
Brightness(lumens)	2200
Projection system	DLP
Size(mm)	98x357x241
Weight	3.4 kg



(a) Green FOV for Zivid Two and black FOV for Acer Predator

(b) Zivid-Projector reference system

**Figure 5.7.:** Zivid-Projector system

The Zivid Two and the Acer predator are mounted onto the KUKA robot. The Zivid Two camera was already mounted onto the robot since SINTEF Manufacturing uses it in an application with a hand-eye calibration. Therefore, a bracket is made for mounting the projector onto the robot. The advantage here with the Acer predator projector is that the FOV of the projector is larger than the Zivid Two with the same working distance. Therefore, the bracket was made so the projector was orientated to overlap in the best possible way due to mounting limitations on the robot.

### 5.2.2. Calibration

When the Zivid two and the camera are mounted onto the robot. A calibration must be conducted to move coordinates from the Zivid Two camera system to the Acer predator coordinate system. Since the Zivid have already calibrated the Zivid Two camera intrinsic parameters, their calibration is used. Their calibration gives an intrinsic matrix as with distortion coefficients,  $K_z$ , and  $d_z$ . As stated in [subsection 4.5.1](#), the calibration procedure is detailed explained in [Appendix B](#). The final calibration result for the Acer Predator intrinsics,  $P_i$ ,  $d_p$ , and the  $T_{cp}$ :

$$K_z = \begin{bmatrix} 1782.09 & 0 & 977.63 \\ 0 & 1782.05 & 587.77 \\ 0 & 0 & 1 \end{bmatrix} P_i = \begin{bmatrix} 1500.72 & 0 & 972.82 \\ 0 & 1498.87 & 1101.91 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.5)$$

$$d_z = [-0.090788, 0.134410, -0.065208, 0.000578, 0.000058] \quad (5.6)$$

$$d_p = [-0.004565, 0.010215, 0.001734, 0.000440, -0.0362493] \quad (5.7)$$

$$T_{cp} = \begin{bmatrix} -0.997 & 0.001 & 0.007 & 111.94 \\ 0.003 & -0.845 & -0.531 & 97.824 \\ -0.006 & -0.532 & 0.843 & 100.82 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.8)$$

The  $T_{cp}$  is shown in [Figure 5.5b](#).



# Chapter 6.

## Experiment

This chapter presents an experiment that runs the implemented pipeline from [chapter 4](#) with the two system from [chapter 5](#).

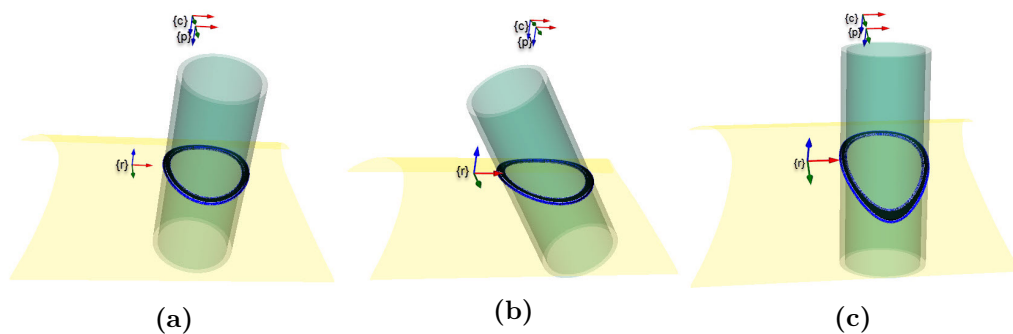
The common goal for both synthetic and lab system is to project a section area onto the leg's surface. Then evaluate the projected result by the evaluation metric presented in the pipeline in [section 4.6](#). When it comes to evaluation is it focused on the result of the deviation between  $A_{gap}$  and  $A_g$ . A decomposition of the presented results, decomposition and arguments are discussed in [chapter 7](#).

### 6.1. Synthetic

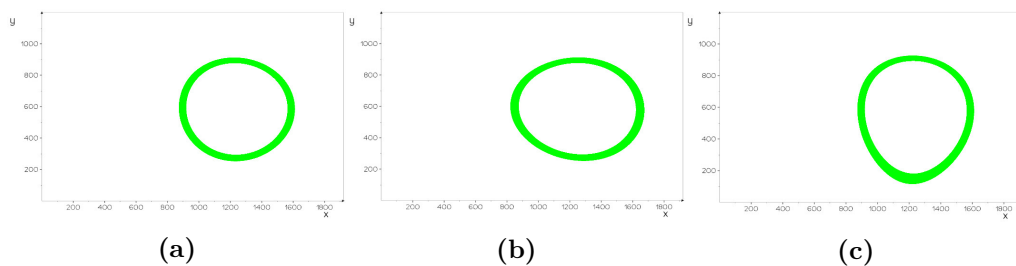
The result from the synthetic system presents the possibility for creating an assembly AR-system in Blender and look at the accuracy presented in [section 4.6](#). Thus, it isn't done a high number of results within this system has been to create a lab system. In the [Table 6.1](#), the result for the synthetic experiment are presented for the configurations in [Figure 6.1](#). The leg diameter is set to  $L_d = 4000mm$ . The default unit in blender is meters, and thereby is all figures plotted in meters.

**Table 6.1.:** Synthetic Experiment

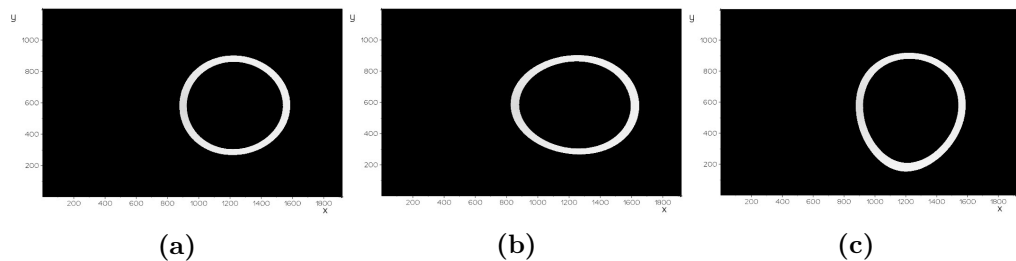
Pose 1							
Exp.	Stub OD	Stub ID	t	Config	Trans $\{r\}_x$	Rot $\{r\}_x$	Rot $\{r\}_y$
1	800 mm	700 mm	100	a, <a href="#">Figure 6.1a</a>	700 mm	0	0
				b, <a href="#">Figure 6.1b</a>	700 mm	0	30°
				c, <a href="#">Figure 6.1c</a>	700 mm	30 °	0



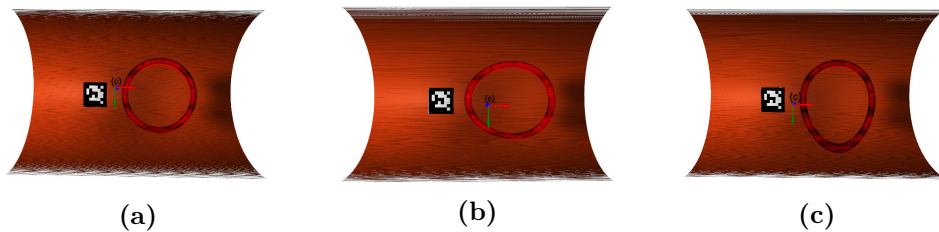
**Figure 6.1.:** Stub Configuration Synthetic



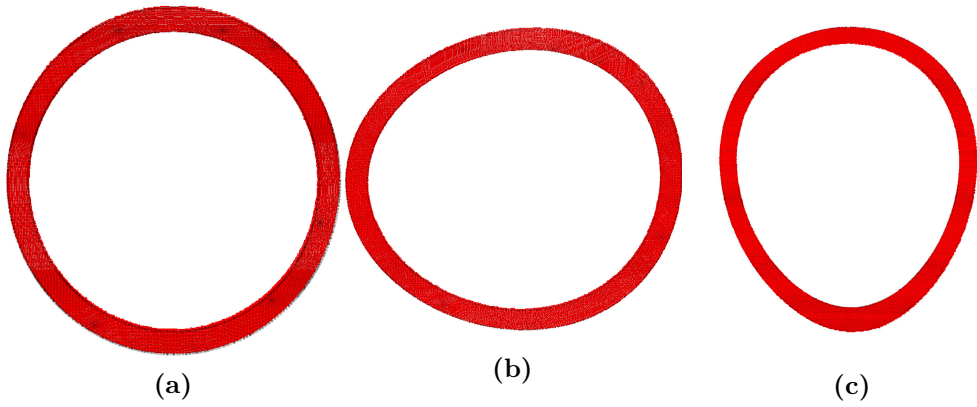
**Figure 6.2.:** Projected projector image



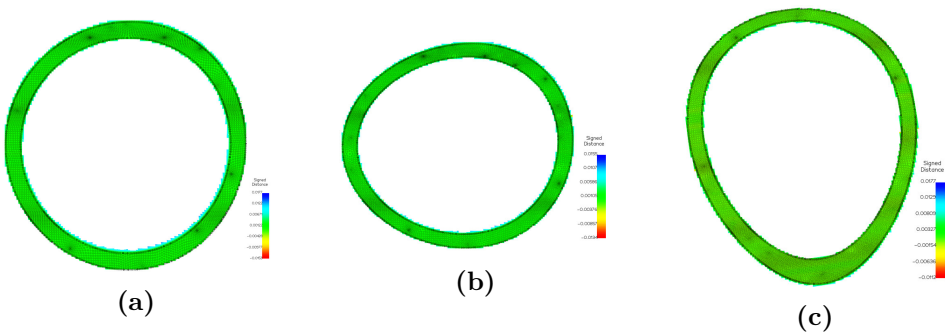
**Figure 6.3.:** Threshold image of coordinates.



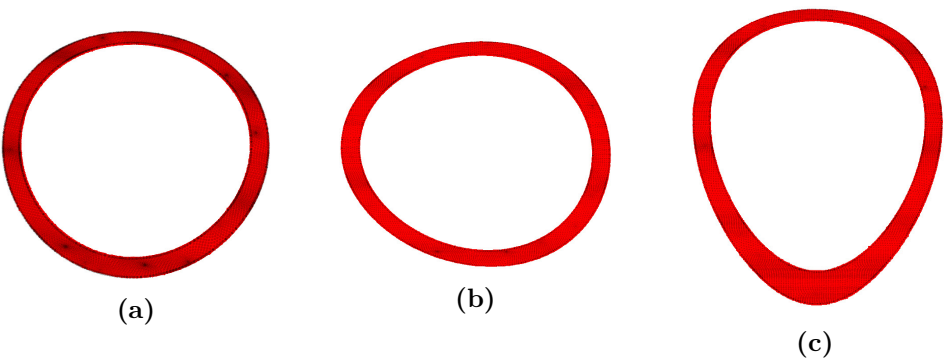
**Figure 6.4.:** Difference in predicted  $A_g$  and projected  $A_{gap}$



**Figure 6.5.:** Detail view predicted  $A_g$  and projected  $A_{gap}$



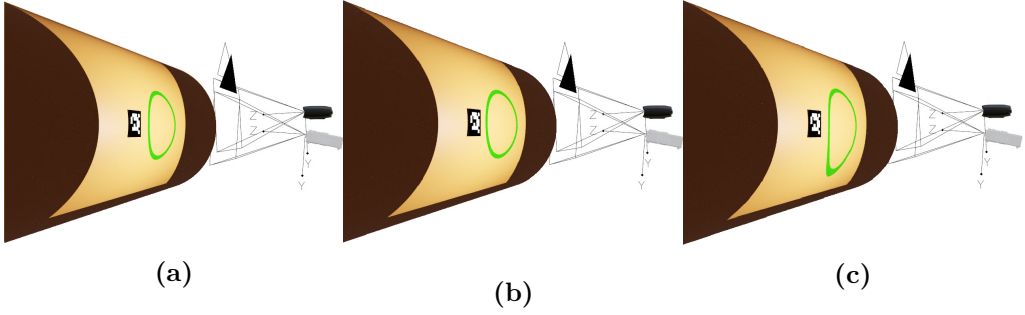
**Figure 6.6.:** Signed distance between  $A_g$  and projected  $A_{gap}$



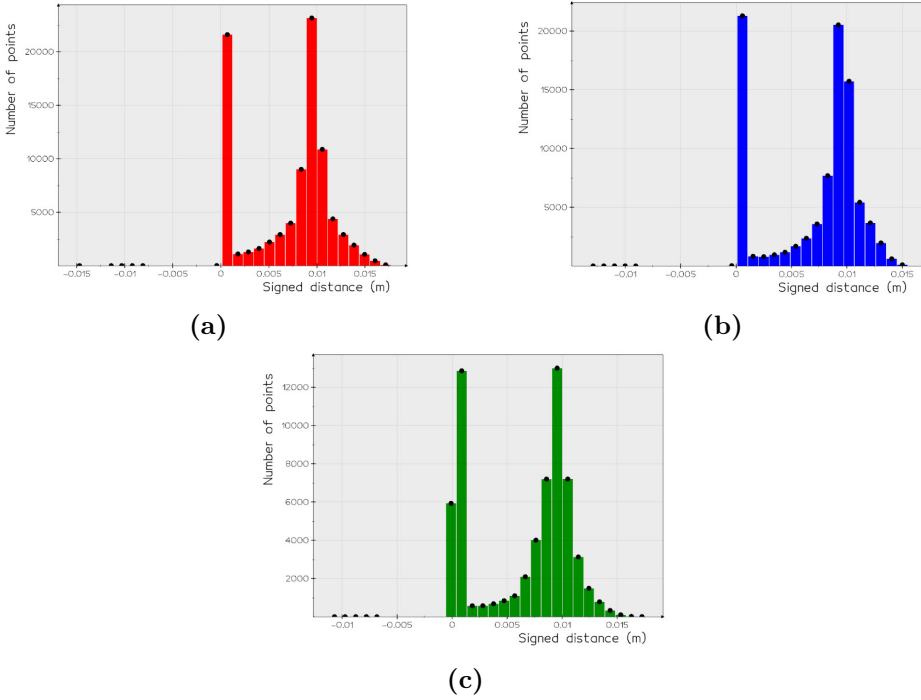
**Figure 6.7.:**  $A_{gap}$  aligned with ICP to  $A_g$

**Table 6.2.:** Transformation for aligning  $A_{gap}$  to  $A_g$  in form of euler angles and translation.

C	$\{r\}_x^\circ$	$\{r\}_y^\circ$	$\{r\}_z^\circ$	$\mathbf{x}$ (mm)	$\mathbf{y}$ (mm)	$\mathbf{z}$ (mm)	Norm( $\mathbf{x},\mathbf{y},\mathbf{z}$ ) (mm)
a	-0.16	0.003	0.008	1.33	-7.40	0.487	7.53
b	-0.09	-0.002	0.027	-0.32	-4.31	0.47	4.35
c	-0.11	-0.02	0.04	-0.10	-5.35	0.41	5.37



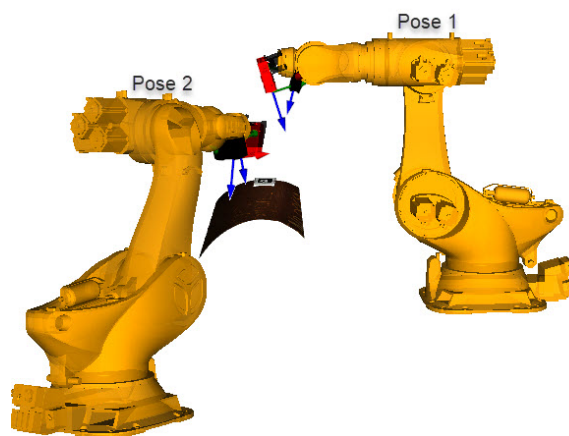
**Figure 6.8.:** Projected image in Blender



**Figure 6.9.:** Distribution of signed distance between  $A_{gap}$  and  $A_g$

## 6.2. Lab

The result from the lab setup presents the possibility for projecting AR stub section cut with a Zivid-Two Acer Predator projector system. The plotted results are as presented in [section 4.6](#). The laying leg in the robot cell is approximately  $L_d = 600mm$ . The Zivid two camera operate with millimeters, and therefore all plots are in millimeters. The [Figure 6.10](#) show the two different poses for the conducted experiment. The main focus for these experiments has been to project a section cut on the leg specimen in the robot-cell with different orientation of the stub as presented in [Table 6.3](#).



**Figure 6.10.:** Capturing Poses relative to the leg.

### 6.2.1. Pose 1

**Table 6.3.:** Pose 1 lab setup experiment

Pose 1							
Exp.	Stub OD	Stub ID	t	Config	Trans $\{r\}_x$	Rot $\{r\}_y$	Rot $\{r\}_x$
1	250 mm	210 mm	20 mm	a, <a href="#">Figure 6.11a</a>	200 mm	0	0
				b, <a href="#">Figure 6.11b</a>	300 mm	0	0
				c, <a href="#">Figure 6.11c</a>	400 mm	0	0
2	240 mm	210 mm	15 mm	a, <a href="#">Figure 6.19a</a>	300 mm	30 °	0
				b, <a href="#">Figure 6.19b</a>	300 mm	45 °	0
				c, <a href="#">Figure 6.19c</a>	300 mm	60 °	0
3	240 mm	210 mm	15 mm	a, <a href="#">Figure 6.27a</a>	300 mm	0	15 °
				b, <a href="#">Figure 6.27b</a>	300 mm	0	30 °
				c, <a href="#">Figure 6.27c</a>	300 mm	0	45 °

## Experiment 1

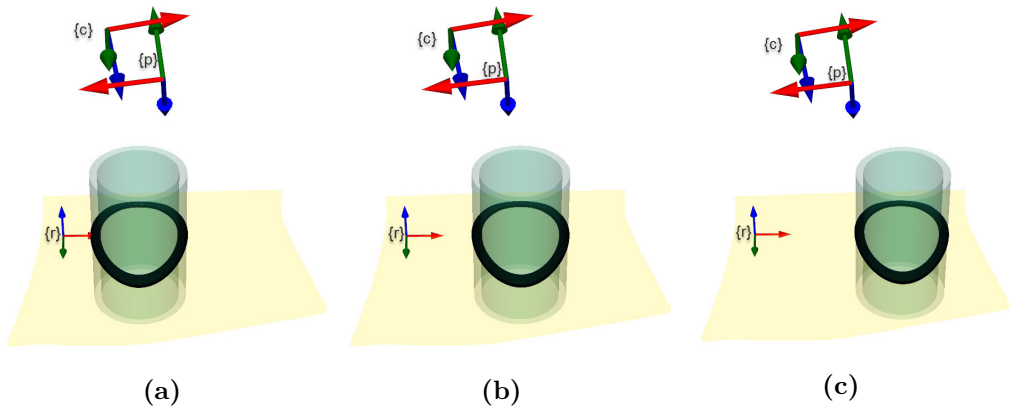


Figure 6.11.: Stub Configuration Pose Experiment 1

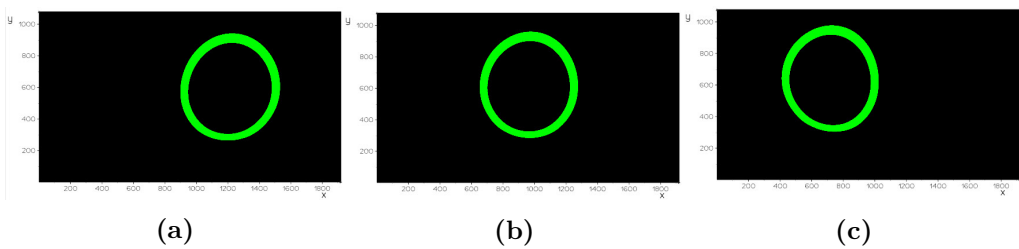
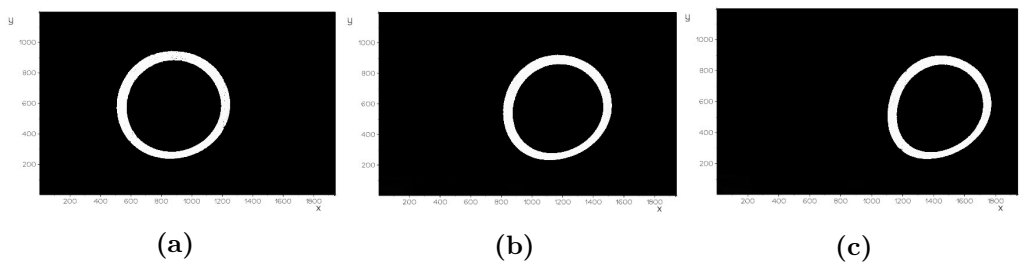


