UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GABRIEL TADIELLO MORAES

Automatic Alignment of Piping System Components and Generation of CAD Models of Industrial Site Plants

Work presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Advisor: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
December 2021

ABSTRACT

The ability to (semi-)automatically obtain CAD models from physical installations has two important benefits: (i) it can be used to identify, as soon as possible during a construction process, any deviations from the original designs; and (ii) it can be used to document complex installations for which CAD representations are outdated or inexistent. Both scenarios have important practical and economic value. An ongoing project in our research group aims to reconstruct CAD representations from point clouds of industrial sites. However, pose estimation of pipes and piping system components is not perfect, resulting in misalignments in the reconstructed scene, which is unacceptable for a CAD model. For this undergraduate thesis, I propose to use optimization techniques to fix these misalignments. I also propose to convert the detected pipes and piping system components into actual CAD model representations for a popular commercial CAD software, namely AutoCAD Plant 3D.

Keywords: Reverse engineering. industrial plant sites. piping systems. CAD.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

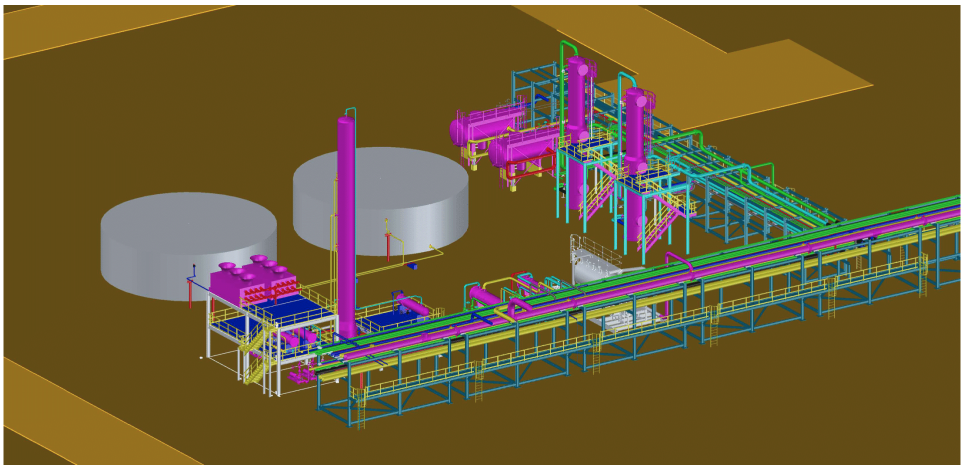
AEC	Architecture, Engineering and Construction
CAD	Computer-Aided Design
DIN	German Institute for Standardisation
ISO	International Organization for Standardization
PCA	Principal Component Analysis
RANSAC	Random Sample Consensus
SDK	Software Development Kit

CONTENTS

1 INTRODUCTION

In the Architecture, Engineering and, Construction (AEC) industry, the process of creating virtual models of existing buildings presents several benefits and has a variety of applications. For instance, it can be used to monitor the construction of a building and find deviations from the original design, or to document installations with inexistent or outdated models. Figure 1.1 shows an industrial site modeled using AutoCAD Plant 3D.

Figure 1.1 – Screenshot of a sample project of an industrial site from Plant 3D.



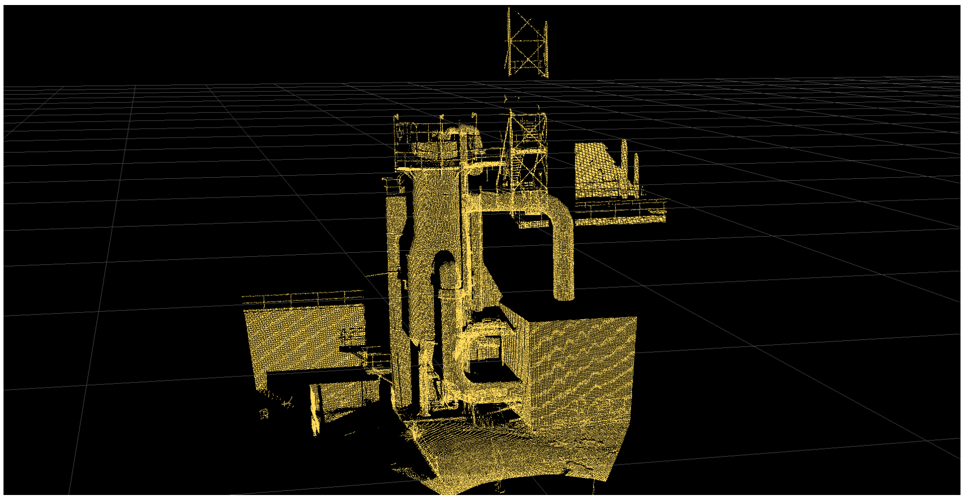
Source: AutoCAD Plant 3D

Given the precision needed, one popular approach is using LiDAR scanners to generate a point cloud representation of the real site (Figure 1.2). Then, this point cloud needs to be processed to reconstruct the CAD model; this process is referred to as reverse engineering (RAJA; FERNANDES, 2007). Most reverse engineering pipelines require a lot of user interaction to reconstruct the scene. This is particularly problematic when dealing with industrial plant sites, where the model can be extremely large and complex.

This work is part of an ongoing project that aims to recreate models of industrial sites with as little as possible user interaction, increasing the efficiency of this process. More specifically, we aim to improve the reconstruction of piping systems, which are present in a large variety of industrial segments. Our contributions include:

- A procedure based on Non-linear Least Squares to improve the estimated poses of piping components by finding pairs of connected components and minimizing the distances of their ports and the sum of their orientations (Chapter 5);
- An interface to improve the components' poses, where the user can, for example, establish connections between ports and specify that pairs of pipes must be parallel,

Figure 1.2 – Example of a point cloud captured from a real industrial site using a LiDAR scanner.