Figure 6.12.: Projected projector image

Figure 6.13.: Threshold image of coordinates in  $\{c\}$

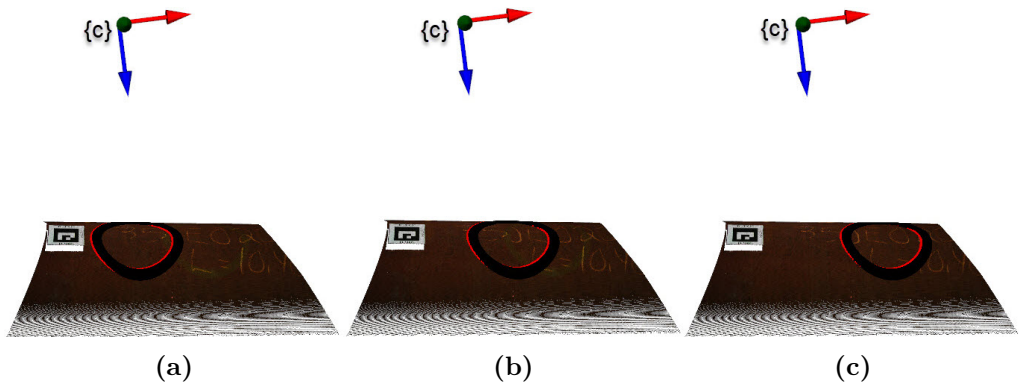


Figure 6.14.: Difference in predicted  $A_g$  and projected  $A_{gap}$  Pose 1

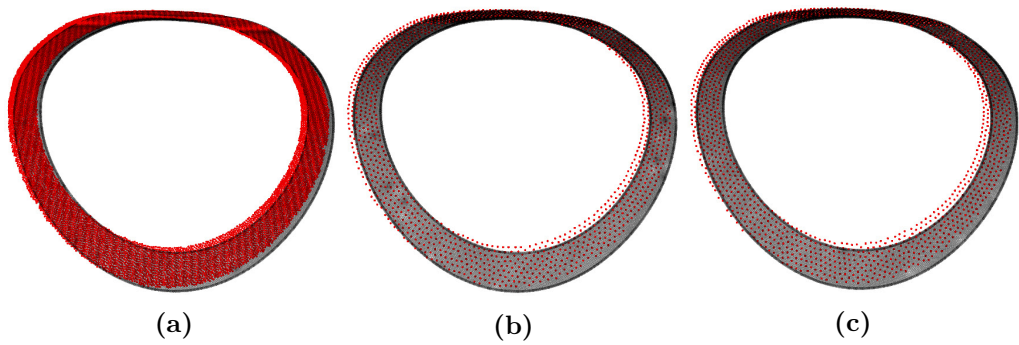


Figure 6.15.: Detail view predicted  $A_g$  and projected  $A_{gap}$

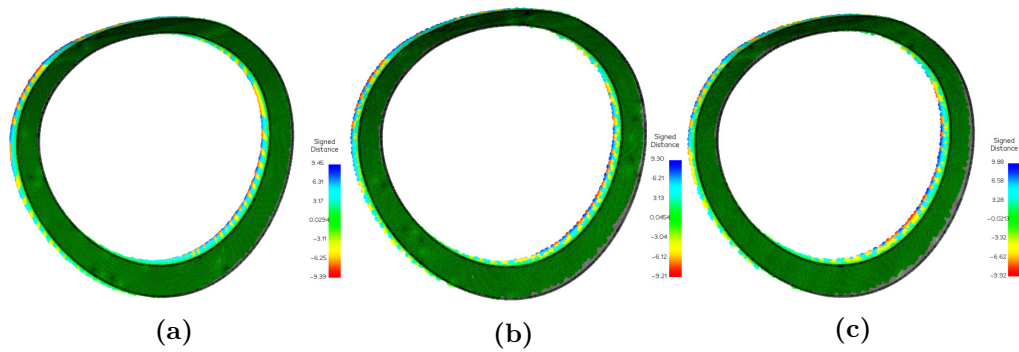


Figure 6.16.: Signed distance between  $A_g$  and projected  $A_{gap}$

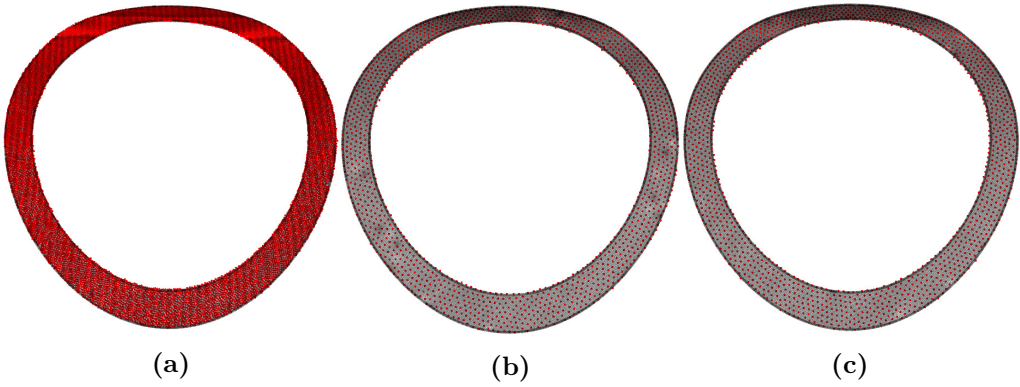


Figure 6.17.:  $A_{gap}$  aligned with ICP to  $A_g$

Table 6.4.: Transformation for aligning  $A_{gap}$  to  $A_g$  in form of euler angles and translation.

<b>C</b>	$\{r\}_x^\circ$	$\{r\}_y^\circ$	$\{r\}_z^\circ$	<b>x</b> (mm)	<b>y</b> (mm)	<b>z</b> (mm)	<b>Norm(x,y,z)</b> (mm)
a	0.41	0.01	0.04	3.79	6.70	1.18	7.79
b	0.60	0.02	0.11	4.03	9.69	1.45	10.60
c	0.702	0.03	0.14	4.13	11.12	1.64	11.98

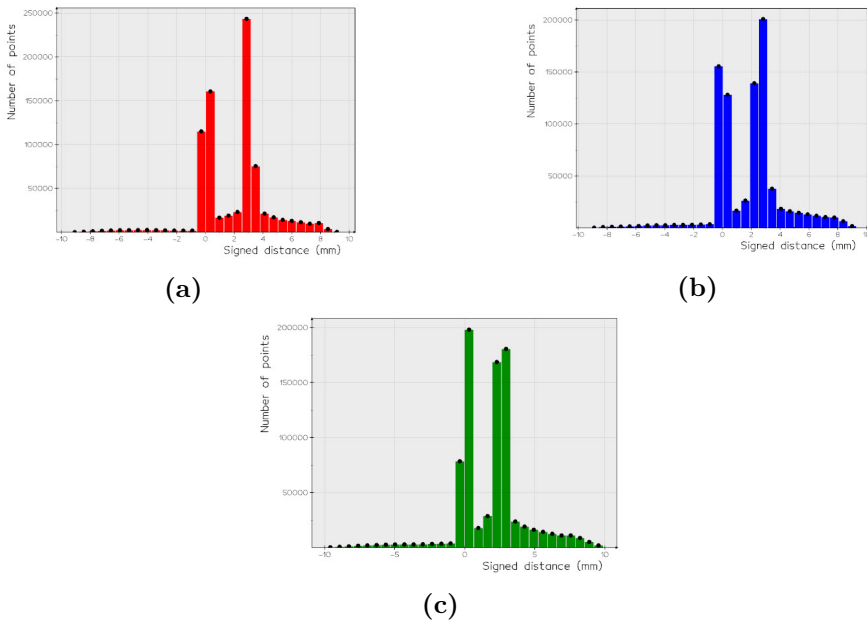


Figure 6.18.: Distribution of signed distance between  $A_{gap}$  and  $A_g$



Experiment 2

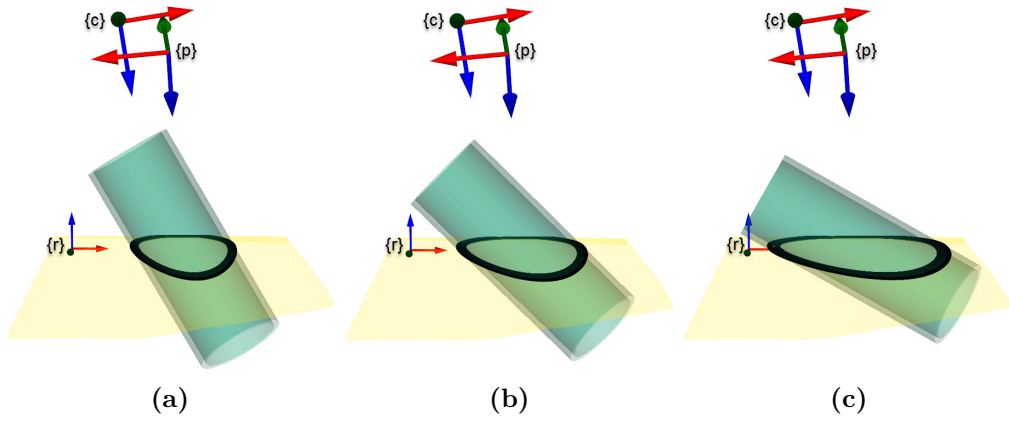


Figure 6.19.: Stub Configuration Pose Experiment 1

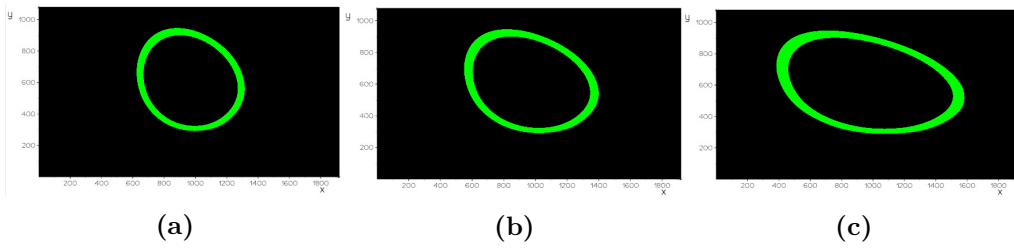


Figure 6.20.: Projected projector image

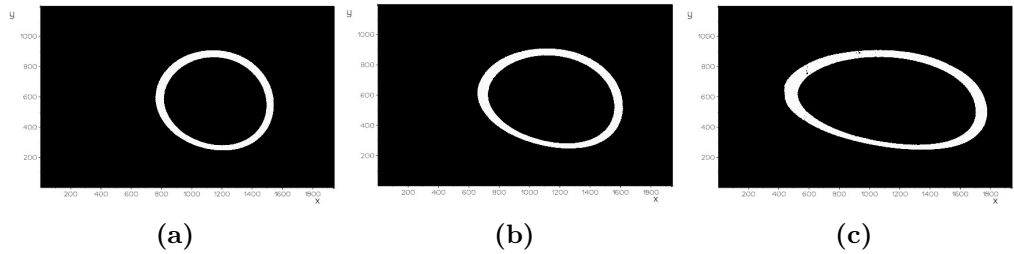


Figure 6.21.: Threshold image of coordinates in  $\{c\}$

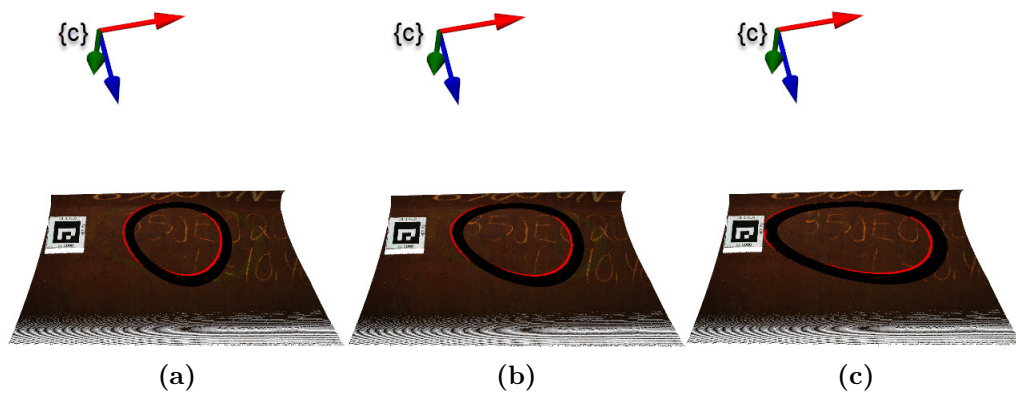


Figure 6.22.: Difference in predicted  $A_g$  and projected  $A_{gap}$  Pose 1

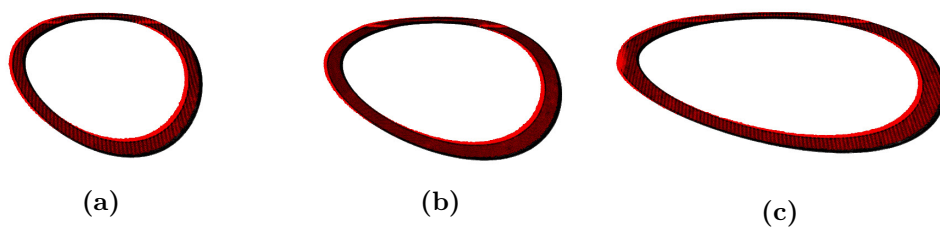


Figure 6.23.: Detail view predicted  $A_g$  and projected  $A_{gap}$

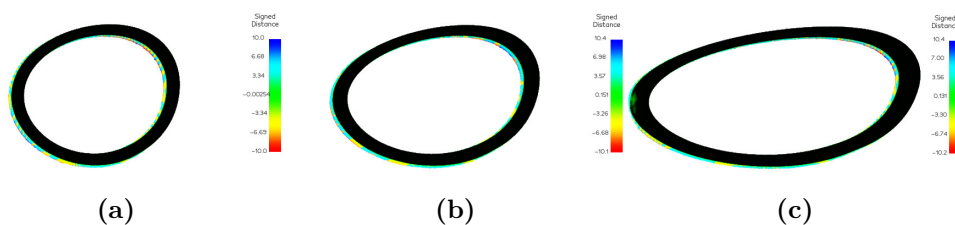


Figure 6.24.: Signed distance between  $A_g$  and projected  $A_{gap}$

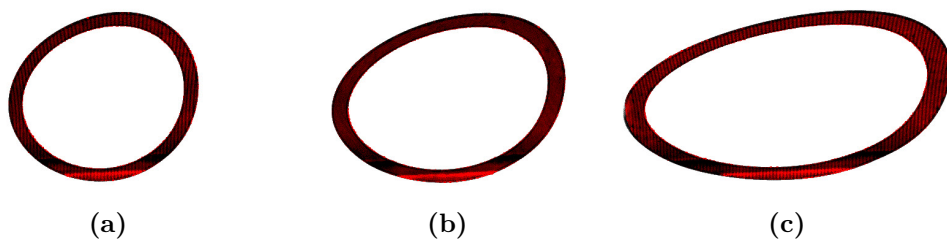
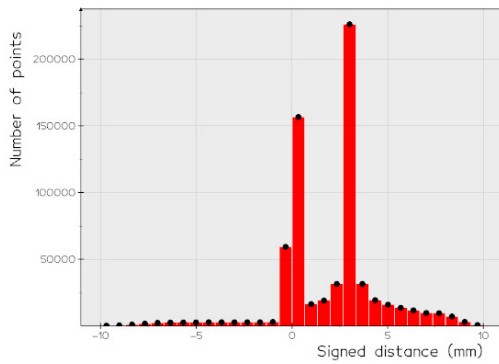


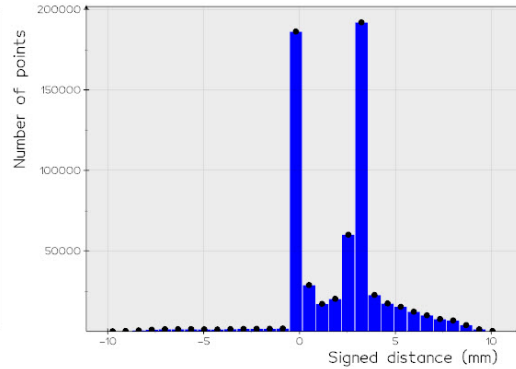
Figure 6.25.:  $A_{gap}$  aligned with ICP to  $A_g$

**Table 6.5.:** Transformation for aligning  $A_{gap}$  to  $A_g$  in form of euler angles and translation.

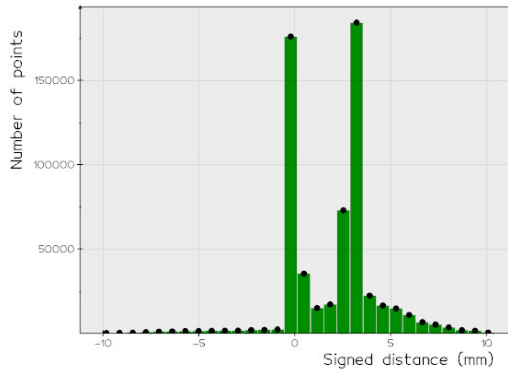
<b>C</b>	$\{r\}_x^\circ$	$\{r\}_y^\circ$	$\{r\}_z^\circ$	<b>x (mm)</b>	<b>y (mm)</b>	<b>z (mm)</b>	<b>Norm(x,y,z) (mm)</b>
a	0.53	0.03	0.05	4.03	8.70	1.45	9.70
b	0.48	0.00	0.15	4.51	7.76	1.39	9.01
c	0.53	0.01	0.12	4.36	8.51	1.46	9.67



(a)  $A_{gap}$  to  $A_g$



(b)  $A_{gap}$  to  $A_g$



(c)  $A_{gap}$  to  $A_g$

**Figure 6.26.:** Distribution of signed distance between  $A_{gap}$  and  $A_g$

## Experiment 3

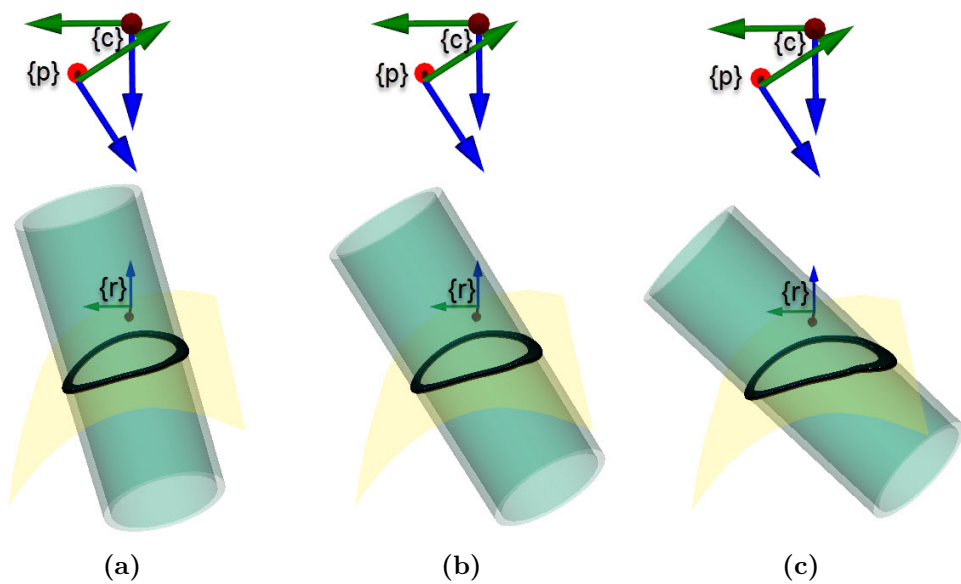


Figure 6.27.: Stub Configuration Pose 1 Experiment 3

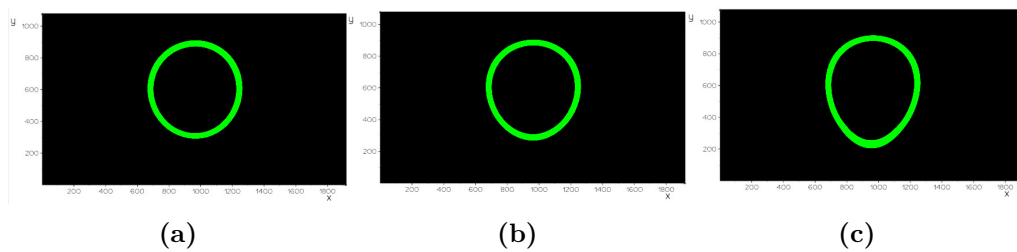
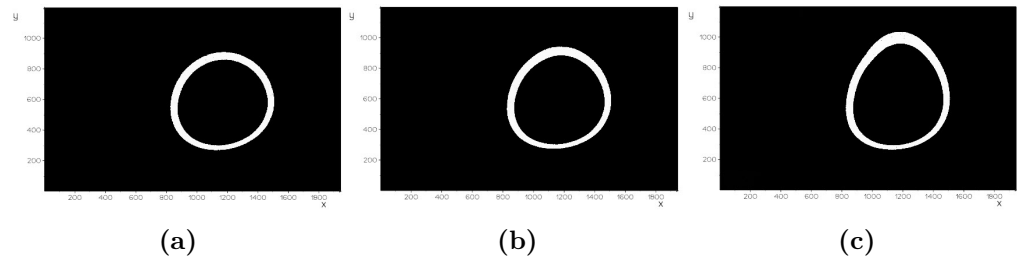
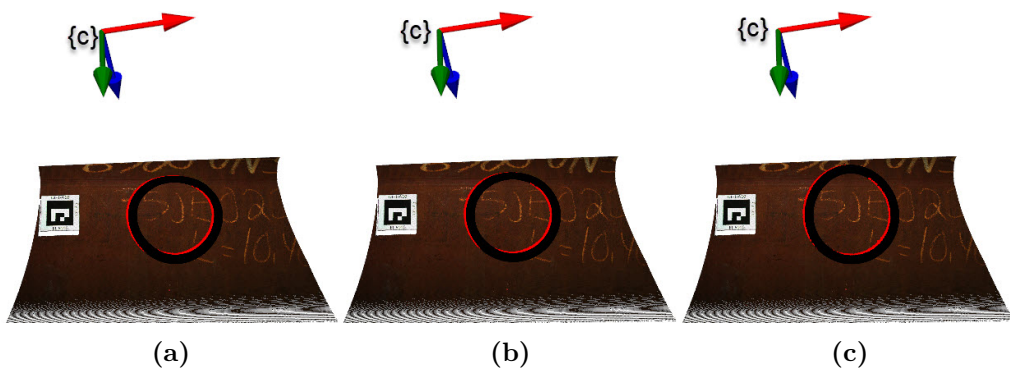
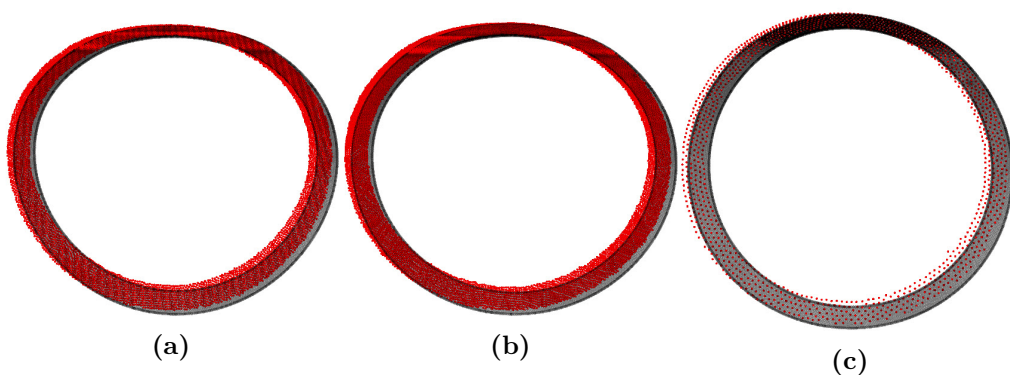