Source: The Authors

perpendicular or collinear (Chapter 5);

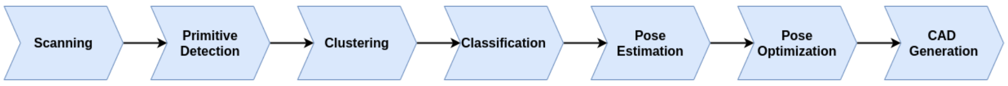
- A plug-in for AutoCAD Plant 3D to import the result of the reconstruction of a scene (Chapter 6);
- A dataset of models of piping system components, compiled from CAD parts provided by TraceParts, as well as industrial site scenes containing components from this dataset (Chapter 4);
- An algorithm to improve the delimitation of detected cylinders that extended through other components, such as elbows and tees, by estimating their medial axis and finding pairs of perpendicular intersecting pipes (Chapter 5);
- An algorithm for the automatic detection of the position and orientation of fittings' and flanges' ports (Chapter 4).

This thesis is structured as follows. Chapter 2 introduces the reverse engineering pipeline of our system. Chapter 3 discusses related works. Chapter 4 presents our dataset of piping system components and the scenes we created with them. In Chapter 5, we will talk about the algorithms we propose to improve the detection of cylinders and our procedure to improve the pose estimation of piping components. Chapter 6 is about the integration of our dataset with AutoCAD Plant 3D and the plug-in we developed to import the reconstructed scenes. In Chapter 7, we will show the results of our optimization-based method to improve the pose estimation of piping components. Lastly, in Chapter 8, we will talk about our conclusions and discuss potential future work.

2 REVERSE ENGINEERING PIPELINE

In this Chapter, we describe the steps of the reverse engineering pipeline of our software, which is an extension of (ARAUJO; OLIVEIRA, 2020c) and is named Point Cloud Editor. These steps are illustrated in Figure 2.1.

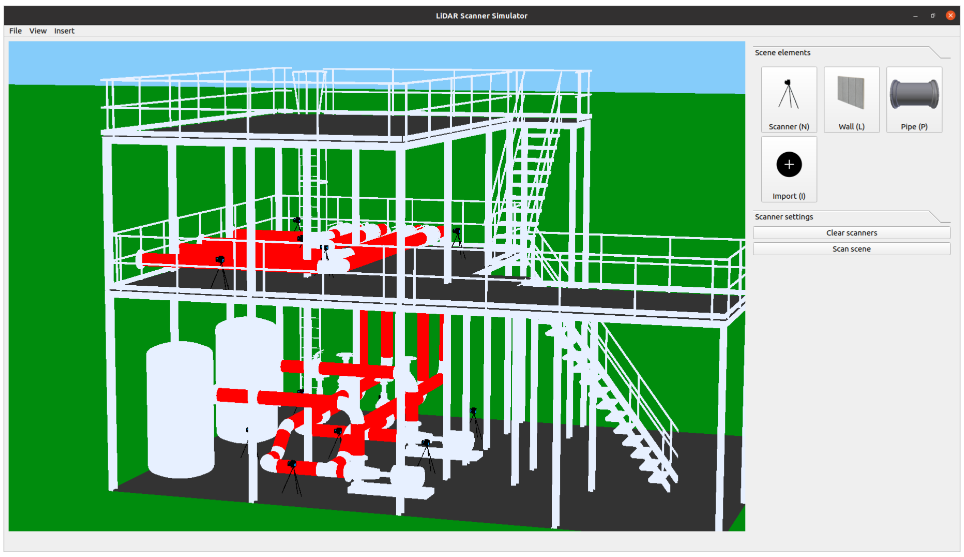
Figure 2.1 – A diagram of our reverse engineering pipeline.



Source: The Authors

The first step is scanning the scene. In a real scenario, this would be performed by scanning the industrial site using a LiDAR scanner. Since we do not have an actual laser scanner available for the project as well as access to industrial installations, for this work we scanned geometric models of industrial sites using a simulator of a LiDAR scanner, as described in Chapter 4. Its interface is shown in Figure 2.2, and Figure 2.3 presents the obtained point cloud.

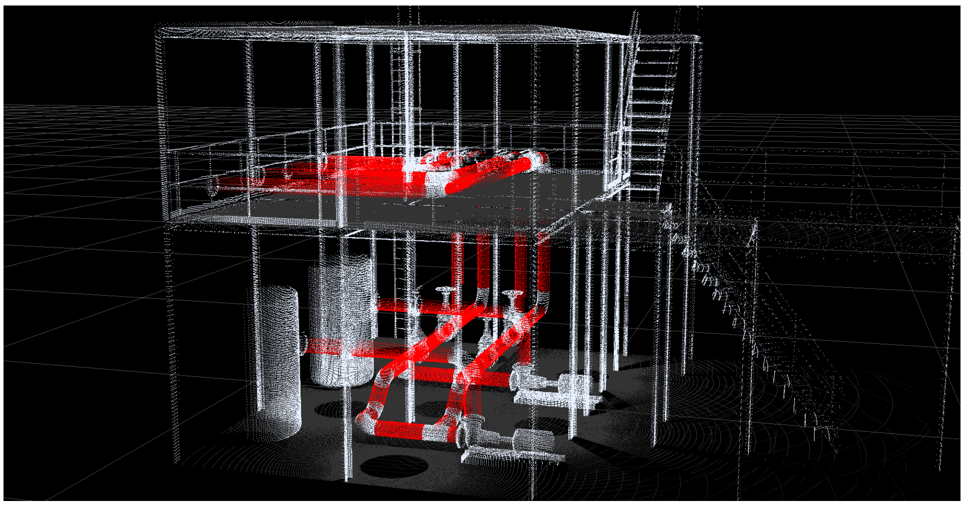
Figure 2.2 – The geometric model of a scene in our simulator of LiDAR scanner.



Source: The Authors

The next step is the detection of primitives that are usually present in industrial sites. These include planes, cylinders and structural elements, such as beams. Figure 2.4 shows an example of a structural element. After the automatic detection of these

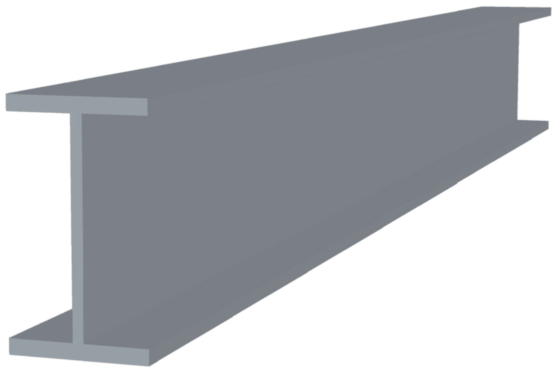
Figure 2.3 – The resulting point cloud after scanning the geometric model.