Figure 6.28.: Projected projector image

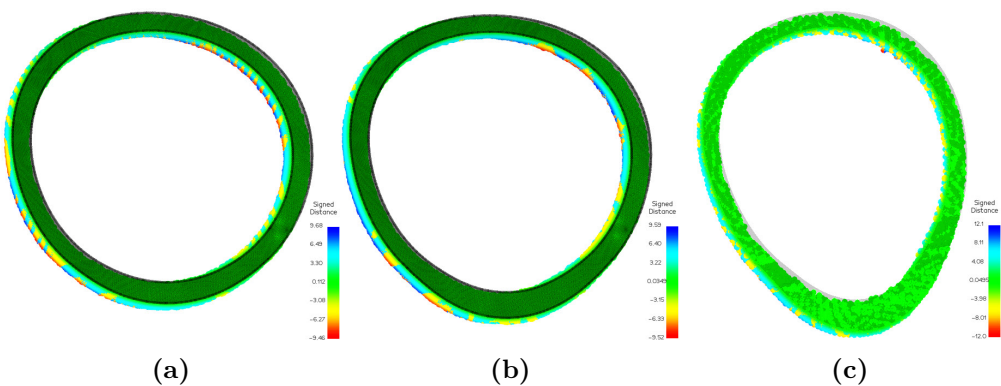
Figure 6.29.: Threshold image of coordinates in  $\{c\}$



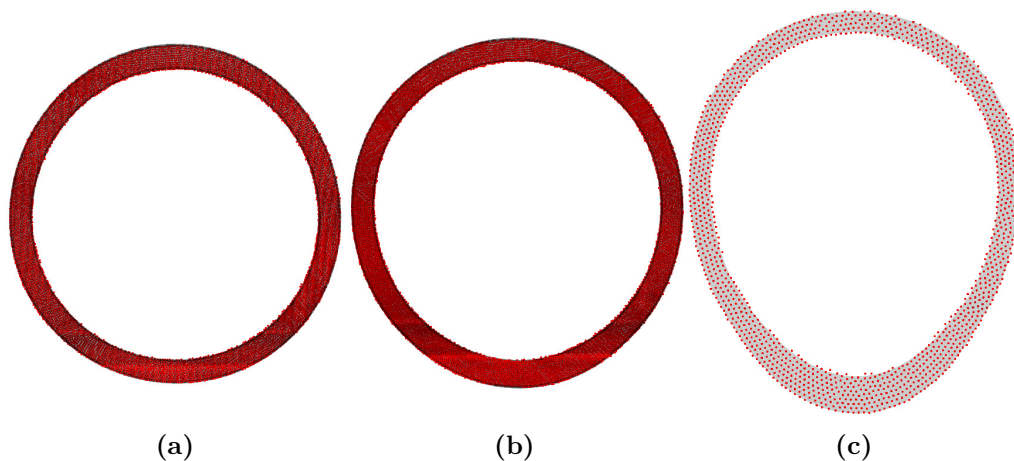
**Figure 6.30.:** Difference in predicted  $A_g$  and projected  $A_{gap}$  Pose 1



**Figure 6.31.:** Detail view predicted  $A_g$  and projected  $A_{gap}$



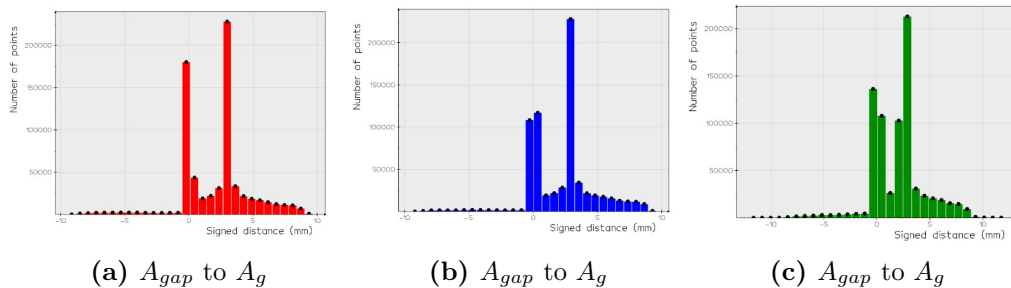
**Figure 6.32.:** Signed distance between  $A_g$  and projected  $A_{gap}$



**Figure 6.33.:**  $A_{gap}$  aligned with ICP to  $A_g$

**Table 6.6.:** Transformation for aligning  $A_{gap}$  to  $A_g$  in form of euler angles and translation.

$C$	$\{r\}_x^\circ$	$\{r\}_y^\circ$	$\{r\}_z^\circ$	$\mathbf{x}$ (mm)	$\mathbf{y}$ (mm)	$\mathbf{z}$ (mm)	$\mathbf{Norm}(\mathbf{x},\mathbf{y},\mathbf{z})$ (mm)
1	0.53	0.03	0.04	3.87	8.61	1.36	9.54
2	0.56	0.04	0.07	3.81	8.87	1.35	9.75
3	0.61	0.02	0.14	4.12	9.82	1.49	10.76



**Figure 6.34.:** Distribution of signed distance between  $A_{gap}$  and  $A_g$

6.2.2. Pose 2

Table 6.7.: Pose 2 Experiment

Pose 2							
Exp.	Stub OD	Stub ID	t	Config	Trans $\{r\}_x$	Rot $\{r\}_y$	Rot $\{r\}_x$
1	240 mm	210 mm	15 mm	a, Figure 6.35a	200 mm	0	0
				b, Figure 6.35b	300 mm	0	0
				c, Figure 6.35c	400 mm	0	0
2	430 mm	400 mm	15 mm	a, Figure 6.43a	325 mm	0	0
				b, Figure 6.43a	325 mm	0	0
				c, Figure 6.43a	325 mm	0	0

Experiment 1

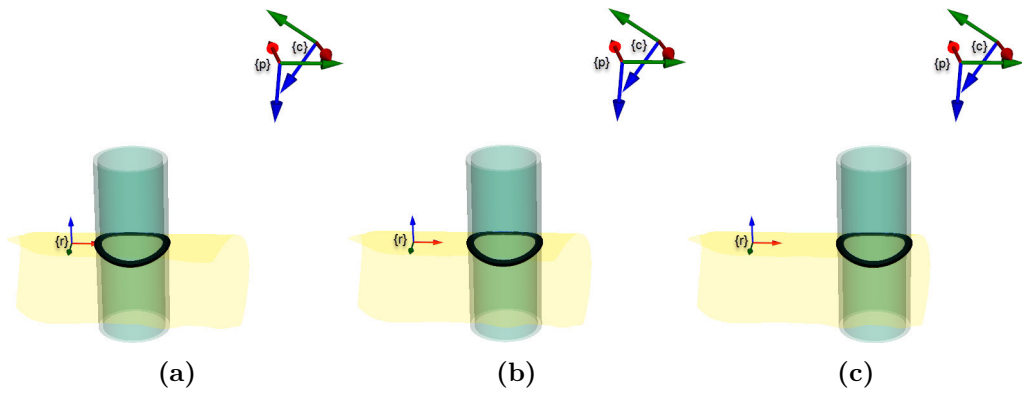


Figure 6.35.: Stub Configuration Pose 2 Experiment 1

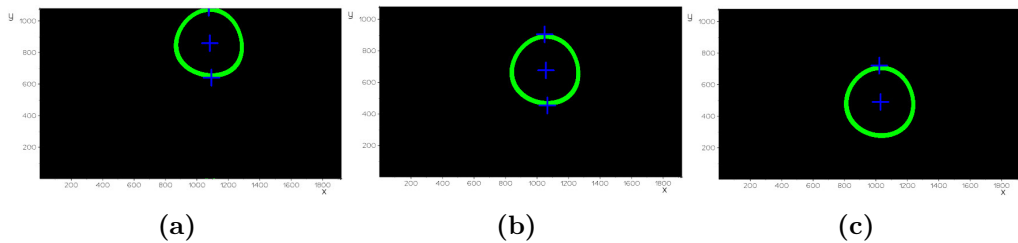


Figure 6.36.: Projected projector image

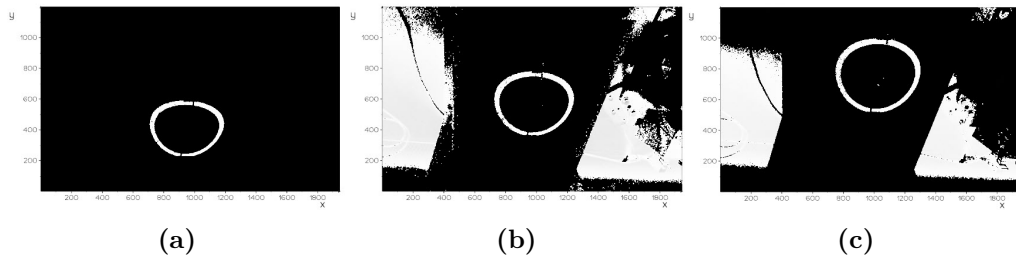


Figure 6.37.: Threshold image of coordinates in  $\{c\}$

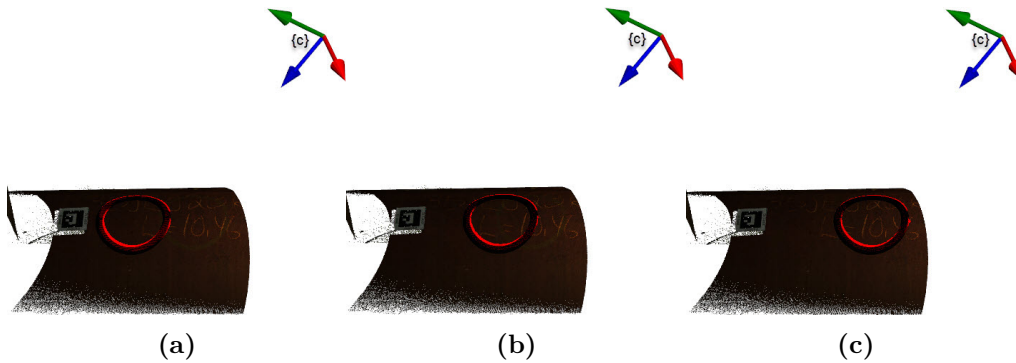


Figure 6.38.: Difference in predicted  $A_g$  and projected  $A_{gap}$  Pose 1

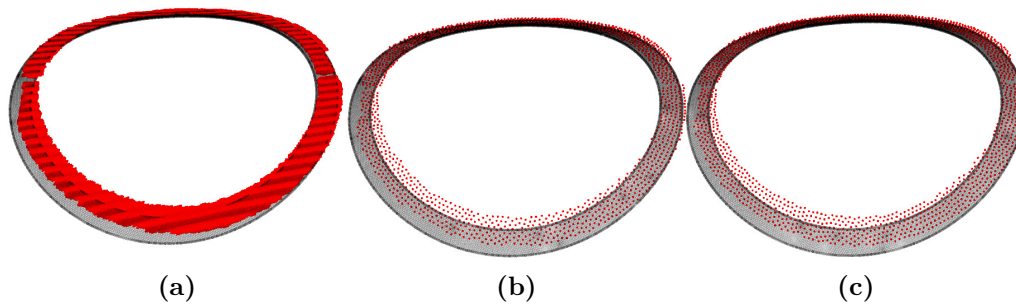


Figure 6.39.: Detail view predicted  $A_g$  and projected  $A_{gap}$



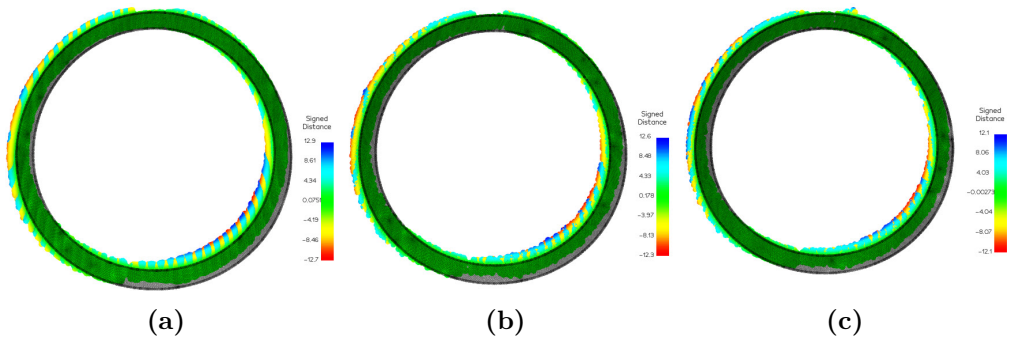


Figure 6.40.: Signed distance between  $A_g$  and projected  $A_{gap}$

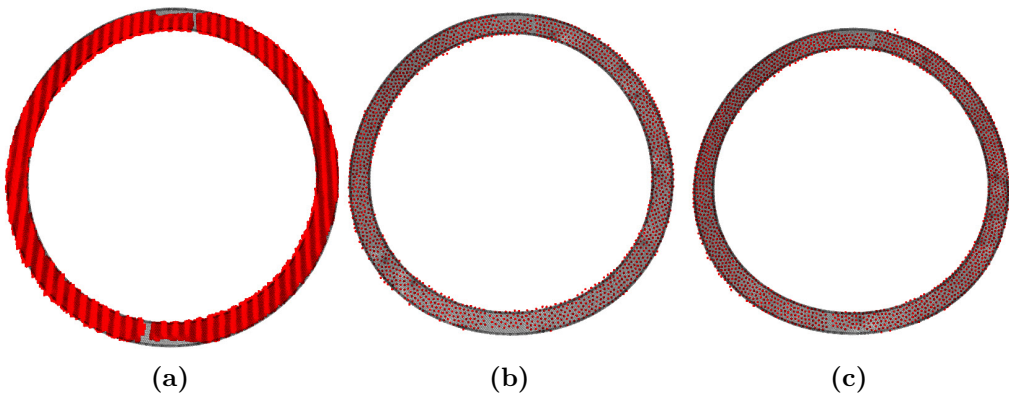


Figure 6.41.:  $A_{gap}$  aligned with ICP to  $A_g$

Table 6.8.: Transformation for aligning  $A_{gap}$  to  $A_g$  in form of euler angles and translation.

C	$\{r\}_x^\circ$	$\{r\}_y^\circ$	$\{r\}_z^\circ$	$\mathbf{x}$ (mm)	$\mathbf{y}$ (mm)	$\mathbf{z}$ (mm)	Norm(x,y,z) (mm)
1	0.23	-1.07	-0.64	21.02	6.74	2.52	22.22
2	0.26	-1.13	-0.77	22.48	6.96	2.5	23.67
3	0.25	-1.09	-0.75	21.74	6.95	2.57	22.97

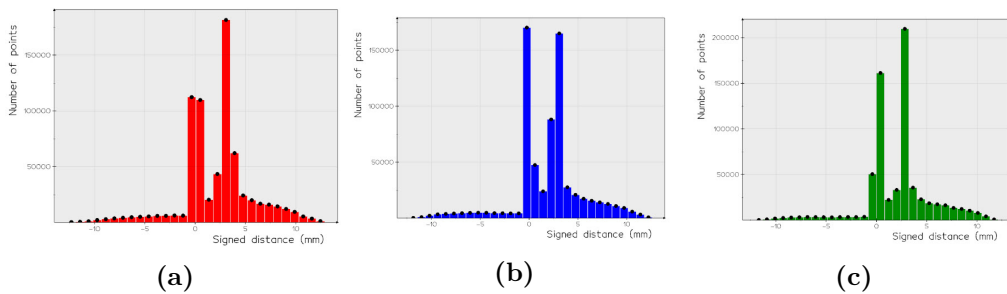


Figure 6.42.: Distribution of signed distance between  $A_{gap}$  and  $A_g$

Experiment 2

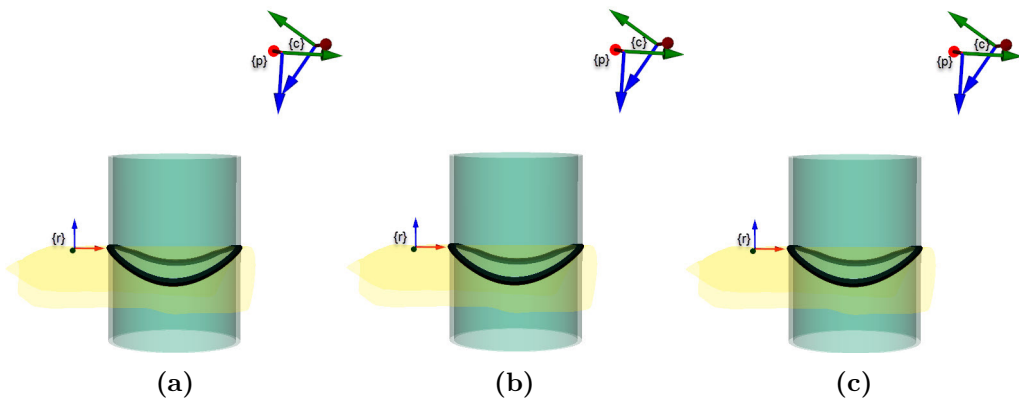


Figure 6.43.: Stub Configuration Pose 2 Experiment 1

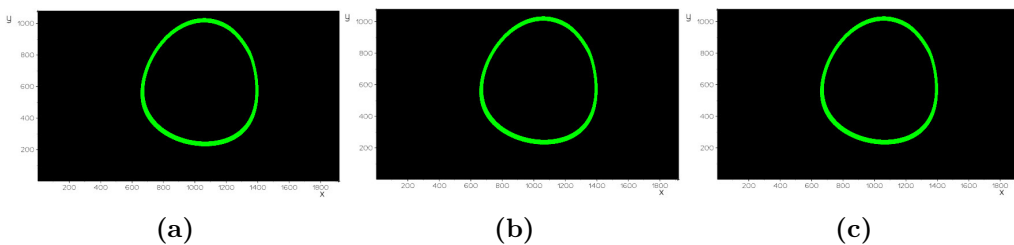


Figure 6.44.: Projected projector image

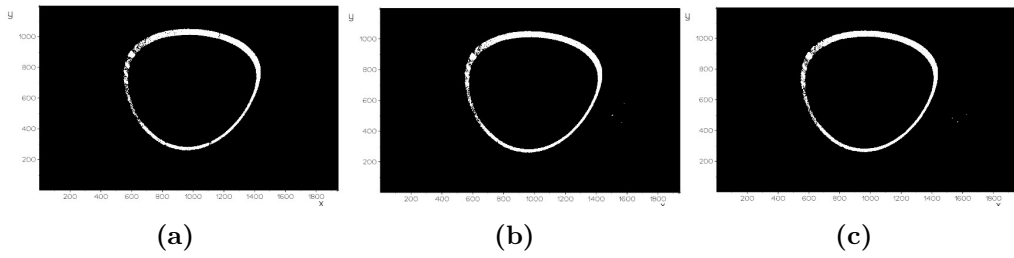


Figure 6.45.: Threshold image of coordinates in  $\{c\}$

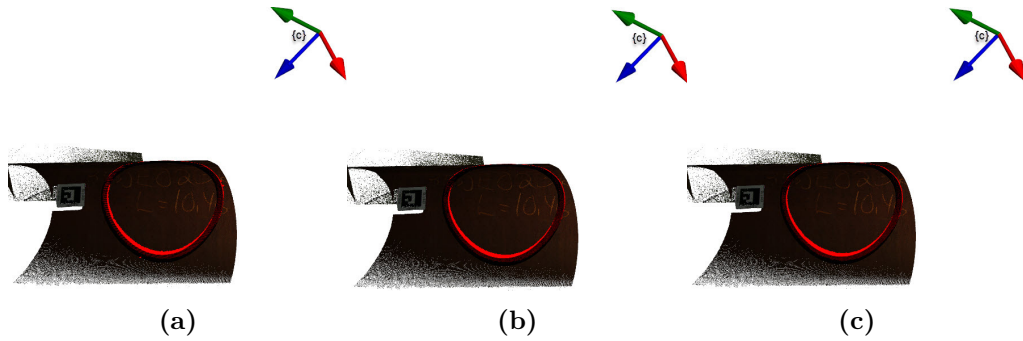


Figure 6.46.: Difference in predicted  $A_g$  and projected  $A_{gap}$  Pose 1

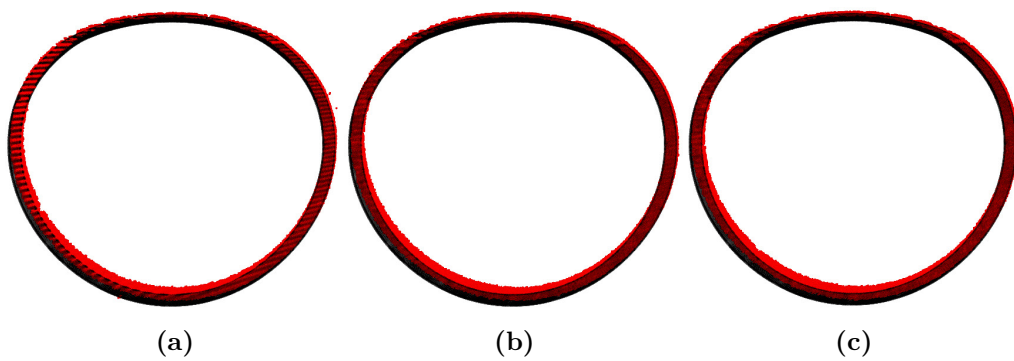


Figure 6.47.: Detail view predicted  $A_g$  and projected  $A_{gap}$

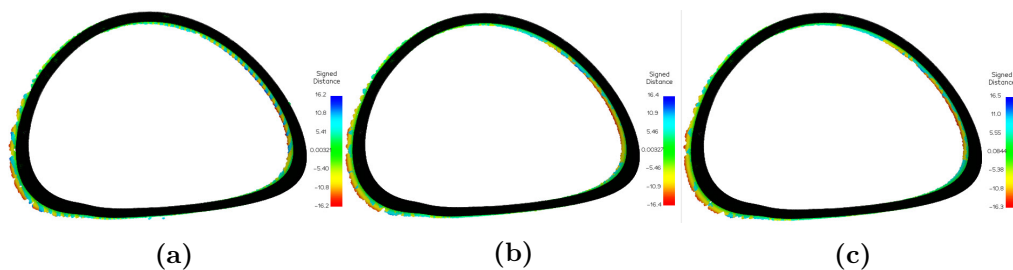


Figure 6.48.: Signed distance between  $A_g$  and projected  $A_{gap}$

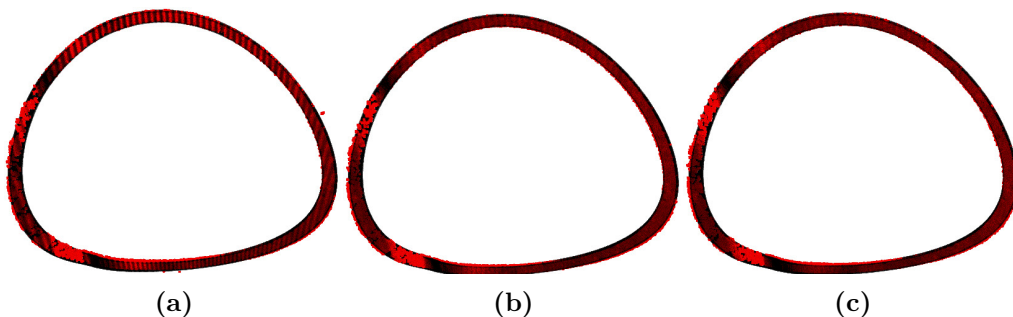


Figure 6.49.:  $A_{gap}$  aligned with ICP to  $A_g$

Table 6.9.: Transformation for aligning  $A_{gap}$  to  $A_g$  in form of euler angles and translation.

C	$\{r\}_x^\circ$	$\{r\}_y^\circ$	$\{r\}_z^\circ$	$\mathbf{x}$ (mm)	$\mathbf{y}$ (mm)	$\mathbf{z}$ (mm)	Norm(x,y,z) (mm)
1	0.31	-1.44	-0.96	28.97	8.06	2.97	30.22
2	0.30	-1.37	-0.90	27.80	6.97	2.30	28.76
3	0.32	-1.38	-1.00	28.05	7.79	2.64	29.23

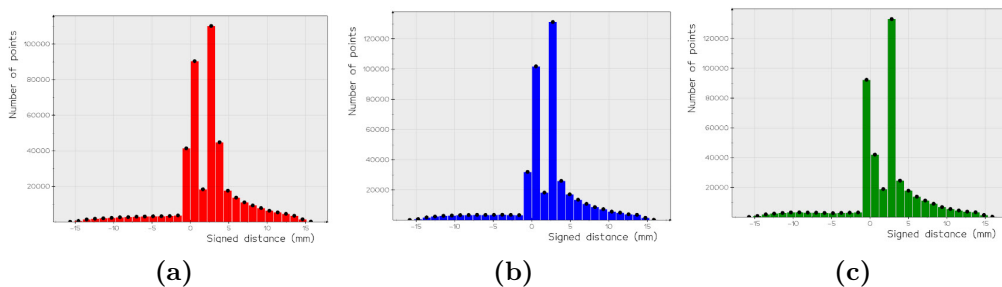


Figure 6.50.: Distribution of signed distance between  $A_{gap}$  and  $A_g$

# Chapter 7.

## Discussion

This chapter investigates the implemented pipeline within possibilities, limitations, and development compared to today's stub assembly procedure. The experiment results are discussed and compared to NORSOK M-101 [3].

### 7.1. Pipeline

When the pipeline were implemented it was a lot of trial, error and research for creating a such system. This section discuss the choice of software, features and algorithms for the implemented pipeline. To have a clear overview of the pipeline it is gone through as in [chapter 4](#).

1. Capture point clouds
2. Point cloud processing
3. Mesh processing
4. Image projection

#### 7.1.1. Capture point clouds

##### **Blender**

The point clouds are either captured through the Zivid two camera or a synthetic 3D camera in Blender. In the synthetic system, you can say that the hardware in the system is through the Blender API. At the start of the master thesis, it was used some time to find out how to create realistic point clouds without being in the lab. Because the lab environment wasn't ready yet, and some of the hardware to the system was missing. Therefore, Blender was looked into as a platform for establishing this.

Blender is an open-source platform that many integrated functions for working with point clouds, CAD, mesh, and computer vision applications. Also, it has a high focus within rendering, cameras, and an integrated python API that makes it possible to extract and automate the process through scripting. In addition to Blender, other open-source platforms as Unity was researched. Unity also has the possibility to render through a camera and creating a projector as a light source. However, Unity is more a game engine than Blender that is designed for modeling..

Since the capturing object in the scene for this application is a cylindrical shape, it was investigated into creating a cylindrical point cloud manually in python. This can be a great tool for testing different cylinder fitting algorithms. However, modeling this point cloud "as" it was captured with a 3D camera. Therefore, further work was done with capturing point clouds through Blender. Before it was discovered that the Z-data of an image could be extracted with an RGB image, a stereo system was created in Blender to make a synthetic point cloud through triangulation. Creating a point cloud as a stereo system wasn't so trivial since it required much tuning for getting a usable point cloud. Here it was seen that the result wasn't good enough compared to a captured point cloud from a Zivid Two camera.

The whole time the goal was to create a synthetic system, within mind that the system should work as it was for a real system. Blender is an independent platform with an integrated Python API. Working with Blender through another Python environment can be difficult. To run functions in Blender through scripting, the script needs to be run through Blender's GUI. This is inappropriate when working with your own python environment with other package dependencies. You either need to change your environment to be in Blender or extract the required data through Blender by running it in the background. Here there a workaround. You can run a Blender python script through a terminal in the background. However, in this project, the focus has been to use Blender to capture data and not for data processing. So one isn't dependent on having Blender in the lab system.

After an image in Blender is rendered, then the file is exported as a .exr file and imported into the Python environment with the OpenEXR package. This package seems not to be maintained anymore since the package is from Python 2.5+ and upwards. After much trying, the package was installed. OpenCV also can read .exr files, but a problem was figuring out how to split the channels in the .exr format and using the OpenEXR package. In other words, a weakness in the synthetic system. When the .exr file is imported, and 5-Channel format is split, and the RGB and Z-data are extracted. Then a colored point cloud can be created with the Blender camera intrinsic. This is done with functions from Open3D and was faster than running the same functions through Numpy. The

disadvantage of using the Open3D function is that the point cloud doesn't have 1:1 correspondence with the points in 3D with pixel coordinates as in Zivid Two point cloud. Despite this, capturing point clouds in Blender allows fast capture of a realistic point cloud of a defined object in a scene. Parameters such as in [section 4.1.1](#) can easily be tuned to capture different point clouds, and this has been an advantage in this project.

## Zivid

The Zivid Two is the core of the lab system, and without an accurate point cloud, it wouldn't produce any good results. Capturing high precision point clouds within a range of a working distance of  $1400mm$  is the Zivid two camera's strength. And within this range, other 3D-cameras as RealSense cameras, Zed are of much worse accuracy. In the datasheet, [section E.1](#), the Zivid Two are compared to another stereo vision camera, and you can see that their accuracy is much higher than the comparable camera.

The Zivid Two properties within focus distance and FOV were highly dependent when picking the Asus Predator Z650 projector. Finding a projector that matches the FOV of the Zivid two camera was a challenge since a normal desk projector's purpose is designed to light up a canvas from a long distance. Therefore, using a short-throw projector produces a large image within a short throwing distance. The projector overlapped well with the Zivid and the projector after adjusting it with a 3D-printed bracket. The Kuka robot also made it much easier to capture with different poses, and the possibility for testing with a rigid setup is an advantage. Since the laying leg tube weighs approximately 150 kg, then moving the setup with the Kuka robot was very practical.

When capturing point clouds in the system for a lab, the settings were adjusted in Zivid studio with the assisted mode in Zivid studio, and a .yaml file was exported to be imported into python when the image should be captured through the Zivid API. This made it easy to capture fast through python.

### 7.1.2. Point cloud processing

After a point cloud is captured, the data need to be processed for further use in the system. This can either be that point cloud need to be downsampled or be cleaned from noise. In this system, downsampling and cleaning have been done through the libraries Open3d and Vedo, with majority of use of Vedo, because of the Vedo mesh intersecting function as argued for in [subsection 7.1.3](#).

Downsampling through voxels is fast and easy to compress a point cloud to a wanted size of points. However, it isn't always so easy to know how many points

are necessary to create a mesh or find the best cylinder of fit. A higher number of points can give a better accuracy but of the cost more computation time. This hasn't been evaluated since this thesis aims to create a method and doesn't focus on details as computation time. The system isn't created for working in real-time, but fast computation time is always an advantage and can be listed in further work.

Outliers and noise are deleted from the point cloud with radius outlier removal and by a set Z-component relative to  $\{c\}$ . Deleting the noise by a threshold value of the Z-component could be argued not to be the best way since changing the pose to leg in the scene, the camera's Z-distance to the leg change relative to the pose. A better way here could be to have a pure radius removal, but radius removal depends on the resolution of points, so it can be hard to set a suitable threshold value. Another way could be to clean noise within a distance from the centroid of the point cloud. However, this depends on the dimension of the leg in the scene. If the leg's dimension is greater than the camera's FOV, then all points would mostly be on the leg's surface, and there wouldn't be any outliers.

The reference system in [subsection 4.3.3](#) is based on a cylinder fitting for finding a cylinder axis. The fitting is done with a least-square fitting through the paper [30]. As stated in the paper, this fitting isn't the most robust cylinder fitting algorithm. Other algorithms as the cylinder RANSAC [31] algorithm were researched and tried, but the cylinder RANSAC algorithm seems to more unstable for this case. As explained in the [subsection 7.2.1](#), this requirement isn't so trivial to evaluate today's stub assembly procedure and therefore hasn't been the main focus for testing different cylinder fitting algorithms. For further development and integration into today's procedure, research can be done in this field.

The reference system is created in combination with the cylinder fitting and an Aruco marker. The cylinder fitting defines the axis along the leg, and the Aruco marker point defines the origin of the  $\{r\}$ -frame. The two other axes in the coordinate system are defined from the cylinder fitting by taking the perpendicular vector and the cross product. Therefore, evaluating that the  $\{r\}$  match today's assembly procedure world coordinate system is challenging. This can be crucial since if the axes don't match, both the rotation and translation could be wrong, and the projection isn't there it should be.

### 7.1.3. Mesh processing

The cleaned and downsampled point cloud can now be created as a mesh from Delaney triangulation. When working with meshes, the essential thing around the whole method is that one is dependent on an algorithm that intersects two meshes and returns a path as a line. It was looked into several libraries for mesh



intersections. However, there aren't so many Open-source libraries that support that. As in the paper *Mesh-based tool path calculations for tubular joints* [4], the mesh algorithm was used through Blender. This method used a lot of energy to find a library that intersects two mesh without too many dependencies. When working through python, this system goal is to create an industrial system that can be further used in Aker Solutions procedures. And it isn't beneficial to have dependencies on third part software like Blender. It is looked into libraries such as Vedo [25], Pyvista [32] and Trimesh [33]. Pyvista and Vedo depend on VTK [22], and most of their functions are written through VTK. And VTK is developed from Kitware, which is a big reliable firm. The advantage of Vedo it is only two dependencies, and this is VTK and Numpy. Also, the Vedo is very well maintained by Marco Musy, and if you have any issues, the response time is within hours to a day. Also, Vedo, through VTK, has a great Plotter for visualization of point clouds, mesh, points, and lines. An advantage is that you can have a plotter with more than one window, making it easier to capture figures with the same view. Another thing is that if you don't want to be dependent on the VTK functions through Vedo, you can import the functions and write them by yourself. Also, if Vedo misses any functions Pyvista or Trimesh has, you can use functions from there since Pyvista are Numpy and VTK, and Trimesh are pure Numpy with no other dependencies.

#### 7.1.4. Image projection

As described in [subsection 4.5.1](#), calibration needs to be done to move the intersection coordinates from the  $\{c\}$ -frame to  $\{p\}$ -frame with a stereo calibration,  $T_{cp}$ , and an intrinsic calibration  $P_i$  to create an image of these coordinates. In the specialization project, a lot of time was spent researching calibration methods for calibrating a projector for both stereo and intrinsic. This established method is described in [Appendix B](#). This calibration is flexible since it is not required to have a printed checkerboard since the checkerboard is projected through the projector. Thus, the 3D points of the checkerboard corners are defined by the Zivid two point cloud. Defining the Zivid two's point cloud points can take calibration error from Zivid's camera into this calibration. However, a calibration can be done fast if you need to change the focus of the projector. When the calibration was done, also, another method was tried, *Simple, Accurate, and Robust Projector-Camera Calibration* [34] was tried in pair with the other calibration. However, the light conditions in the robot cell was a challenge for both calibration since it wasn't possible to turn off the light in this cell. The Mesh brown calibration gave an extraordinary result that was long from the other calibration method. Therefore, the calibration method from [Appendix B](#) was used.

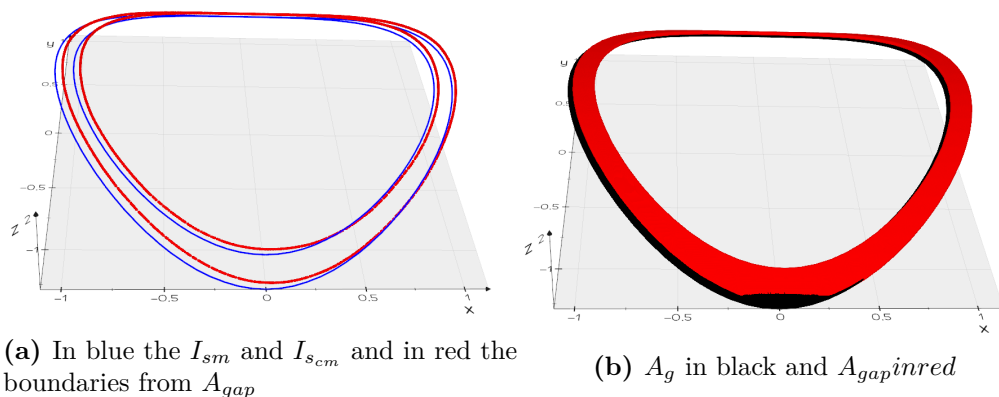
After the calibration, the result is obtained, and coordinates of the intersection

lines are created and a contour image. The pixels area for the  $A_{gap}$  is defined by the `cv2.fillPoly` function that fills all pixels within a contour. This is fast and was easy to use since OpenCV is already used in the project. Other ways to do this could be to only intersecting one cylinder instead of two with the leg-mesh and thicken the line by all tangents of each line that defines the curve. However, this could take more computation time to do, and you need to determine the thickness of  $A_{gap}$  in pixels in the projector image.

### 7.1.5. Evaluation

As stated in the next [section 7.2](#), the evaluation of the conducted experiment in [chapter 6](#) are only evaluated to the transforming requirement in [subsection 7.2.3](#) since an actual stub section cut aren't available to compare against a projected cut.

When settling for an evaluation method for finding the deviation between  $A_g$  and  $A_{gap}$ , it was researched into several methods. In addition to ICP it was investigated into compare the boundaries of  $A_{gap}$  against  $I_{sm}$  and  $I_{scm}$  as in the [Figure 7.1](#).

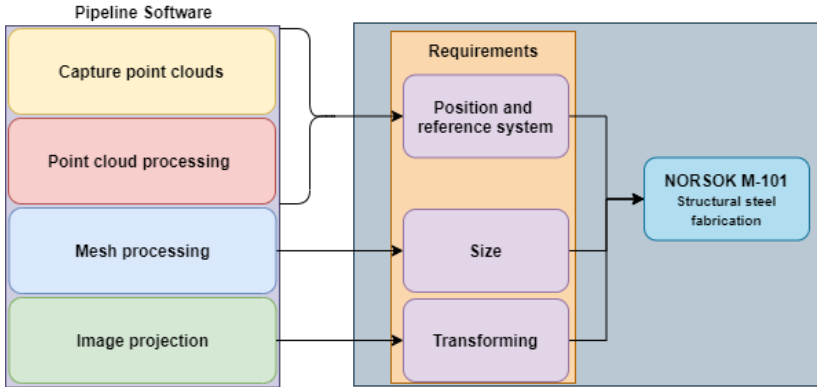


**Figure 7.1.:** Boundaries compared to intersection lines

In some cases, it was difficult to extract the boundaries of the  $A_{gap}$  because of the noise in the point cloud. Therefore, ICP was used as a more general solution for finding the transformation between  $A_g$  and  $A_{gap}$ . However, if there is a lot of noise in the image in [Figure 4.31d](#), and if some points don't get removed from outlier removal than it can disturb the result.