Source: The Authors

primitives, the user can still improve the results, for example, by selecting points and fitting a primitive to them. Figure 2.5 shows the detected planes, cylinders and structural elements in the point cloud shown in Figure 2.3.

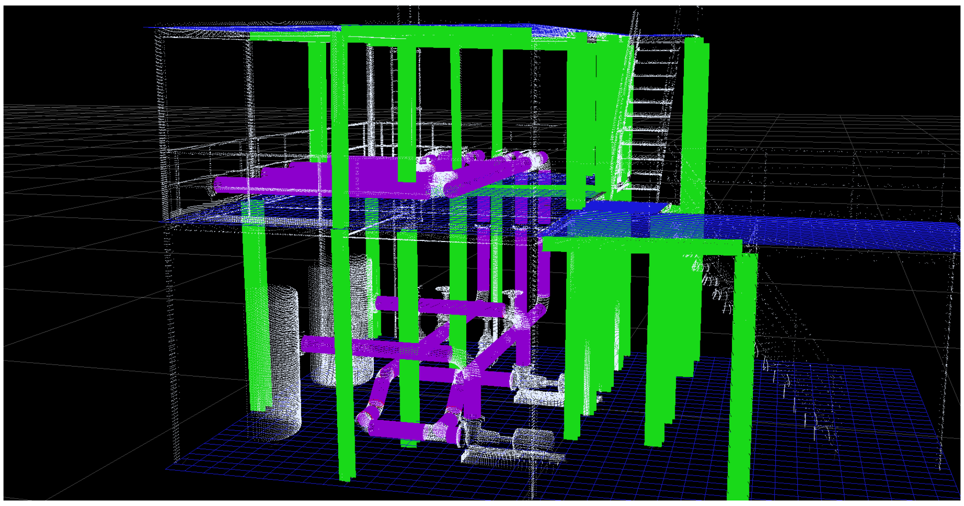
Figure 2.4 – An example of a structural element.



Source: The Authors

After logically removing the points that belong to the detected primitives, the points left are mostly part of piping system components, as shown in Figure 2.6. The next step is to find clusters of points that represent the piping components. These clusters are classified using a neural network into one of several classes, families, and sizes of

Figure 2.5 – The detected planes (in blue), cylinders (in purple) and structural elements (in green) of the scene, after user refinement.

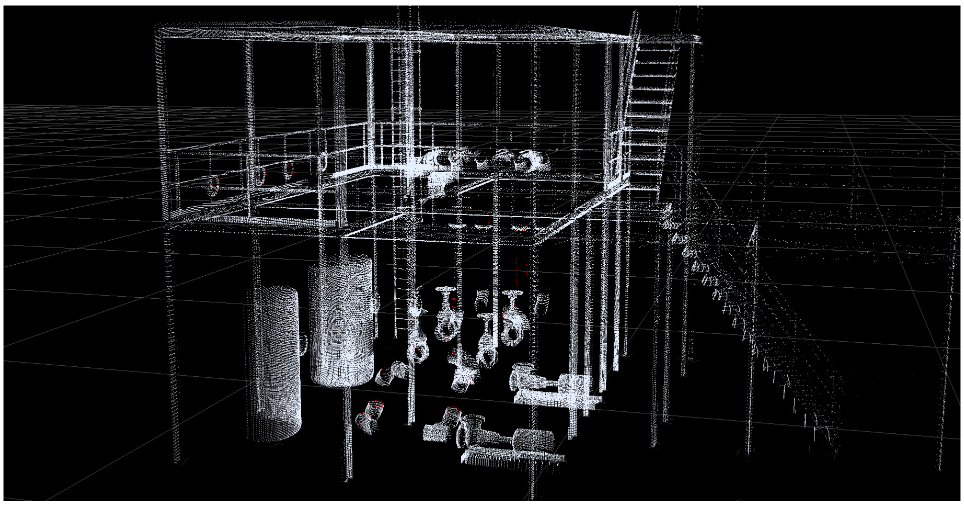


Source: The Authors

piping components. This neural network was trained using point clouds sampled from the geometric models of the components from our dataset, described in Chapter 4. Then we need to estimate the pose of each component, represented by a translation and rotation to be applied to the geometric model of the component. As the clustering, classification and pose estimation steps have been developed in parallel with this work, they are not integrated with our pipeline yet. Figure 2.7 shows preliminary results of the detection and pose estimation of piping components.

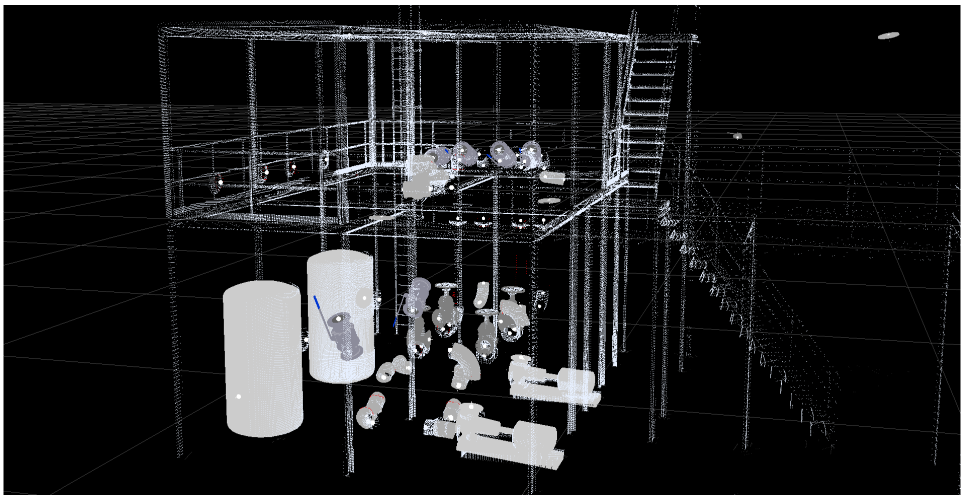
As the pose estimation is not perfect, the detected components might not be properly connected. In order to fix this misalignment, first we automatically find connections between ports and then optimize the components' and pipes' poses. Like in the primitive detection step, it is also possible to manually improve the optimization step, for instance, by establishing new connections between ports, or determining that a pair of cylinders must be perpendicular. Lastly, the detected planes, pipes, piping system components and beams are exported to AutoCAD Plant 3D, where the user can modify the reconstructed scene as needed. The pose optimization and CAD generation are the focus of this work.