## 7.2. Requirements

In order to compare the system against the today's procedure, one can discuss three system requirements that need to be met to satisfy the defined tolerances in NORSOK-M101 [3]:



**Figure 7.2.:** System requirements against NORSOK-M101 [3]

1. **Position and reference system:** The system depends of a reference system,  $\{r\}$ , defined in a reference point, so coordinates can be specified in the right transformation compared to the real world coordinate system,  $\{w\}$ .
2. **Size:** The defined  $A_g$  must match the size of a real stub section cut.
3. **Transformation:** The transformation of  $A_g$  to  $A_{gap}$  must be accurate enough to meet the tolerances in NORSOK-M101 [3].

### 7.2.1. Position and reference system

In terms of position and reference system, there are some dependencies in the system that define whether the  $\{r\}$  match the  $\{w\}$ . The  $\{r\}$  is defined by the cylinder fitting algorithm to create a 3-axes coordinate system. The difficult thing here to know is if the coordinate axes match the measurement axes used in today's stub assembly procedure. Whether the  $\{r\}_z$  would match the  $\{w\}_z$  is hard to measure since the  $\{c\}_z$  orientation deviates from the  $\{w\}_z$ . The position of the  $\{r\}$  is defined by Aruco marker recognition in the Zivid image plane. The Aruco point needs to match today's stub assembly procedure reference point. Let's say the  $\{r\}$  were defined perfectly in today's assembly procedure reference point. The accuracy would be so accurate as of the defined accuracy that the Zivid Two camera has. However, today's measurement is also so precise as the measurement method that is used.

### 7.2.2. Size

The  $A_g$  is defined by mesh intersection of the leg mesh,  $L_m$ , and the defined synthetic cylinders,  $S_m$  and  $S_{cm}$ . These synthetic cylinders can deviate from the real stub since the real stub would mostly have deviation within in the listed [Appendix A](#). So if the real stub differs highly from a nominal synthetic cylinder, the  $A_g$  doesn't match the actual stub section cut. A solution for this could be to scale the projector image to overlap with the actual cross-section cut area.

### 7.2.3. Transforming

The position and reference system and size are set and implemented to project a  $A_g$  onto the leg's surface. One thing is researching other methods to create a reference system and a reference point; however, evaluating this is not easy when you don't have a cut stub cross-section to compare against the projected cut. On the other hand, researching multiple cylinder fitting algorithms with different methods to detect a reference point would strengthen the position. These aren't evaluated in the experiment since it hasn't been prioritized when creating proof of concept. However, this evaluation is categorized into further work of the thesis.

When the defined intersection lines,  $I_{sm}$  and  $I_{scm}$  are defined in terms of the system  $\{r\}$ , the coordinates need to be transformed from  $\{c\}$  to  $\{p\}$  with a  $T_{cp}$  obtained from a stereo calibration. After the coordinates of  $I_{sm}$  and  $I_{scm}$  are transformed to  $\{p\}$ , the correspondence 3D to pixels coordinates can be created with the  $P_i$  and be corrected for distortion  $d_p$ . When transforming and creating an image from the calibration result, it can be errors that make the coordinates not correlate with the coordinates before projection in  $\{c\}$ .

## 7.3. Experiment

This section discusses the experiment results for synthetic and lab results within the plotted results against the evaluation method in [section 4.6](#). The results are evaluated against the transformation requirements, which are explained in the last section.

### Synthetic

Looking at the synthetic experiment, you see in the [Figure 6.4](#) that it seems that the  $A_{gap}$  match with  $A_g$ , that all points lay within the same area. As explained in the system, the default unit that Blender operates within is in meters. However, looking at the transformation for matching  $A_{gap}$  to  $A_g$  gives a considerable error

within the translation to be a synthetic system. What the error comes from can be multiple things such as the extracted intrinsic parameters of the camera, or how the projector is modeled to find the projector intrinsic parameters, an error in the transformation between the  $\{c\}$  and  $\{p\}$ . When looking at the different properties in the scene, as the modeled transformation between  $\{c\}$  and  $\{p\}$ , the extracted  $T_{cp}$  gives an error of  $-1$  in terms of rotation, and a  $0.5mm$  in terms of translation.

## Lab

In the lab experiment, it is tested with two different poses relative to the laying scene. Looking at [Table 6.4](#) in pose 1, the pose gives a result comparable to the tolerances in the appendix. Positional tolerances with a round tube that are  $d_n < 35000mm$  are  $6mm$  and  $d_n > 3500$  is set to  $10mm$ . However, these are positional tolerances and not for grinding, which is the goal here. So for pose 1, the result for experiment 1-3, a deviation within a norm of a vector of a range from  $7.79mm - 11.98mm$  with an average of  $9.86mm$ . This result is pretty good that shows that the system can be developed for industrial use. The different configurations in experiments 1-3 show different orientations of intersection that give different projection coordinates in the projector image plane.

In pose 2, the result for experiment 1-2, deviation within the norm of the vector is in the range from  $22.22mm - 29.23mm$ , with the highest deviation within the translation along the  $\{c\}_x$ . This isn't the best result, and the reason for that could be that these results did have more noise than the images in pose 1. It looks like somehow the camera settings used to capture the point cloud weren't tuned in.

The deviation in both pose 1 and 2 experiments can indicate an error in the forms of different things. Important to mention is that the  $A_{gap}$  aren't the same in pose 1 and pose 2, since the defined diameters  $S_m$  and  $S_{cm}$  aren't the same, and therefore is difficult to compare the result from the two different poses. However, some factors can indicate the difference of error between the two poses. When the projector intrinsic parameters were calibrated with the stereo calibration between  $\{c\}$  and  $\{p\}$ , the calibration was done with a focus distance of approximately  $500 - 700mm$ , depending on the orientation of the calibration plane. And in pose 1, the working distance to the leg was around  $500 - 600mm$ . In pose 2, however, the working distance was around  $800 - 1000$ , indicating that the error is higher and resulting in more deviation in pose 2 than pose 1.

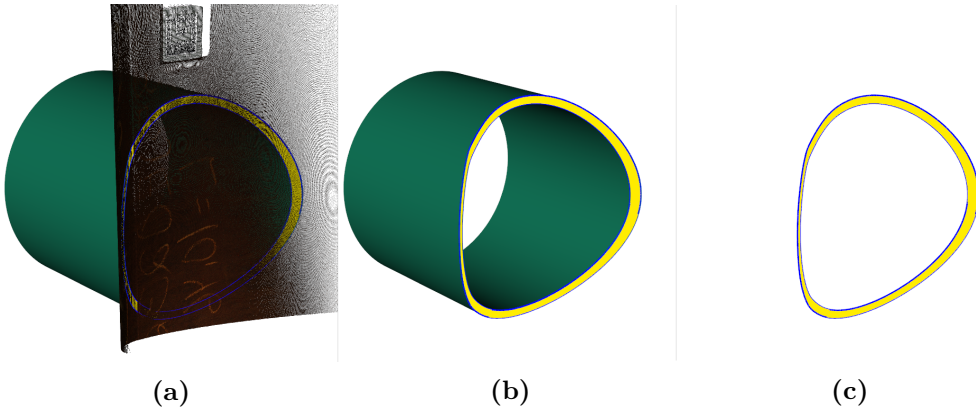
Although the result indicates that the system can be used in today's procedure, since accuracy pose 1 accuracy is near the requirement for NORSOK-M-101 [3], and can be said that are within the requirements for grinding.

## 7.4. Scaling and today's procedure

This section compares the implemented method in [chapter 4](#) against today's stub assembly procedure in [section 2.3](#) to investigate which possibilities of the implemented method can be used to improve today's method.

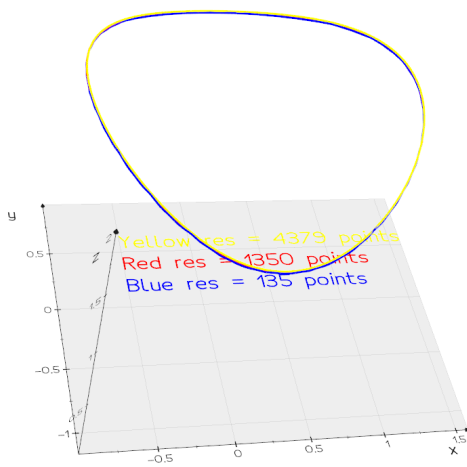
### 7.4.1. Stub section cut

Instead of defining the stub section as in [subsection 2.3.1](#), where a CNC plasma cutter defines the cut from the leg and stub's nominal diameter. The section cut can be defined by the intersection of the  $L_m$  with  $S_m$  and  $S_{cm}$  captured by the Zivid two camera. This path can either be used as G-code into the already CNC plasma cutter or the path can be moved into a robot coordinate system with a hand eye calibration. In the method, the  $S_m$  and  $S_{cm}$  are defined as synthetic meshes, and these meshes can be also be defined by capturing the meshes with a robot. This can lead to getting a better accuracy of the cut since the meshes would have better accuracy within the tube tolerances as in [Appendix A](#).

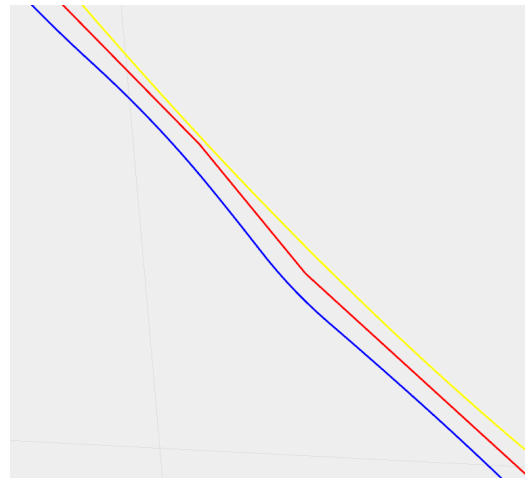


**Figure 7.3.:** Intersection path with the cut

As stated in the previous work, [subsection 1.3.1](#), the intersection paths are interpolated to a specified resolution. Through Vedo, the intersection path is obtained as in [Figure 7.3c](#). Suppose the path is of insufficient resolution. In that case, the path could either be interpolated by creating a spline of the path or create more triangle elements on the mesh by subdividing each triangle. By having more elements would give a higher resolution when intersecting two meshes. The number of points used in the first place to create mesh would also impact the resolution.



(a) Resolution



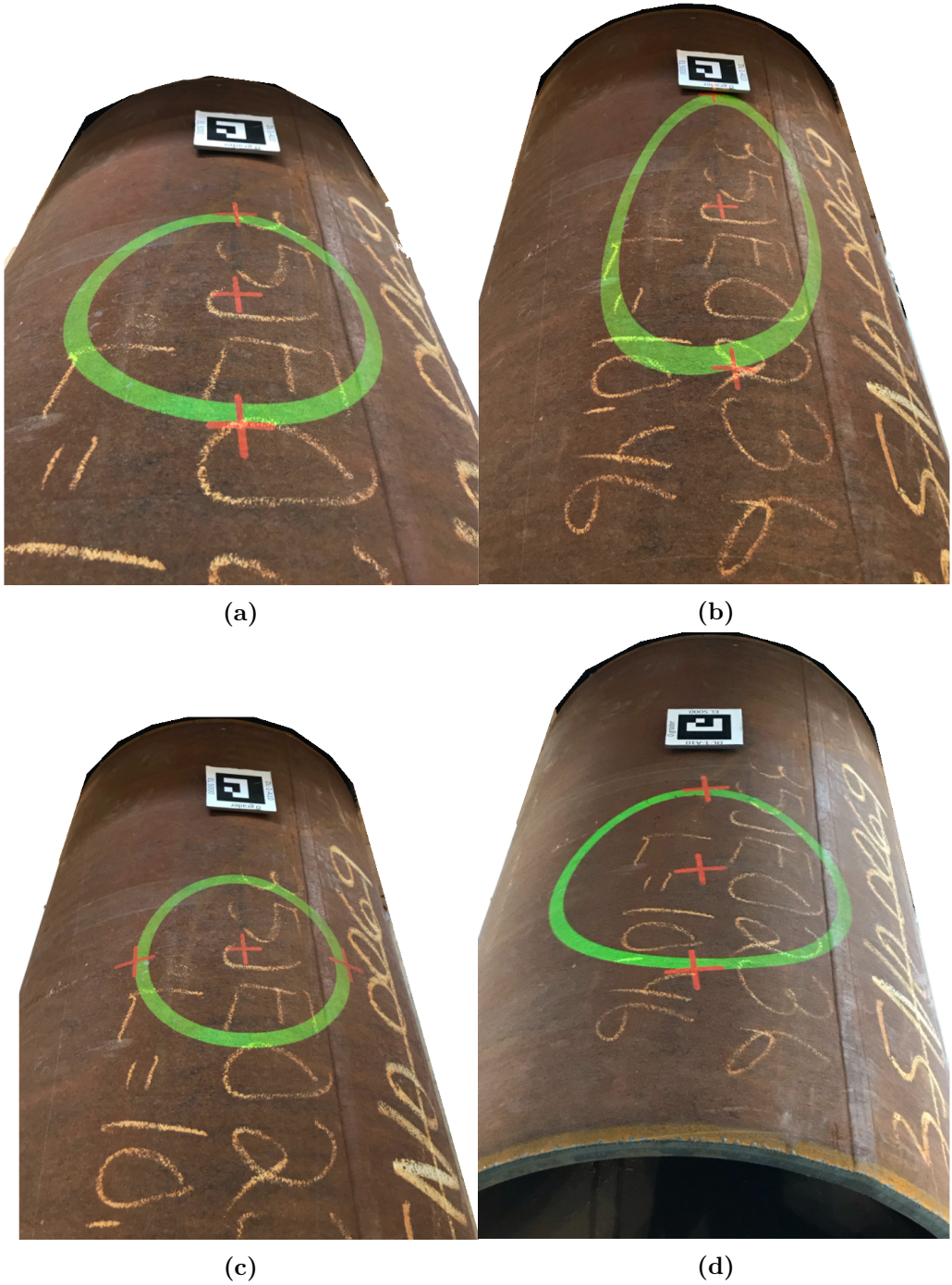
(b) Blue: intersect between two coarse mesh, Red: Spline interpolation of the blue curve by 10 times, Yellow: Subdivide mesh before intersection by a factor 5 of both meshes.

**Figure 7.4.:** Interpolation of intersection path, the shift of the curvature for see the different.

### 7.4.2. Lifting up and down

The advantage with the implemented method is that the operation in [subsection 2.3.2](#) and [subsection 2.3.3](#) can be avoided and lead to less overhead traveling crane operations through the assembly. The projection cut can be projected onto the leg as in the [Figure 2.11](#) instead of lifting, mark, and then lift down again before starting to grind the area where the stub hits the leg as in the. This could make the procedure more efficient than today and contribute to the fact that the yard at Verdal is more competitive.





(a)

(b)

(c)

(d)

Figure 7.5.: Projection in the lab area



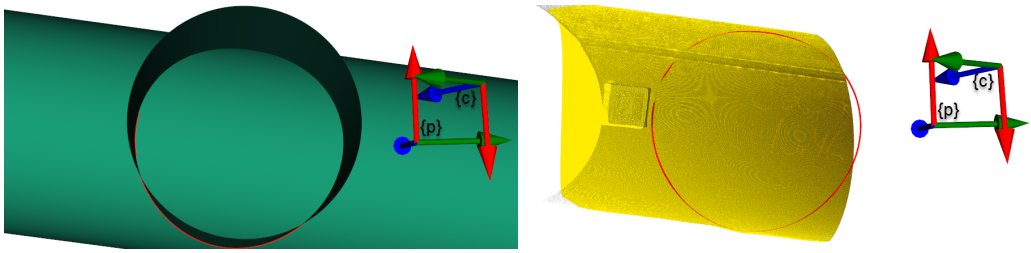
In the [Figure 2.10](#) the toe and heel points are shown. The figures on the previous page show the toe and heel point with the center point. Whether these points would be as the actual world toe, heel, and the center point isn't evaluated because of the difficulty to evaluate the requirement in [subsection 7.2.1](#), and are further work for integration of this method into today's procedure.

Since the intersection path is obtained, a robot can both cut out the section cut and weld the stub onto the leg after a defined welding groove is set.

### 7.4.3. Scaling

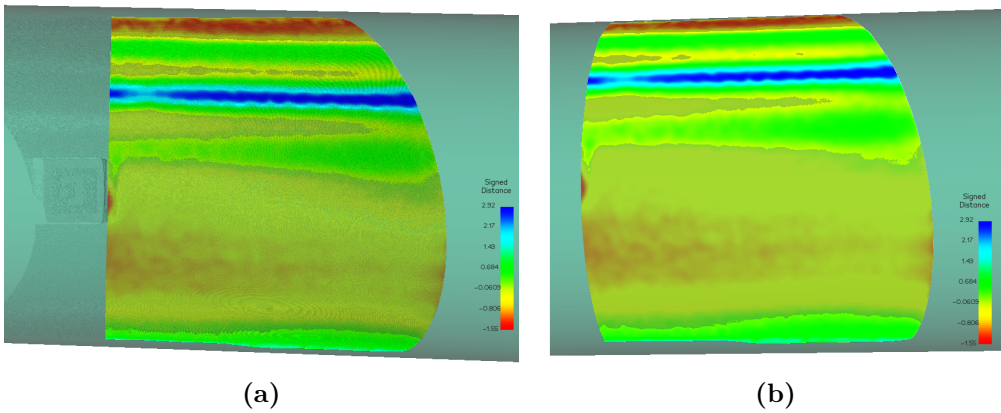
Since the tested dimension in the lab experiment is a leg width diameter of  $L_d = 600\text{mm}$ , the system is pretty scaled-down compared to many of the dimensions at the yard at Verdal. The diameters can vary from 2 meters to 6 meters. The Zivid Two's FOV would not be large enough for the largest dimensions. The largest stated FOV is  $1426 \times 894\text{ mm}$  with a working distance of  $1400\text{ mm}$ . Since the focus distance is  $700\text{ mm}$ , also the accuracy would decrease by more considerable distances. At the focus distance, the spatial resolution is  $0.39\text{ mm}$ . Zivid also has another camera that is designed for the larger dimensions, the Zivid One + camera. This camera can have a working distance of up to  $3000\text{ mm}$  with a spatial resolution of  $1.11\text{ mm}$ . This camera could have been interesting to test further at the factory out at Aker Solutions Verdal.

Another thing that could be researched is maximizing the use of a short-throw projector in pair with a 3D camera. Suppose the projector has a larger FOV than the 3D camera. Then this can be used as an advantage with some challenges. As in the [Figure 7.6a](#), the stub configuration is larger than the captured point cloud. If then, the projector has a larger FOV than the 3D camera. The cylinder fitting in the implemented method can be used to create a synthetic mesh instead of creating a mesh from the captured point cloud. However, this would make an impact on the deviation within tolerances on the captured surface. But for grinding application, where the tolerances aren't crucial, this can be used to get a bigger FOV for projection the grinding surface.



(a) Stub configuration larger than the yellow mesh. (b) Intersection curve larger than captured Zivid two point cloud.

**Figure 7.6.:** Stub configuration larger than the captured point cloud.



**Figure 7.7.:** Interpolation of intersection path, the shift of the curvature for see the different.

A problem that can occur when projecting outside the 3D camera FOV is that the calibration between the 3D camera and the projector can be more inaccurate for the area outside the 3D camera FOV. This is something to take in mind when investigating this.

# Chapter 8.

## Conclusion

This project has developed an automated method for marking the stub cross-section cut for grinding using a 3D-camera-projector setup. The method is implemented both synthetically, through the Blender platform, and on an industrial system consisting of a Zivid Two 3D camera with an Acer projector. Based on the information presented by the results in [chapter 6](#) as discussed in [chapter 7](#) and illustrations explaining the implemented method in [chapter 4](#), it can be concluded that the primary objective of this thesis has been fulfilled.

Various experiments were performed with both synthetic and real-world data where stubs in different configurations were transformed from the defined reference system to a set configuration. With the assumptions mentioned in [section 7.2](#), such as the reference system and the projected cross-section not being compared to a real cross-section in the same configuration, it is seen that results were obtained close to the position tolerances for tubular nodes in the standard NORSOK-M101, *Structural steel fabrication*.

The synthetic experiment shows that Blender is suitable as a synthetic 3D camera projector system. Deviations in the results could indicate that either the extracted projector intrinsic parameters are not accurate enough or the transformation between the modeled camera and the projector is incorrect. The default units in Blender are set to meters, so the extracted format could also be a factor that impacts the results.

The lab experiments in two different poses present projection areas in different sizes onto the leg's surface. The observed deviations could indicate a calibration error between the Zivid Two camera and the Acer projector.

The implemented method can also be extended to other applications within the current stub assembly procedure. For example, the intersection path between the meshes in the implemented method can also automate cutting and welding

procedure by using robotic systems with a hand-eye calibration.

## 8.1. Further Work

Further work could consist of looking into the reference system in the method and finding a way to evaluate that the coordinates are transformed as the manual measurement in today's stub assembly procedure. The reference system axes are based on a cylinder fitting algorithm for defining the common cylinder axis to the captured point cloud of the leg's surface. The least square fitting algorithm [30] used here is not compared to other algorithms, thus an investigation into alternative algorithms should be performed. Some suggestions for comparing methods could be to:

1. Add or compute the normal components of the point cloud data.
2. Use a RANSAC [31] results as an initial guess, optimize the cylinder coefficients with the inlier points and compute normals using nonlinear optimization algorithms, such as a Levenberg–Marquardt algorithm.

The detection of the Aruco marker defines the reference system's position. In today's procedure, the reference point is a marker that is punched into the leg's surface. Instead of recognizing the Aruco Marker, the punched marker can be identified with CNNs or create another reference system.

The projected area should be matched to a real cross-section cut stub specimen from the CNC plasma cutter to see the deviation in combination with the reference system.

A static test rig should be built with the proper hardware and be tested in the environment at Aker Solutions Verdal to evaluate the method's robustness and find further improvement possibilities.

# References

- [1] E. T. Nerol, “3D-Camera-Projector Calibration with Point Clouds,” *TPK 4560*, Dec. 2020.
- [2] *Oil platform*, Nov. 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Oil\\_platform](https://en.wikipedia.org/wiki/Oil_platform).
- [3] “Structural steel fabrication,” Standards Norway, N-1326 Lysaker, Norway, Standard, 2011.
- [4] M. Lind and P. Ystgaard, “Mesh-based tool path calculations for tubular joints,” *Advances in Mechanical Engineering*, vol. 13, pp. 1–13, 2019.
- [5] T. L. Sundkøien, “Rørmontasje Ved Jacketsbygging,” *Department of Mechanical and Industrial Engineering*, vol. 12, pp. 4–15, 2013.
- [6] Kvaerner, *Edvard grieg jacket, the various parts of a jacket explained*, [Online; accessed September 7, 2020], 2019. [Online]. Available: [https://www.kvaerner.com/wp-content/uploads/2019/04/Jacket\\_explained-infographics\\_6-1.jpg](https://www.kvaerner.com/wp-content/uploads/2019/04/Jacket_explained-infographics_6-1.jpg).
- [7] *Oil platform*, Nov. 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Oil\\_platform](https://en.wikipedia.org/wiki/Oil_platform).
- [8] “Fabrication and testing of offshore structures,” Kvaerner, Kvaerner Verdal, Norway, Manual, 2019.
- [9] HGG Group, *Spc 1500–3000 pt, cnc pipe cutting machine for vessels and offshore*, [Online; accessed September 8, 2020], 2017. [Online]. Available: [https://hgg-group.com/wp-content/uploads/2016/01/Pipe\\_cutting\\_services-367x200.jpg](https://hgg-group.com/wp-content/uploads/2016/01/Pipe_cutting_services-367x200.jpg).
- [10] K. Lynch and F. C. Park, *Modern robotics: mechanics, planning, and control*. Cambridge University Press, 2017.
- [11] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Second. Cambridge University Press, ISBN: 0521540518, 2004.
- [12] P. O. Egeland, “Robot vision,” Tech. Rep., 2020.
- [13] *Corner detection*, Jul. 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Corner\\_detection](https://en.wikipedia.org/wiki/Corner_detection).

- [14] C. Harris and M. Stephens, “A combined corner and edge detector,” in *Proc. AVC*, doi:10.5244/C.2.23, 1988, pp. 23.1–23.6.
- [15] F. Mokhtarian and R. Suomela, “Curvature scale space for robust image corner detection,” in *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, vol. 2, 1998, 1819–1821 vol.2. DOI: [10.1109/ICPR.1998.712083](https://doi.org/10.1109/ICPR.1998.712083).
- [16] S. M. Smith and J. M. Brady, *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997. DOI: [10.1023/a:1007963824710](https://doi.org/10.1023/a:1007963824710). [Online]. Available: <https://doi.org/10.1023/a:1007963824710>.
- [17] *Camera calibration*. [Online]. Available: [https://docs.opencv.org/master/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html).
- [18] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000. DOI: [10.1109/34.888718](https://doi.org/10.1109/34.888718).
- [19] B. Huang, *Kinect and projector pair calibration*, Mar. 2018. [Online]. Available: <https://bingyaohuang.github.io/Calibrate-Kinect-and-projector/>.
- [20] N. Janakiev, *Understanding the covariance matrix*, Aug. 2018. [Online]. Available: <https://datascienceplus.com/understanding-the-covariance-matrix/>.
- [21] *Centering matrix*, Oct. 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Centering\\_matrix](https://en.wikipedia.org/wiki/Centering_matrix).
- [22] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit—An Object-Oriented Approach To 3D Graphics*, Fourth. Kitware, Inc., 2006.
- [23] A. Updegrove, N. Wilson, and S. Shadden, “Boolean and smoothing of discrete polygonal surfaces,” *Advances in Engineering Software*, vol. 95, pp. 16–27, May 2016. DOI: [10.1016/j.advengsoft.2016.01.015](https://doi.org/10.1016/j.advengsoft.2016.01.015).
- [24] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R  o, M. Wiebe, P. Peterson, P. G  rard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: [10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2). [Online]. Available: <https://doi.org/10.1038/s41586-020-2649-2>.
- [25] M. M. et al., “Vedo, a python module for scientific analysis and visualization of 3d objects and point clouds,” 2021. DOI: [doi.org/10.5281/zenodo.4287635](https://doi.org/10.5281/zenodo.4287635).

- [26] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
- [27] Q.-Y. Zhou, J. Park, and V. Koltun, “Open3D: A modern library for 3D data processing,” *arXiv:1801.09847*, 2018.
- [28] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: [10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55).
- [29] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
- [30] D. Eberly, “Least squares fitting of data by linear or quadratic structures,” July 15, 1999.
- [31] M. A. Fischler and R. C. Bolles, “Random sample consensus,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. DOI: [10.1145/358669.358692](https://doi.org/10.1145/358669.358692). [Online]. Available: <https://doi.org/10.1145/358669.358692>.
- [32] C. B. Sullivan and A. Kaszynski, “PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK),” *Journal of Open Source Software*, vol. 4, no. 37, p. 1450, May 2019. DOI: [10.21105/joss.01450](https://doi.org/10.21105/joss.01450). [Online]. Available: <https://doi.org/10.21105/joss.01450>.
- [33] Dawson-Haggerty et al., *Trimesh*, version 3.2.0. [Online]. Available: <https://trimsh.org/>.
- [34] D. Moreno and G. Taubin, “Simple, accurate, and robust projector-camera calibration,” in *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, 2012, pp. 464–471. DOI: [10.1109/3DIMPVT.2012.77](https://doi.org/10.1109/3DIMPVT.2012.77).
- [35] *Camera calibration and 3d reconstruction*. [Online]. Available: [https://docs.opencv.org/2.4/modules/calib3d/doc/camera\\_calibration\\_and\\_3d\\_reconstruction.html](https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html).





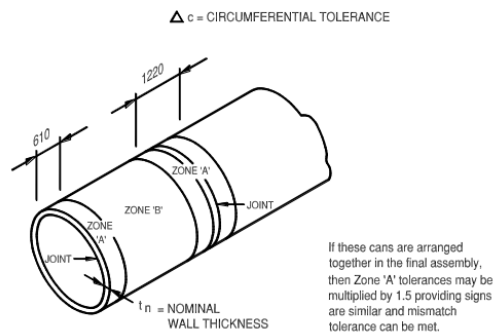
# Appendix A.

## Fabrication Tolerances

### A.1. Tubulars

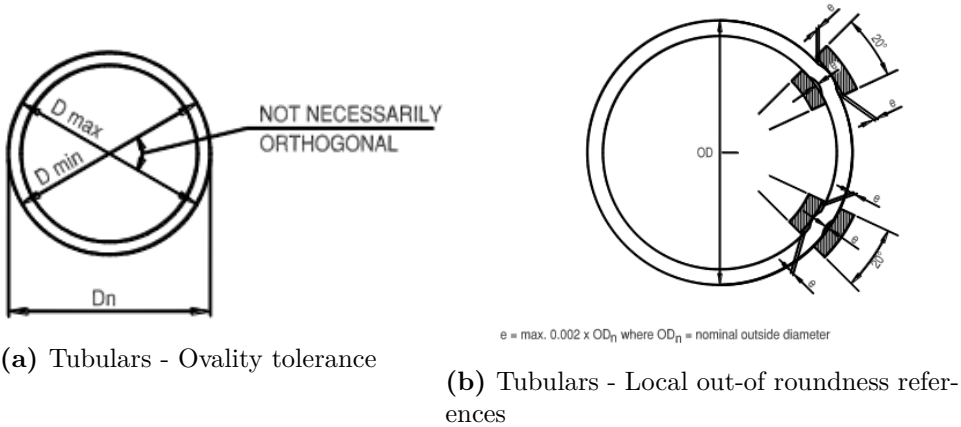
This section include the required tolerances according to NORSOK STANDARD M-101 Structural Steel Fabrication [3], and taken straight out of the standard. The allowable tolerance given for individual tubular segments shall not be cumulative for the finished tubular.

1. **Circumference:** The external circumference shall not depart from the nominal external circumference by more than the following in the figure:
  - a) measured at joints or within  $\pm 610$  mm from the joint (Zone A in Figure E.5) 30 % of the nominal wall thickness or  $\pm 10$  mm, whichever is the smallest;
  - b) the tolerances in (i) may be increased by 50 % for the remaining length of the tubular (Zone B in Figure E.5).



**Figure A.1.:** Tubulars - Circumferential tolerance

2. **Out of roundness(Ovality):** Ovality is defined as the difference between the measured maximum and minimum internal (or external) diameters and shall not be more than 1 % of the nominal OD (OD<sub>n</sub>) or 8 mm, whichever is the least, see figure.

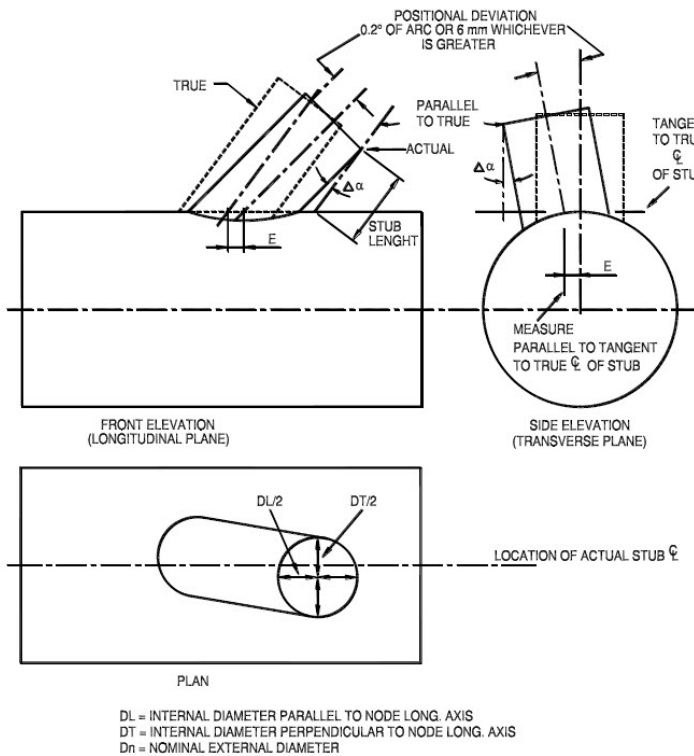


**Figure A.2.:** Ovality

3. **Out of circularity:** Circularity is defined as the difference between the actual and the average radius, both being determined from the optimum centre of the tubular. Maximum difference is not to be more than 0,25 % of OD<sub>n</sub>.
4. **Straightness:** The maximum allowable deviation from straightness in any 3 m increment of length shall be 3 mm. The straightness deviation over tube length (L), shall not exceed 0,001 x L, with maximum 10 mm deviation for lengths up to 12 m. Above 12 m length maximum allowable deviation is 12 mm. Out of straightness shall be checked on two longitudinal planes separated by 90.
5. **Length:** Unless otherwise noted, the tubular shall be delivered within following tolerances:
- unbevelled ends:  $L_a \geq L_n + 25$  mm.
  - bevelled ends:  $L_a = L_n \pm 5$  mm.
6. **Tube ends:** The tube ends shall be perpendicular to the longitudinal axis within the following tolerances:
- unbevelled ends: 5 mm.
  - bevelled ends: 3 mm.

7. **Local out-of roundness:** The local out-of roundness shall not deviate from the theoretical curvature by more than  $e = 0,002 \times OD_n$  (see Figure E.7). The local out-of roundness shall be measured inside or outside over  $20^\circ$  of the circumference.
8. **Local straightness:** is defined as the deviation of the shell plate from a straight generator of length (L) parallel to the true centre line of the tubular. This tolerance shall not exceed 20 % of the wall thickness. Local straightness shall be checked on the inside or outside of tubulars with a nominal external diameter greater than 2 000 mm or with a nominal external diameter to nominal wall thickness ratio greater than 65. These checks shall be carried out at  $45^\circ$  intervals of arc with  $L = 3$  m.

## A.2. Tubular nodes



Dn	E	$\Delta \alpha$
$\leq 3500$ mm	6 mm	$\pm 0.2^\circ$
$> 3500$ mm	10 mm	$\pm 0.2^\circ$

Figure A.3.: Node stub location



## Appendix B.

# Zivid and Projector Pair Calibration

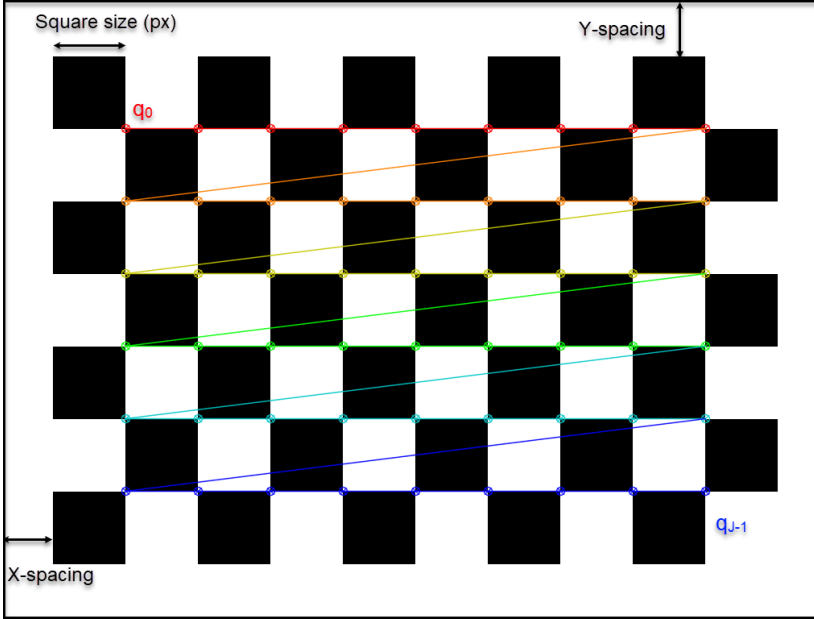
This chapter introduces the Zivid and projector pair calibration method. A calibrate method using Zhang’s method [18] without a printed checkerboard. Instead, the checkerboard pattern can be projected onto a plane and capture at least three different poses. This calibration derive the intrinsic parameters of the projector and the extrinsic parameter between the Zivid and the projector using OpenCV[35] functions as *CalibrateCamera()* and *StereoCalibrate()*. This method is inspired from Bingyao Huang [19]. All sourced code is linked to this [repository](#).

## B.1. Projector calibration

A projector can be considered the inverse of a camera, and the pinhole model also applies to a projector. The difference is that the camera captures the screen’s image while the projector projects it onto the screen. When calibrating a camera, the object’s points are known as the known parameter, mentioned in [subsection 3.1.5](#), and the image points are the unknown parameter. Projector calibration is the opposite of this, the image points in the projector image coordinates are known, and the object points are unknown. The main problem here is that the object’s points are unknown.

### B.1.1. Generate checkerboard image

In this method, for projector calibration, a checkerboard pattern is needed. Checkerboard pattern can be generated from websites or at your own using OpenCV's integrated script for generating a [pattern](#). Since the projected checkerboard pattern is the projector projected image, the pattern needs to be the same as the projector's resolution. Since the OpenCV *findchessboardcorners* requires white borders around the checkerboard pattern, an added spacing in both x and y directions is needed.



**Figure B.1.:** Checkerboard Pattern(9 columns x 6 rows), Resolution (1024 X 768). Black border are for visualization the x and y-spacing.

The inner corner coordinates in the projector image space is saved in the right order to be compared to the object points. The checkerboard criteria define how many inner corners there are in the checkerboard pattern. In [Figure B.1](#) there are 54 inner corners because of a pattern with (9 columns x 6 rows). The image points is denoted as

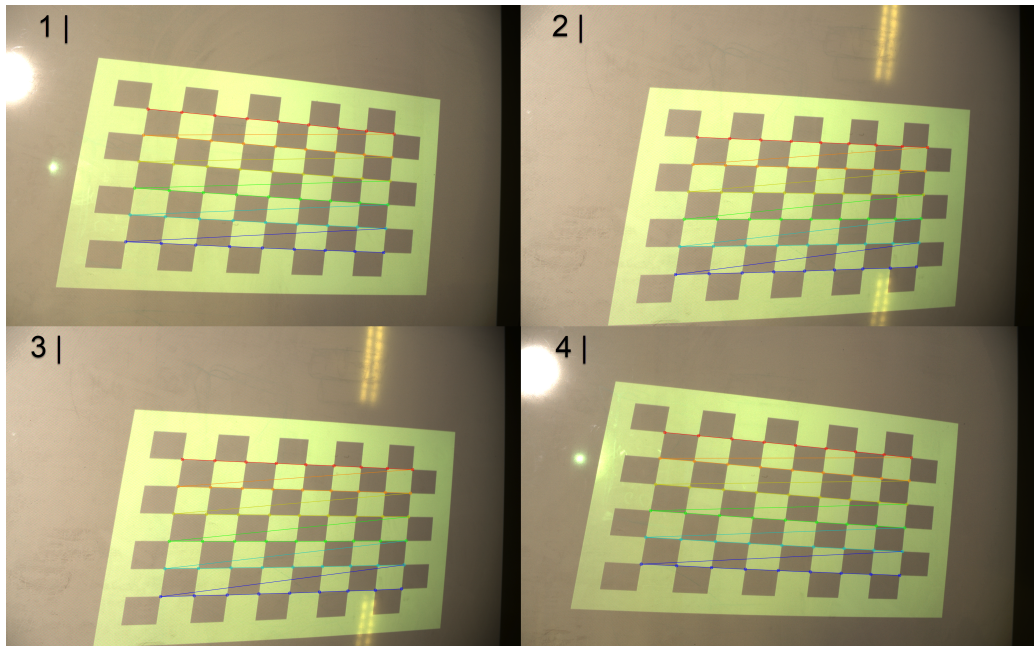
$$\mathbf{P}_p^{2d} = [\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_i, \dots, \mathbf{q}_{J-1}] \quad (\text{B.1})$$

where  $\mathbf{q}_i = [u_i, v_i]$  are the correspondent pixel coordinates of the  $i^{\text{th}}$  in the projector image space and  $J$  are number of inner corners in [Figure B.1](#). An important moment is that [\(B.1\)](#) match the color order in the image [Figure B.1](#). The red row represent the first 9 element and the dark blue represent the last one in [\(B.1\)](#). The saved checkerboard coordinates are obtained from the function

`Save_Projector_Image_Coordinates()` in `ProjectorCalibrate.py`.

### B.1.2. Capture zivid image frames

Now the checkerboard image [Figure B.1](#) can be projected by the projector onto a plane [Figure B.2](#) to capture at least three different poses according to Zhang's method.