Figure 2.6 – The remaining points after primitive detection, that belong mostly to piping system components.



Source: The Authors

Figure 2.7 – The detected piping system components before optimization, using a preliminary result of the pose estimation.



Source: The Authors

3 RELATED WORK

This Chapter briefly discusses other works related to primitive detection, optimization in computer graphics, image processing, and computational photography, and to reverse engineering of CAD models.

3.1 Primitive Detection

Planes and cylinders represent a significant portion of industrial plant sites, so primitive detection is an important step when reconstructing their models. There are several techniques for the detection of planes and cylinders from point clouds, and they can be divided into three categories: Hough Transform (LIMBERGER; OLIVEIRA, 2015) (RABBANI; HEUVEL, 2005), RANSAC (SCHNABEL; WAHL; KLEIN, 2007) (JIN; LEE, 2019), and region growing (ARAUJO; OLIVEIRA, 2020b) (TRAN; CAO; LAURENDEAU, 2015). For the detection of planes, our system uses the algorithm proposed in (ARAUJO; OLIVEIRA, 2020b), and for the detection of cylinders, the algorithm proposed in (ARAUJO; OLIVEIRA, 2020a). For the detection of structural elements, our system uses an algorithm developed in parallel with this work, which has not been published yet.

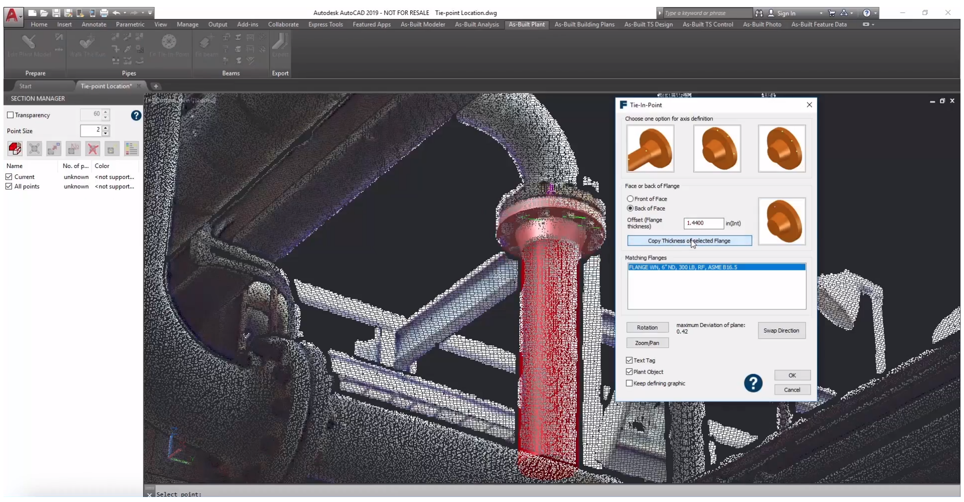
3.2 Optimization in Computer Graphics

Optimization has been widely used in the fields of computer graphics, image processing and computational photography for different purposes, such as Poisson image editing (PEREZ; GANGNET; BLAKE, 2003), shape-from-shading (WU et al., 2014), as-rigid-as-possible warping (SORKINE; ALEXA, 2007), blind deconvolution (FERGUS et al., 2006), and image denoising (ZORAN; WEISS, 2011). It has also been used for transient imaging (HEIDE et al., 2013a) and to achieve high-quality computational imaging while using simple lenses (HEIDE et al., 2013b). (DEVITO et al., 2017) and (HEIDE et al., 2016) created Domain Specific Languages to model Graphics and Imaging optimization problems while generating efficient implementation.

3.3 CAD Related

In the context of reconstructing CAD models from point clouds, Li et al. (LI et al., 2019) proposed a method for fitting primitives to point clouds scanned from individual parts, based on supervised learning. Ivson and Celes (IVSON; CELES, 2014) use a shape matching algorithm to identify repeated instances of geometry in CAD models to reduce memory usage. However, their solution does not work when the topology of the meshes of the repeated instances are different. To overcome this limitation, Figueiredo, Ivson, and Celes (FIGUEIREDO; IVSON; CELES, 2021) uniformly samples point clouds from the models and use a deep learning-based approach to find repeated geometry. (KOCH et al., 2019) created a dataset of CAD models for geometric deep learning.

Figure 3.1 – Screenshot of the interface of FARO’s As-Built for AutoCAD Software.

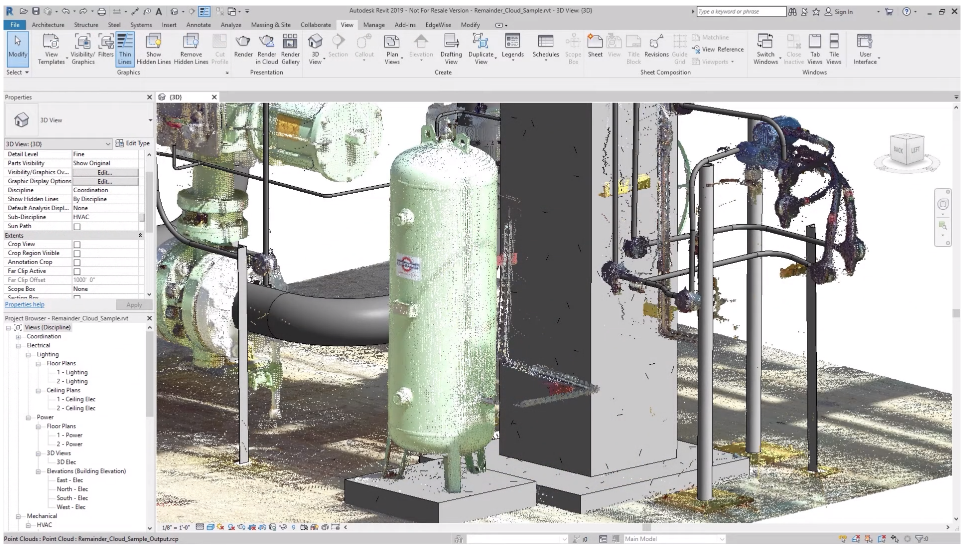


Source: FARO

3.4 Other Reverse Engineering Software

There are a few available reverse engineering software for reconstructing CAD models of industrial sites from point clouds. FARO’s *As-Built for AutoCAD Software* (FARO, 2021) extends AutoCAD and provides several tools for reverse engineering, like semi-automatic reconstruction of piping systems; Figure 3.1 shows its interface. They also provide a similar tool for Autodesk Revit. ClearEdge3D’s *EdgeWise* (CLEAREDGE3D, 2021) can be used to automatically or semi-automatically detect several elements present in an industrial site, including pipes and structural elements. However, the software does not handle some essential piping system elements, such as valves and tanks. Figure 3.2 shows a scene reconstructed using *EdgeWise*.

Figure 3.2 – Semi-automatically reconstructed scene using EdgeWise. Here we can see that tanks and valves (shown in green) remain as point clouds, so the user can reconstruct them using another method.



Source: ClearEdge3D

Autodesk *ReCap Pro* (AUTODESK, 2021d) is a scanning software that allows importing point clouds into other CAD software from Autodesk, such as AutoCAD and Revit. Besides, AutoCAD also provides some commands to help the user create geometry from point clouds, like extracting the centerline of a cylinder or finding the edge between two planar segments. 3DReshaper (TECHNODIGIT, 2020) is a software for processing point clouds that also provide some tools for the reconstruction of surfaces and curves. Lastly, there is also software to convert scans from a single object to a CAD model, such as Geomagic (3DSYSTEMS, 2021), QUICKSURFACE (GOMEASURE3D, 2021) and XTract3D (POLYGA, 2021).

3.5 Summary

This Chapter discussed related works in several fields, specially in reverse engineering. As we saw, other works suffered from significant limitations, such as working with individual components, and requiring a lot of user interaction.

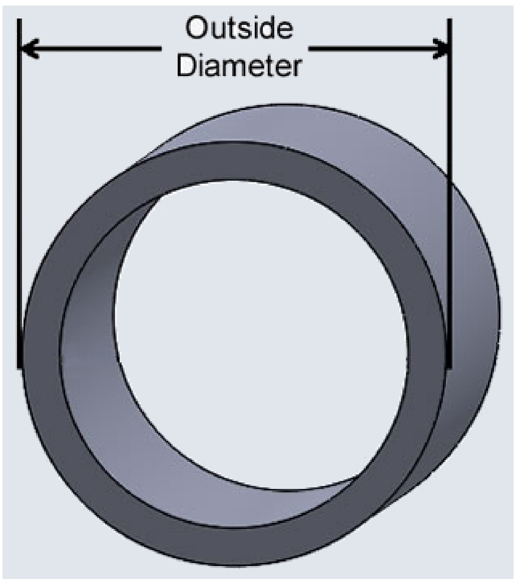
4 SYNTHETIC INDUSTRIAL SCENES

Manufacturers of piping system components in different countries follow different standards. It would be desirable to reconstruct the CAD model of industrial sites regardless of standards. However, it would be considerably hard to determine what model is being used in a scene. Because of that, we first collected models of piping system components to create a dataset, and try to reconstruct scenes that contain only these components. This Chapter describes our new dataset and the industrial sites we have modeled using it.

4.1 Piping Concepts

Before introducing our dataset, we briefly explain some concepts about piping systems that are used throughout this thesis. First, pipes and piping components have nominal diameters associated with them. A nominal diameter consists of the letters DN followed by an unsigned integer that indicates the outside diameter of the end connections. Figure 4.1 illustrates the outside diameter of a pipe, and Table 4.1 shows the nominal diameters between DN10 and DN300 according to the ISO 6708 standard, obtained from The Engineering Toolbox[1]. The nominal diameters of all components from our dataset are inside this range.

Figure 4.1 – The outside diameter of a pipe.



Source: PVC Vent Screens

[1]https://www.engineeringtoolbox.com/nps-nominal-pipe-sizes-d_45.html

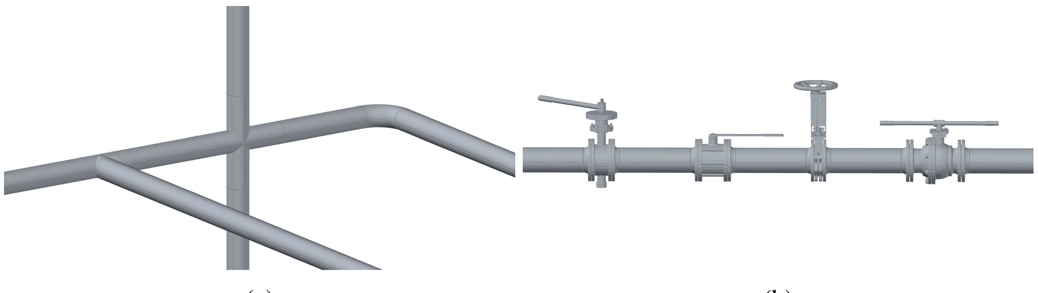
Table 4.1 – List of classes of models with their respective type, number of families and number of models.