**Figure B.2.:** 4 projected checkerboard poses onto a plane

Given a Zivid color checkerboard image, the extracted 2D checkerboard corners are found by using OpenCV's `findChessboardCorners`. The Zivid camera image coordinates are given as

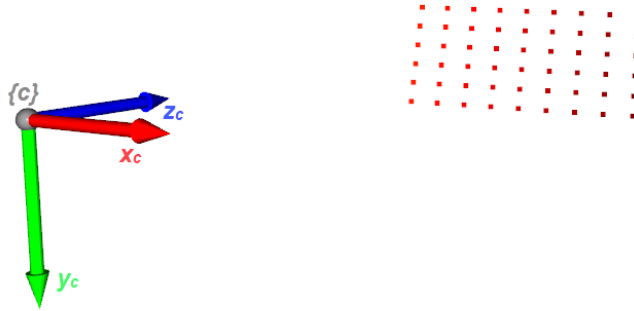
$$\mathbf{P}_c^{2d} = [l_0, l_1, \dots, l_i, \dots, l_{J-1}] \quad (\text{B.2})$$

where  $\mathbf{q}_i = [u_i, v_i]$  are the correspondent pixel coordinates of the  $i^{\text{th}}$  in the Zivid camera image space. It is important that the  $\mathbf{P}_c^{2d}$  iterate over the same order as in (B.1). Since the Zivid point cloud is organized 1:1 that each pixel coordinate in  $\mathbf{P}_c^{2d}$  has correspondence XYZ-coordinates. Then the extracted 3D-coordinates would be denoted as

$$\mathbf{P}^{3d} = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_{J-1}] \quad (\text{B.3})$$

where  $\mathbf{x}_i = [x_i, y_i, z_i]$  as the corresponding 3D coordinates of  $\mathbf{P}_c^{2d}[i]$ . Also  $\mathbf{P}^{3d}$  can

be obtained by using the depth(Z-coordinate) for the checkerboard pattern and the Zivid intrinsic parameters. The visualized  $\mathbf{P}^{3d}$  checkerboard board coordinates



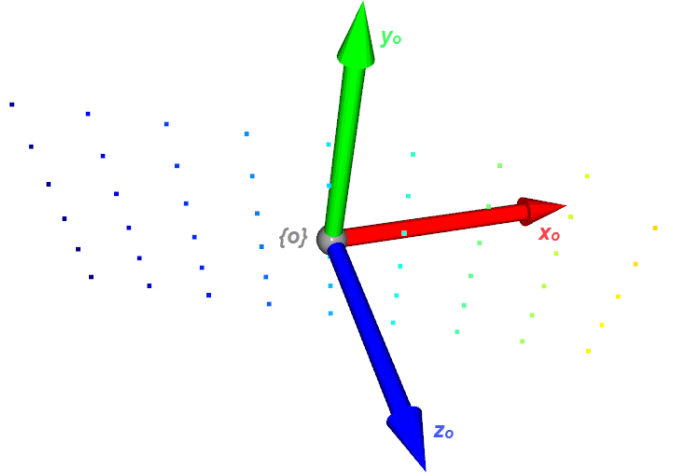
**Figure B.3.:** Checkerboard Coordinates in depth space relative to the Zivid camera in Open3d.  $\{c\}$ , represent the Zivid camera frame.

From the figure, the Z-coordinate of  $\mathbf{P}^{3d}$  are nonzero because the coordinates are defined in the Zivid depth camera’s view space rather than the projected checkerboard’s objects space. As explained in [subsection 3.1.5](#) that object points Z-coordinate is required to be set equal to zero to be sent in as an input to *CalibarateCamera* since Zhang’s method assumes all object points reside on the XY plane of the checkerboard object-space. The solution for this was to set the Z-Coordinate equal to zero within a conventional printed checkerboard since all checkerboard coordinates were on the same plane. In this case, the projected checkerboard is distorted and skewed due to the projector perspective projection. The distortion varies each time the plane pose is changed relative to the Zivid-projector system. Because the distance to each point isn’t the same, the projected checkerboard image will have a different unknown scale and shape.

### B.1.3. Rotate 3D points using eigenvectors

Since  $\mathbf{P}^{3d}$  has a planar shape, the checkerboard coordinates can be transformed to the canonical view, so the centroid of the checkerboard coordinates is set as the origin. This is obtained using either left or right side in [\(3.39\)](#), and then checkerboard coordinates would like the [Figure B.4](#) below





**Figure B.4.:** Origin set as checkerboard coordinates as centroid,  $\{o\}$  represent the checkerboard object frame.

We denote the coordinates in the figure as

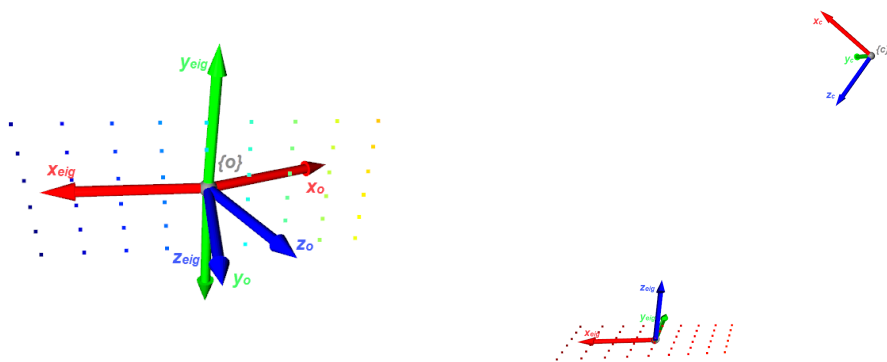
$$\mathbf{P}_c^o = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_{J-1}] \quad (\text{B.4})$$

Aligning the checkerboard coordinates with the XY-plane, the coordinates need to be rotated using the plane normal (Z-axis) and X, Y axes directions in the checkerboard object space. This can be done in the three ways explained in [subsection 3.2.2](#). Since the projected checkerboard points are projected onto a plane with tolerance deviation within planarity, the points have some irregularity within laying in the same plane. The best-fitting solution is to obtain a checkerboard coordinate axis relative to the plane using eigenvectors of the covariance matrix determined by the (3.41), using the SVD. In the figure [Figure B.4](#), the eigenvector is visualized.

In order to rotate the coordinates aligned with XY-plane the eigenvector  $U^T$  need to be left multiplied to  $\mathbf{P}_c^o$  as in the (3.42). Then the checkerboard coordinates are aligned with the XY-plane [Figure B.6](#). As mentioned, the plane has deviation within planarity, and the Z-values need to be equal to zero to be sent into *CalibrateCamera* and *StereoCalibrate* functions. These coordinates are denoted as

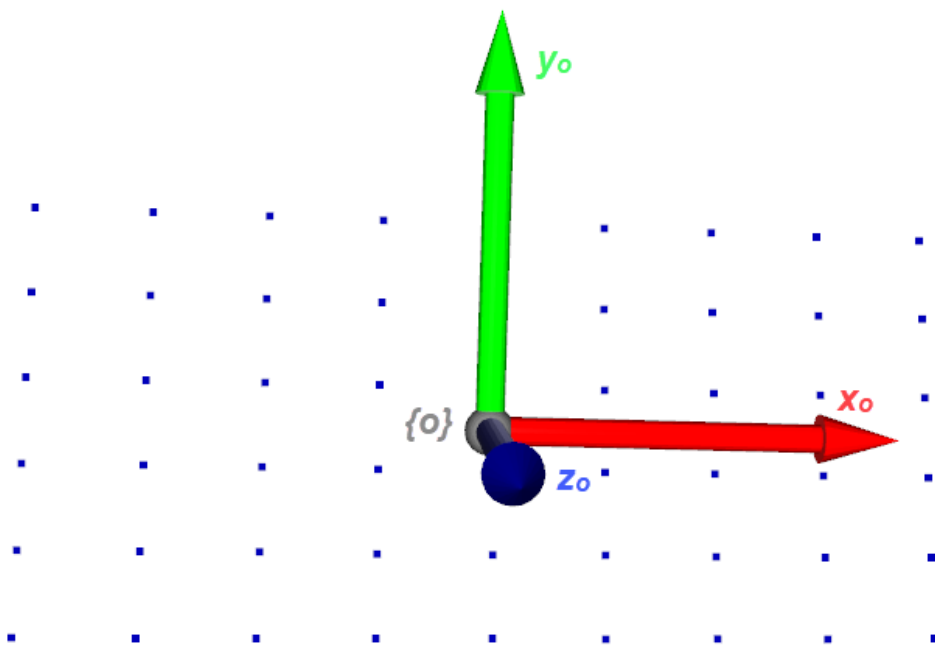
$$\mathbf{P}_{checker}^{\text{object}} = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_i, \dots, \mathbf{x}_{J-1}] \quad (\text{B.5})$$

Where  $\mathbf{P}_{checker}^{\text{object}}$  is the relative coordinate in the checkerboard object space.



(a) Eigenvectors compared to Figure B.4 (b) Eigenvector relative to the zivid camera

**Figure B.5.:** Eigenvectors



**Figure B.6.:** Checkerboard coordinates aligned with the XY-Plane

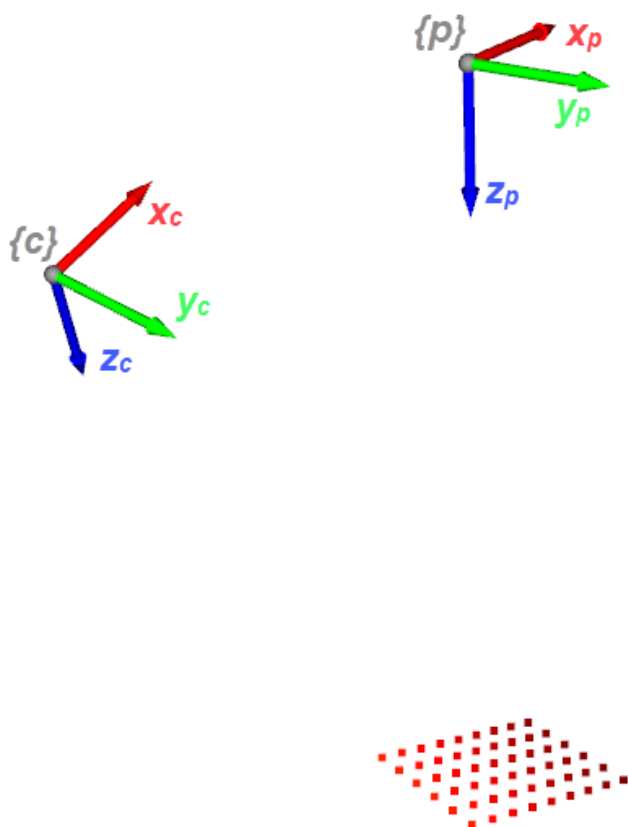
### B.1.4. Projector intrinsics and Zivid depth camera extrinsics

Finally we have the pairs of objectpoints( $\mathbf{P}_{checker}^{object}$ ) and imagepoints( $\mathbf{P}_p^{2d}$ ) and the resolution size of the projector, which can be send in to the function *CalibrateCamera()* and the function return the intrinsic as in [subsection 3.1.5](#),  $\mathbf{P}_{intrinsics}$  the distortion coefficients [Equation 3.20](#)),  $\mathbf{P}_{dist}$ , the vector rotation for each checkerboard view  $\mathbf{r}_{vecs}$  with the same translation vector  $\mathbf{t}_{vecs}$

In OpenCv the *stereocalibrate* estimates the transformation between two cameras(camera-projector) making a stereo pair. The input is the objectpoints seen by the stereo-pair, the intrinsic and distortion coefficients of the stereo-pair. The transformation in the form of rotation and translation is expressed relative to left camera in the system. The input send in is:

1. objectsPoints - ( $\mathbf{P}_{checker}^{object}$ )
2. imagePoints1 - ( $\mathbf{P}_c^{2d}$ )
3. imagePoints2 - ( $\mathbf{P}_p^{2d}$ )
4. cameraMatrix1 - Zivid camera intrinsic matrix
5. distCoeffs1 - Distortion coefficients zivid camera
6. cameraMatrix2 -  $\mathbf{P}_{intrinsics}$
7. distCoeffs2 -  $\mathbf{P}_{dist}$

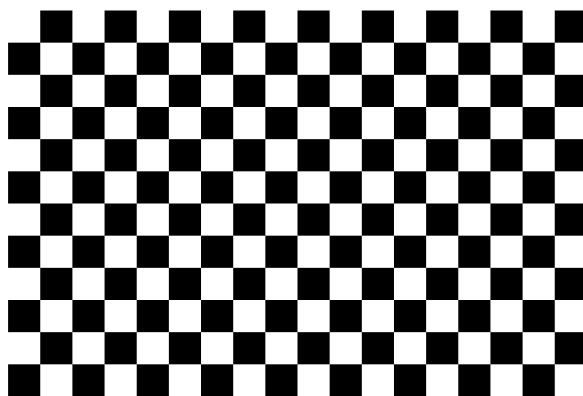
The output are the R, the rotation between the Zivid camera space and the projector space. T are the translation vector in between the two coordinate systems. Below we see the transformation between the two coordinate systems:



**Figure B.7.:** The projector frame  $\{p\}$  relative to the zivid camera frame  $\{c\}$ .

## Appendix C.

# Additional Figures



**Figure C.1.:** Chessboard 11x17, square size 60 pixels



# Appendix D.

## Mathematics

### D.1. Centering matrix

$$C_n \cdot P_{3d} = P_{3d} - \bar{X} \tag{D.1}$$

Where:

$$1. C_n = I_n - \frac{1}{n} \cdot II^T_n = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} - \frac{1}{n} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}_{n \times 1} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}_{1 \times n}$$

$$2. P_{3d} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \dots & \dots & \dots \\ x_n & y_n & z_n \end{bmatrix}_{n \times 3}$$

$$3. \bar{X} = [\bar{x}_n, \bar{y}_n, \bar{z}_n]$$

If we look at left side in equation 4.1:

$$\begin{bmatrix} 1 - \frac{1}{n} & \frac{1}{n} & \frac{1}{n} \\ \frac{1}{n} & 1 - \frac{1}{n} & \frac{1}{n} \\ \frac{1}{n} & \frac{1}{n} & 1 - \frac{1}{n} \end{bmatrix}_{n \times n} \cdot \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \dots & \dots & \dots \\ x_n & y_n & z_n \end{bmatrix}_{n \times 3} \tag{D.2}$$

=

$$\begin{bmatrix} (1 - \frac{1}{n}) \cdot x_1 - \frac{1}{n} \cdot x_2 \cdots - \frac{1}{n} \cdot x_n & (1 - \frac{1}{n}) \cdot y_1 - \frac{1}{n} \cdot y_2 \cdots - \frac{1}{n} \cdot y_n & (1 - \frac{1}{n}) \cdot z_1 - \frac{1}{n} \cdot z_2 \cdots - \frac{1}{n} \cdot z_n \\ -\frac{1}{n} \cdot x_1 + (1 - \frac{1}{n}) \cdot x_2 \cdots - \frac{1}{n} \cdot x_n & -\frac{1}{n} \cdot y_1 + (1 - \frac{1}{n}) \cdot y_2 \cdots - \frac{1}{n} \cdot y_n & -\frac{1}{n} \cdot z_1 + (1 - \frac{1}{n}) \cdot z_2 \cdots - \frac{1}{n} \cdot z_n \\ \dots & \dots & \dots \\ -\frac{1}{n} \cdot x_1 - \frac{1}{n} \cdot x_2 + \cdots (1 - \frac{1}{n}) \cdot x_n & -\frac{1}{n} \cdot y_1 - \frac{1}{n} \cdot y_2 + \cdots (1 - \frac{1}{n}) \cdot y_n & -\frac{1}{n} \cdot z_1 - \frac{1}{n} \cdot z_2 + \cdots (1 - \frac{1}{n}) \cdot z_n \end{bmatrix}_{n \times 3}$$

$$=$$

$$\begin{bmatrix} x_1 - \frac{1}{n}(x_1 + x_2 + \cdots x_n) & y_1 - \frac{1}{n}(y_1 + y_2 + \cdots y_n) & z_1 - \frac{1}{n}(z_1 + z_2 + \cdots z_n) \\ x_2 - \frac{1}{n}(x_1 + x_2 + \cdots x_n) & y_2 - \frac{1}{n}(y_1 + y_2 + \cdots y_n) & z_2 - \frac{1}{n}(z_1 + z_2 + \cdots z_n) \\ \dots & \dots & \dots \\ x_n - \frac{1}{n}(x_1 + x_2 + \cdots x_n) & y_n - \frac{1}{n}(y_1 + y_2 + \cdots y_n) & z_n - \frac{1}{n}(z_1 + z_2 + \cdots z_n) \end{bmatrix}_{n \times 3}$$

(D.3)

$$\begin{bmatrix} x_1 - \frac{1}{n}(x_1 + x_2 + \cdots x_n) & y_1 - \frac{1}{n}(y_1 + y_2 + \cdots y_n) & z_1 - \frac{1}{n}(z_1 + z_2 + \cdots z_n) \\ x_2 - \frac{1}{n}(x_1 + x_2 + \cdots x_n) & y_2 - \frac{1}{n}(y_1 + y_2 + \cdots y_n) & z_2 - \frac{1}{n}(z_1 + z_2 + \cdots z_n) \\ \dots & \dots & \dots \\ x_n - \frac{1}{n}(x_1 + x_2 + \cdots x_n) & y_n - \frac{1}{n}(y_1 + y_2 + \cdots y_n) & z_n - \frac{1}{n}(z_1 + z_2 + \cdots z_n) \end{bmatrix}_{n \times 3}$$

(D.4)

$$=$$

$$\begin{bmatrix} x_1 - \bar{x}_n & y_1 - \bar{y}_n & z_1 - \bar{z}_n \\ x_2 - \bar{x}_n & y_2 - \bar{y}_n & z_2 - \bar{z}_n \\ \dots & \dots & \dots \\ x_n - \bar{x}_n & y_n - \bar{y}_n & z_n - \bar{z}_n \end{bmatrix}_{n \times 3}$$

(D.5)

If we look at the right side in equation 4.1 we see that it is the same as the left



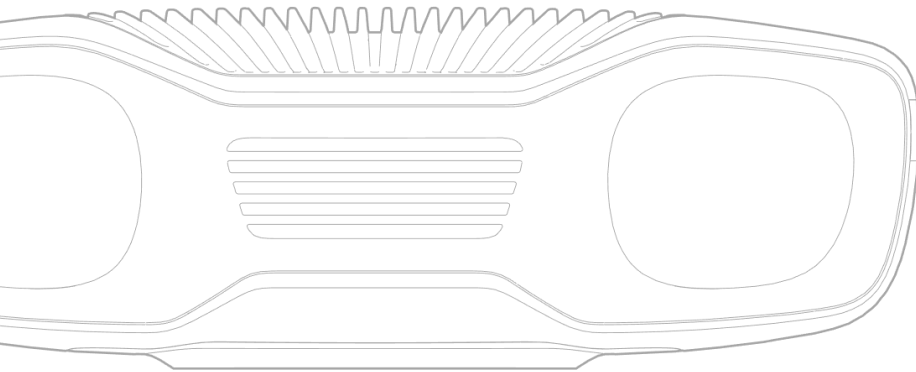
side:

$$\begin{bmatrix} x_1 - \bar{x}_n & y_1 - \bar{y}_n & z_1 - \bar{z}_n \\ x_2 - \bar{x}_n & y_2 - \bar{y}_n & z_2 - \bar{z}_n \\ \dots & \dots & \dots \\ x_n - \bar{x}_n & y_n - \bar{y}_n & z_n - \bar{z}_n \end{bmatrix}_{n \times 3} = \begin{bmatrix} x_1 - \bar{x}_n & y_1 - \bar{y}_n & z_1 - \bar{z}_n \\ x_2 - \bar{x}_n & y_2 - \bar{y}_n & z_2 - \bar{z}_n \\ \dots & \dots & \dots \\ x_n - \bar{x}_n & y_n - \bar{y}_n & z_n - \bar{z}_n \end{bmatrix}_{n \times 3} \quad (\text{D.6})$$

# Zivid Two

Technical specification

PRELIMINARY



# Appendix E.

## Data-Sheet

### E.1. Zivid Two

## Table of Contents

Table of Contents .....	2
General specifications .....	3
Operating distance and field of view .....	4
Accuracy specifications .....	6
Common conditions .....	6
Typical specifications .....	7
Physical specifications .....	9
Mechanical drawings .....	10
Connectors .....	12
Revision history .....	13

## General specifications

3D technology	Structured light
Imaging	1944 x 1200 (2.3 MP) Native 3D Color
Point cloud output	3D (XYZ) + Color (RGB) + SNR
Aperture (A)	f/1.8 to f/32
Shutter (S)	1/600 s to 1/10 s
Gain (G)	1x to 16x
Projector Brightness (B)	1/4x to 1.8x 1x = 360 lumens
Exposures per 3D frame	13
Min acquisition time	60 ms
Calibration	Factory calibrated Zivid Camera Model 2 (ZCM2)
Data interface	Ethernet (10 GigE)
Power	24 V DC
Operating temperature	0° to 45° C
Storage temperature	-20° to 60° C
Safety and EMC	CE CB EN60950 FCC Class A

## Operating distance and field of view

Recommended working distance (mm)	400 to 1200
Max working distance (mm)	300 to 1500
Field of view (mm)	754 x 449 at 700
Spatial resolution (mm)	0.39 at 700

FIGURE 1 - FIELD OF VIEW

All values in degrees or mm.