DN	*Outside Diameter*
DN10	17.2
DN15	21.3
DN20	26.9
DN25	33.7
DN32	42.4
DN40	48.3
DN50	60.3
DN65	76.1
DN80	88.9
DN100	114.3
DN125	139.7
DN150	168.3
DN200	219.1
DN250	273.0
DN300	323.9

Source: The Engineering ToolBox

Piping components can be classified into fittings, valves, reducers, and flanges. Fittings control the direction of the piping system, and valves control their flow. Figure 4.2 shows examples of valves and fittings. Reducers are used when it is necessary to reduce the nominal diameter of a pipe. Flanges are used to connect flanged components to pipes; in our dataset, all valves are flanged. All fittings and reducers from our dataset are butt-welded, that is, they are directly connected to pipes. Figure 4.3 shows the difference between butt-welded and flanged connections.

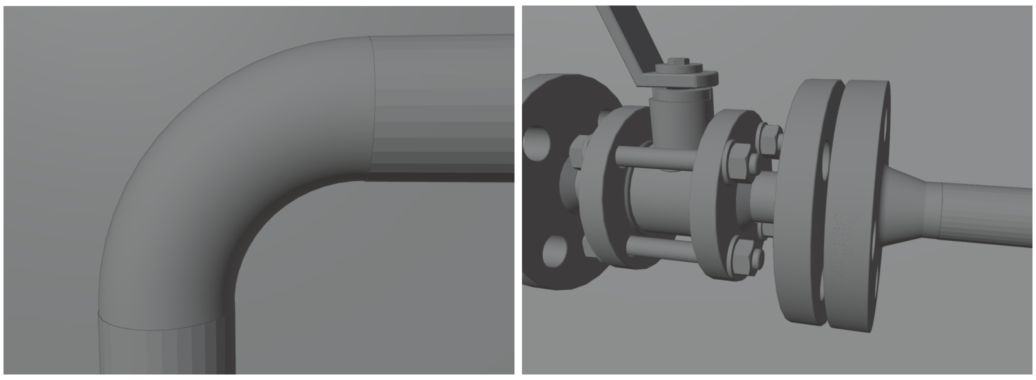
Figure 4.2 – Simple piping systems showing examples of fittings (a) and valves (b).



(a) (b)

Source: The Authors

Figure 4.3 – Examples of butt-welded connections (a), where the pipe is directly welded to the component, and an example of flanged connection (b), where the pipe is connected to a valve through a flange.



(a) (b)

Source: The Authors

4.2 Piping Components Dataset

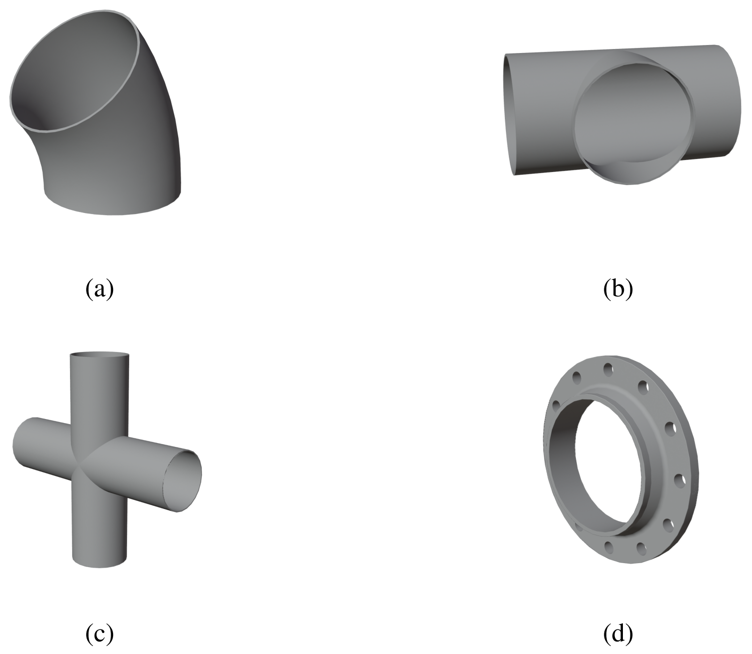
For our dataset of piping system components, we have downloaded models from TraceParts (TRACEPARTS, 2021), a digital library that contains CAD files from several manufacturers of different areas. A few extra models, such as pumps and vessels, were obtained from AutoCAD Plant 3D (Figure 4.6). Models can be hierarchically structured into classes, which in turn are formed by groups of families. Inside each family the models only differ by their sizes. Each class also has a type associated to it. For instance, *gate valve* and *90º elbow* are examples of classes, and their types are, respectively, *valve* and *fitting*. Figure 4.4 shows examples of fittings and a flange from our dataset, and Figure 4.5 shows examples of valves from our dataset. Besides, to guarantee that the models in our dataset would fit properly, we chose to include only models that follow DIN/ISO standards. A list of the classes from our dataset can be seen in Table 4.2.

4.2.1 Detecting Component Ports

Another necessary piece of information for each component is the position and orientation of its ports. A port is the point where these components are connected to a pipe or another component. This information is used both in the optimization step and to instantiate our models in a CAD software.

As manually detecting ports is very time consuming, we tried to automate this

Figure 4.4 – Models of a 45º elbow (a), a tee (b), a cross (c) and a welding neck flange (d) downloaded from TraceParts.



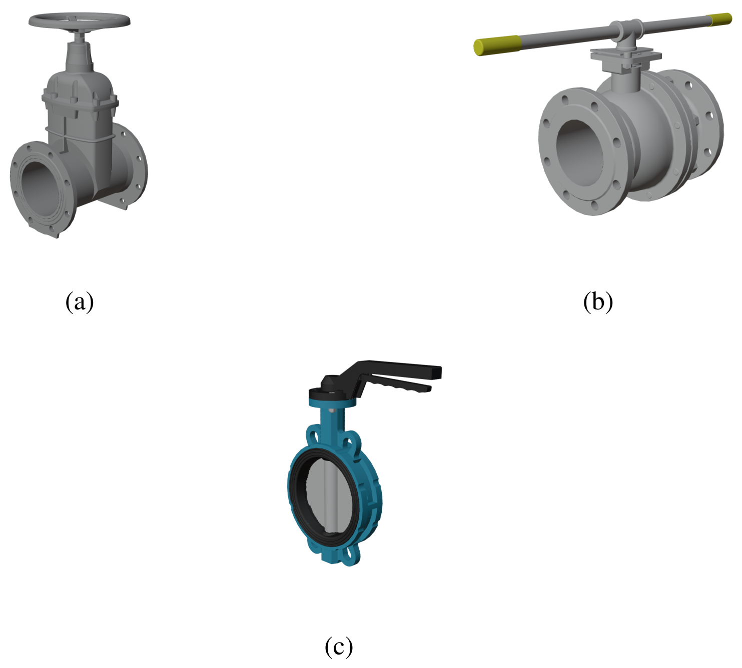
Source: TraceParts

process. In order to do that, we used geometry processing to find circles around the ports, using the OpenMesh (BOTSCH et al., 2002) library to handle the data structure of the mesh. The key idea is to find edges in which the normals of the faces that contain them form an angle of 90º, which is shown in Figure 4.7. We then need to find cycles of edges and check if these cycles are circles. To check if a cycle corresponds to a circle, we first estimate a circle from three vertices, and then calculate the average squared distance from the points of the edges of the cycle to the estimated circle; if this value is below a threshold, the cycle is considered to be a circle.

Lastly, as there can be multiple circles around a single port, as seen in Figure 4.7, we also need to merge circles whose centers are close. After that, every circle gives origin to a port, where the port's position is equal to the center of the circle, and its orientation is estimated computing the normal of the triangle defined by three random points picked from the circle. We also need to guarantee that the port's orientation is pointing away from the other ports. To ensure that a port $p0$ points away from the position of a port $p1$, we need the angle between the orientation of $p0$ and the vector $p0.position - p1.position$ to be lower than 90º. If this is not the case, we need to flip the orientation of $p0$.

This algorithm worked well with most fittings, flanges, and reducers; we can see

Figure 4.5 – Models of a gate valve (a), a two piece ball valve (b) and a wafer butterfly valve (c) downloaded from TraceParts.



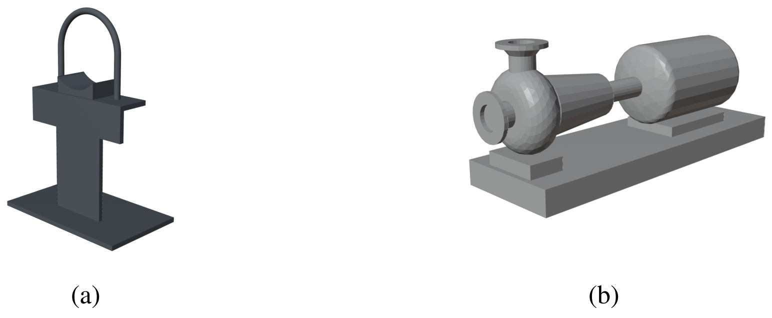
Source: TraceParts

an example in Figure 4.8. However, as the variability between different models of valves is significant, we were not able to automatically detect the ports for most of the valve models. The ports of these models were manually detected using the software Blender (COMMUNITY, 2018) to select the points of the circles around the ports. The result of the port detection is saved to an XML file; Figure 4.9 shows an example of such a file.

4.3 The AutoCAD Plant 3D

There are a few software for AEC professionals specifically developed to model industrial sites, such as AVEVA E3D Design (AVEVA, 2021) and Intergraph Smart 3D (HEXAGON, 2021). In our project, we decided to use AutoCAD (AUTODESK, 2021a) and, more specifically, its toolset to model industrial sites, AutoCAD Plant 3D (AUTODESK, 2021c). This software is widely used in industry, has a free educational license, and provides an SDK for developing plug-ins to extend it.

Figure 4.6 – Models of a pipe support (a) and a centrifugal pump (b) obtained from AutoCAD Plant 3D.



Source: AutoCAD Plant 3D

4.3.1 Modeling Scenes with AutoCAD Plant 3D

To test our pipeline of reverse engineering, we created five scenes containing the components of our dataset using AutoCAD Plant 3D; in Chapter 6, we will explain how we are able to instantiate the components from our dataset inside Plant 3D. Having scenes built using components from our dataset is important as we can only properly reconstruct them. Figure 4.10 shows two examples of synthetic scenes modeled by us. After modeling the scenes with Plant 3D, we exported them to the Wavefront (.obj) file format using a third-party plug-in (VISIONWORKPLACE, 2021), so that we could scan these scenes using our simulator of LiDAR scanner.

4.4 LiDAR Scanner Simulator

There are several approaches to scan the geometry of a scene. For instance, one could sample points on the surface of the faces of the models in the scene proportionally to the area of each face. However, this approach does not handle occlusions between objects and provides components regularly sampled throughout the scene, whereas in real scenarios, components closer to the scanner are more densely sampled than those further away.

To obtain a point cloud from the modeled scenes, we use the LiDAR Simulator, initially developed as part of (ARAUJO; OLIVEIRA, 2020c). Using this software, one can position scanners around the scene represented by its geometric model. This scene is then rendered from the point of view of each scanner using OpenGL, and then sampled

Table 4.2 – List of classes of models with their respective type, number of families and number of models.