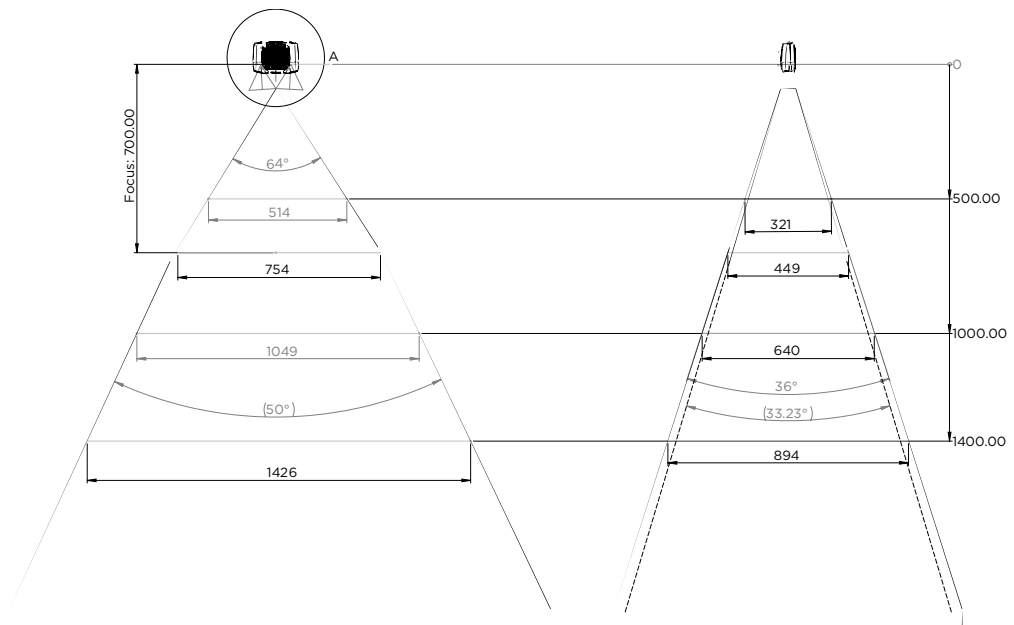
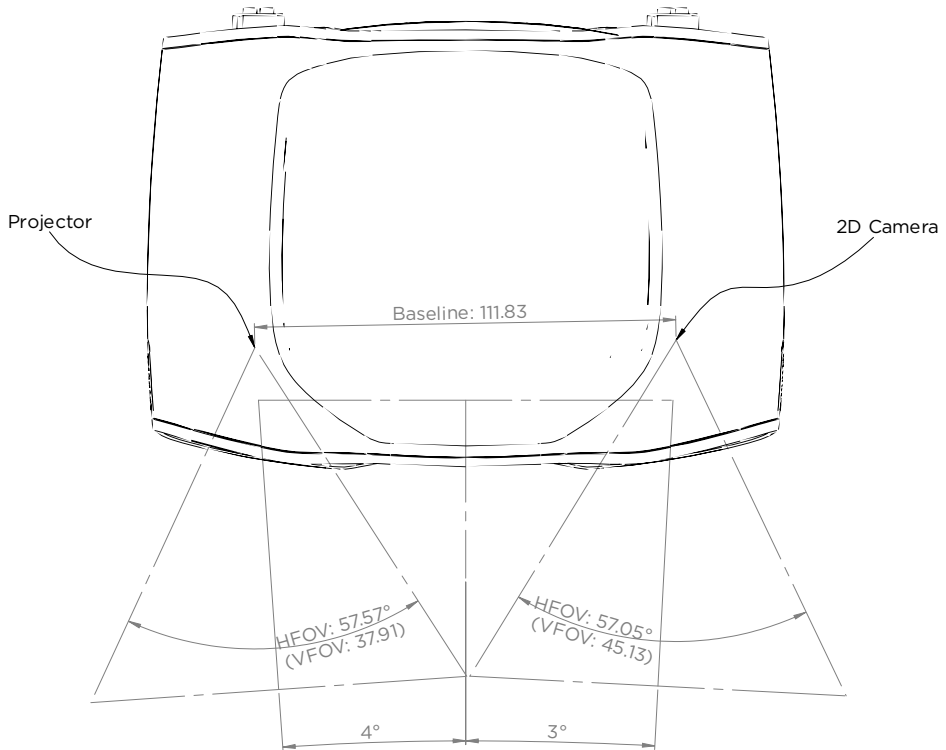


FIGURE 2 - FIELD OF VIEW

All values in degrees or mm.



## Accuracy specifications

### Common conditions

Applies to all specifications unless otherwise stated.

Parameter	Description	Typical
Working distance (D)		700 mm
Ambient temperature (Ta)		25°C
Ambient light (La)		250 lux
Aperture (A)		f/4.0
Gain (G)		1.0x
Projector Brightness (B)		1.8x
Framerate	Capture rate used during measurement.	0.2 FPS
Other		81% center crop HDR = off 10 min warm-up



## Typical specifications

Typical numbers are given at common conditions unless otherwise specified.

Property	Description	Typical
Point precision	1 $\sigma$ Euclidian distance variation for a point between consecutive measurements.	60 $\mu$ m
Local Planarity Precision	1 $\sigma$ Euclidian distance variation from a plane for a set of points within a smaller local region.	90 $\mu$ m
Global Planarity Trueness	Average deviation from a plane in field of view with noise filtering.	< 180 $\mu$ m
Global Planarity Accuracy	Average deviation from a plane in field of view without noise filtering.	< 240 $\mu$ m
Dimension Trueness	Average dimension error in field of view over 10 consecutive measurements.	< 0.20 %
	Standard deviation of dimension error in field of view over 10 consecutive measurements.	0.10%

FIGURE 3 - POINT PRECISION VS. WORKING DISTANCE

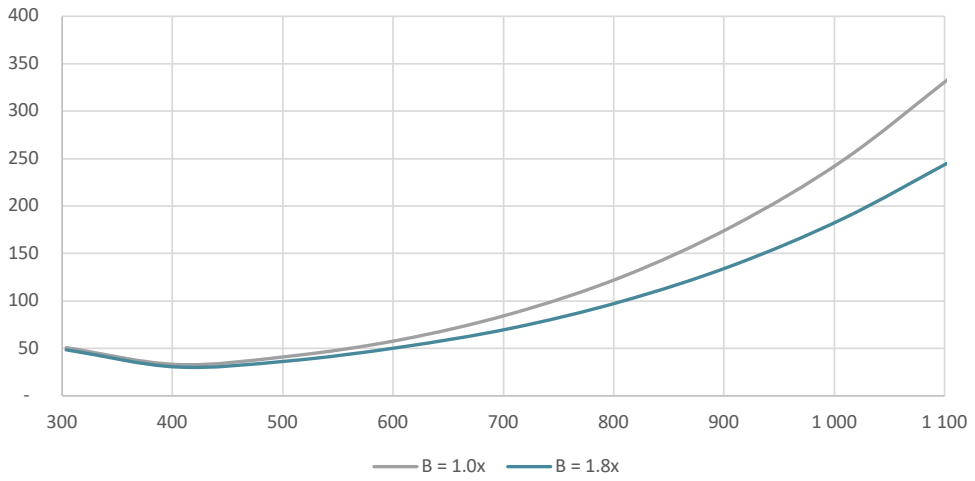
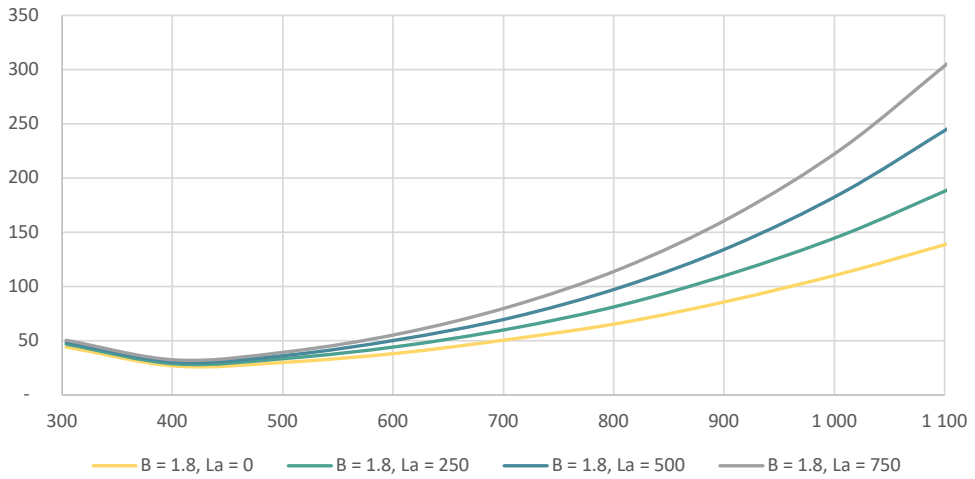


FIGURE 4 - POINT PRECISION VS. WORKING DISTANCE VS. AMBIENT LIGHT



## Physical specifications

Size and weight	Magnesium body 169 mm x 56 mm x 122 mm 880 g
Environmental	IP65 5 G Random 15 G Shock
Power connector	M12-5
Data connector	10 GigE M12-8
Power adapter	24 V 5A EU, US, and UK power plug options

# Mechanical drawings

FIGURE 5 - DIMENSIONS

All values in degrees or mm.

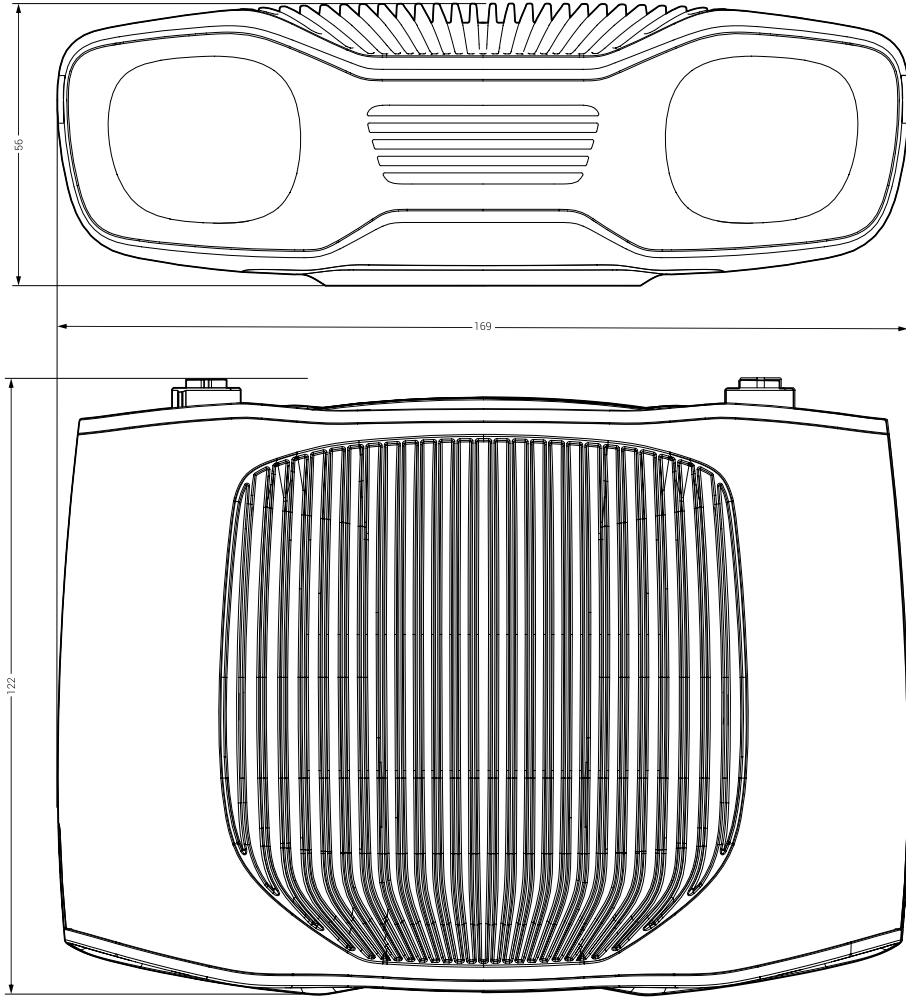
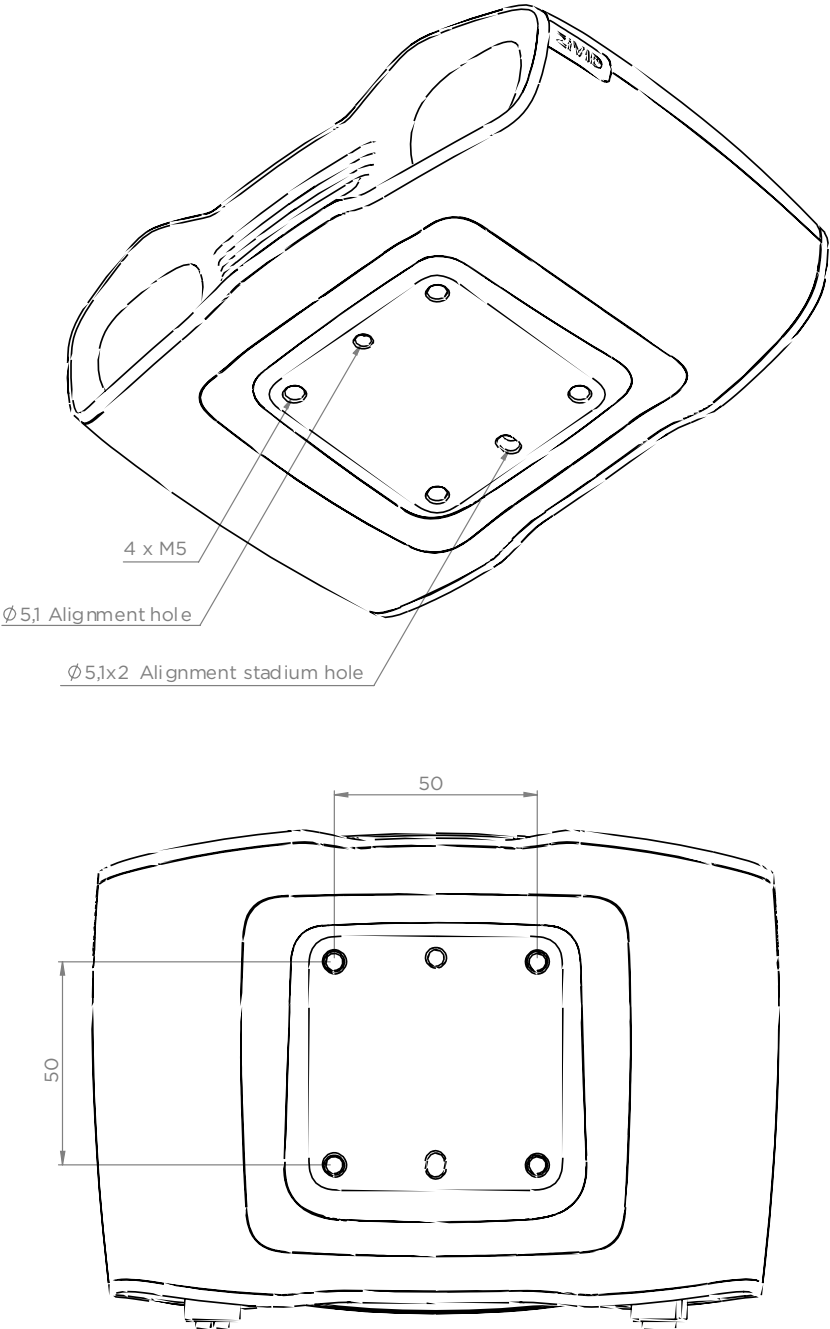


FIGURE 6 - MOUNTING OPTIONS

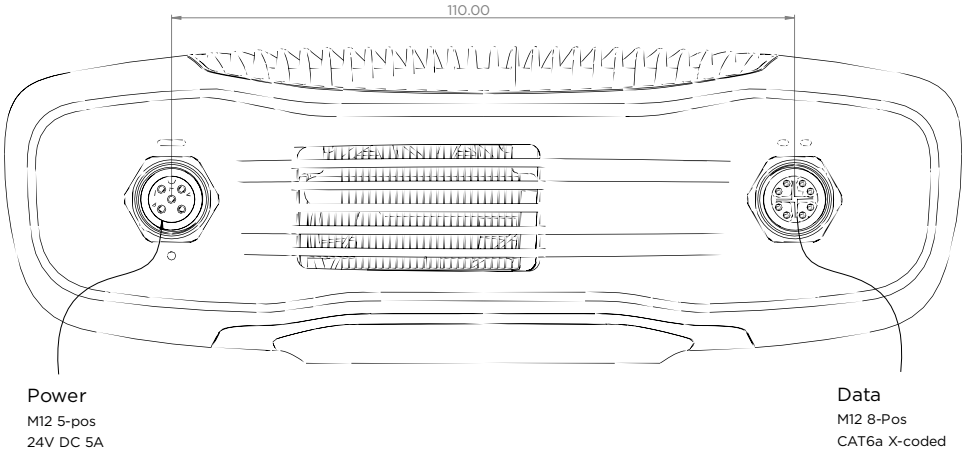
All values in degrees or mm.



Connectors

FIGURE 7 - CONNECTORS

All values in degrees or mm.



## Revision history

Ver.	Date	Notes
0.1	11/20	Initial version. Preliminary.

Zivid AS  
Gjerdrums vei 10A  
NO484 Oslo  
Norway

© 2020 Zivid AS. All rights reserved. Subject to change without notice.



## E.2. Acer Predator Z650 Specifications

The specifications below are subject to change without notice. For final specs, please refer to Acer's marketing documentation.

Projection system	DLP™
Resolution	Native: 1080p (1920 x 1080) Maximum: WUXGA (1920 x 1200) (supports reduce blanking only)
Computer compatibility	Refer to the "Compatibility Modes" section for more information.
Video compatibility	NTSC (3.58/4.43), PAL (M/N), PAL 60, SECAM, HDTV (720p, 1080i, 1080p), EDTV (480p, 576p), SDTV (480i, 576i)
Aspect ratio	Auto, 4:3, 16:9, Full, Letter Box (L.BOX), 16:6
Displayable colors	1.07 billion colors
Projection lens	<ul style="list-style-type: none"> <li>H7550ST/E155S/HE-815ST/H1P1403/H7550STz/ Z650/EG40S/HG-80ST/Q1P1504: F = 2.6 - 2.78, f = 10.20 mm - 11.22 mm, 1:1.1 Manual Zoom and Manual Focus</li> <li>H7550BD/E155D/HE-815J/H1P1406/H7550BDz: F = 2.59 - 2.87, f = 16.88 mm - 21.88 mm, 1:1.3 Manual Zoom and Manual Focus</li> </ul>
Projection screen size (diagonal) with clear focus	<ul style="list-style-type: none"> <li>H7550ST/E155S/HE-815ST/H1P1403/H7550STz/ Z650/EG40S/HG-80ST/Q1P1504: 54" (137 cm) - 300" (762 cm)</li> <li>H7550BD/E155D/HE-815J/H1P1406/H7550BDz: 30" (76 cm) - 300" (762 cm)</li> </ul>
Projection distance with clear focus	<ul style="list-style-type: none"> <li>H7550ST/E155S/HE-815ST/H1P1403/H7550STz/ Z650/EG40S/HG-80ST/Q1P1504: 3.0' (0.9 m) - 15.0' (4.6 m)</li> <li>H7550BD/E155D/HE-815J/H1P1406/H7550BDz: 3.3' (1.0 m) - 25.1' (7.6 m)</li> </ul>
Throw ratio	<ul style="list-style-type: none"> <li>H7550ST/E155S/HE-815ST/H1P1403/H7550STz/ Z650/EG40S/HG-80ST/Q1P1504: 131" ±3% @ 2 m (0.69 - 0.76:1)</li> <li>H7550BD/E155D/HE-815J/H1P1406/H7550BDz: 79" ±3% @ 2 m (1.15 - 1.50:1)</li> </ul>
Horizontal scan rate	15 - 100 KHz
Vertical refresh scan rate	24 - 120 Hz
Keystone correction	+/-40 Degrees (Vertical and horizontal), Manual & Auto
Digital zoom	2 X
Audio	10W x 2
Weight	Approximate 3.5 kg (7.71 lbs.)
Dimensions (W x D x H)	357 x 241 x 98 mm (1.17' x 0.79' x 0.32')
Power supply	Universal AC input 100 - 240 V, input frequency 50/60 Hz
Power consumption (typical)	325 W
Operating temperature	0°C to 40°C / 32°F to 104°F