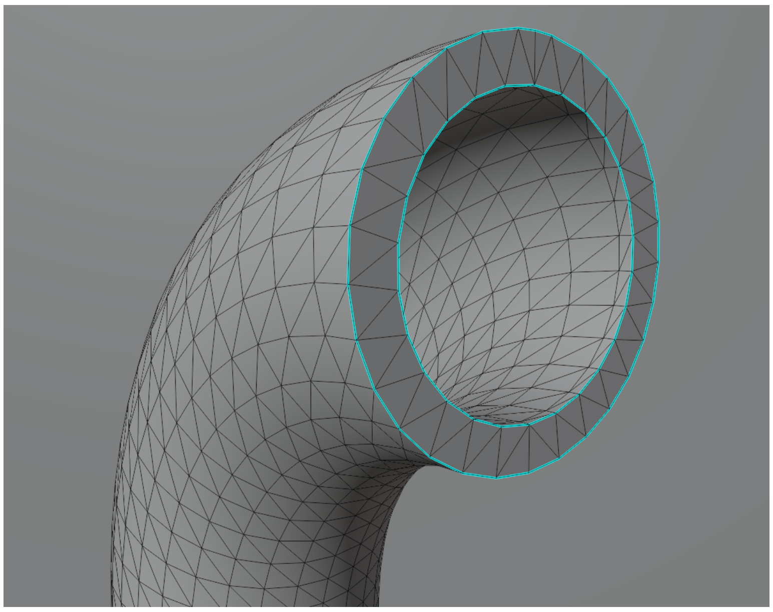
Class	*Type*	*Number of Families*	*Number of Models*
Cap	Fitting	1	15
Cross	Fitting	1	12
Elbow 45	Fitting	1	15
Elbow 90	Fitting	1	13
Tee	Fitting	1	14
Flanged Butterfly Valve	Valve	1	6
Gate Valve	Valve	2	19
Knife Gate Valve	Valve	2	11
Three Piece Ball Valve	Valve	2	14
Two Piece Ball Valve	Valve	3	18
Wafer Ball Valve	Valve	2	16
Wafer Butterfly Valve	Valve	4	33
Blind Flange	Flange	1	13
Flat Flange	Flange	1	12
Welding Neck Flange	Flange	1	14
Concentric Reducer	Reducer	1	27
Eccentric Reducer	Reducer	1	25
Reducing Tee	Reducer	1	25
Pipe Support	Others	2	3
Pump	Others	2	5
Vessel	Others	1	5

producing a 3D point cloud. Figure 4.11 shows the resulting point cloud after scanning a scene using the LiDAR Simulator.

A few minor modifications were made in the scanner for this work. Before, we were calculating the position of the point using the top left corner of the pixel. Because of that, the observations of different scanners in the same scene would not be properly aligned. To fix this issue, now we calculate the position of the point using the center of the pixel. We also created a new file format to save the position of the scanners and the configurations used to scan the scene, allowing us to reproduce how a point cloud was scanned.

One last detail worth mentioning is that we also need the scene being scanned to be in meters. When scanning a scene in millimeters, we noticed that the position of the points further away from the scanner was not very precise. This happens because the z-buffer has a larger precision for objects that are closer to the camera. However, after scanning, we need to convert the scene back to millimeters, as the next steps of our pipeline expect the scene to be in millimeters.

Figure 4.7 – A 90º elbow, with the edges our algorithm searches for highlighted in light blue. Notice how the highlighted edges define two circles centered at one of the component's ports.

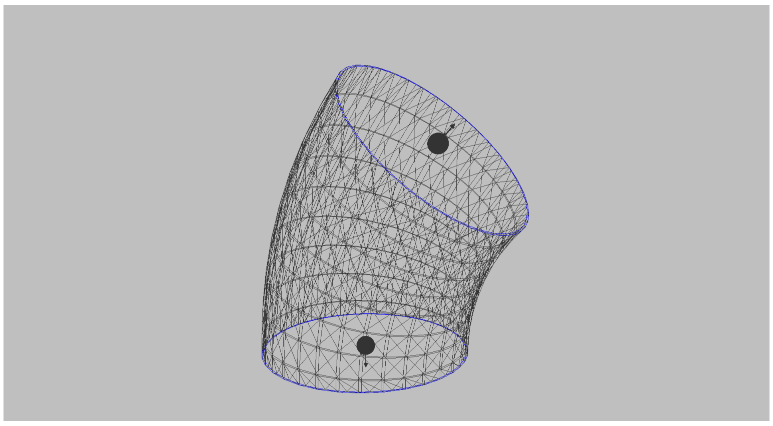


Source: The Authors

4.5 Summary

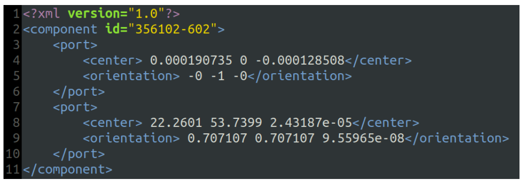
This Chapter described our dataset of piping system components, the modeling of scenes using AutoCAD Plant 3D, and the generation of point clouds using our simulator of a LiDAR scanner. We also presented an algorithm for detection of the position and orientation of ports of the components of our dataset that only worked for fittings, flanges, and reducers.

Figure 4.8 – Automatically detected ports from a model of a 45º elbow; the edges highlighted in blue define the circles used to estimate the position and orientation of the ports.



Source: The Authors

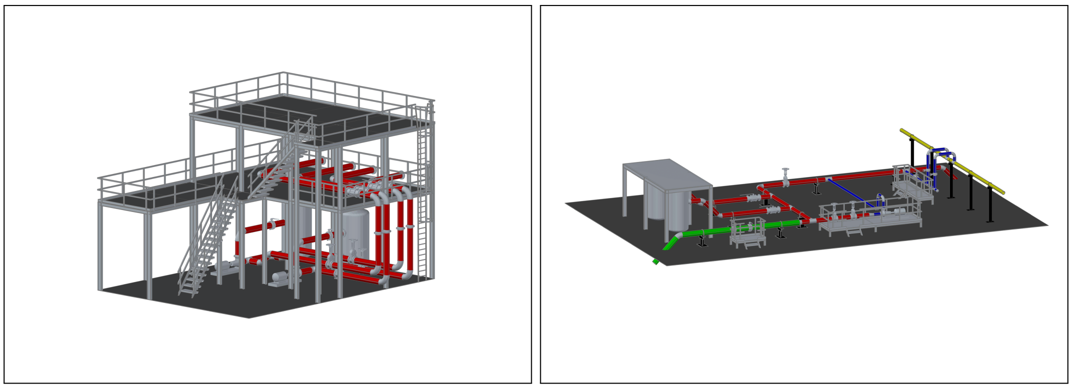
Figure 4.9 – Example of an XML file that contains the result of the port detection of an Elbow 45.



```xml
<?xml version="1.0"?>
<component id="356102-602">
    <port>
        <center> 0.000190735 0 -0.000128508</center>
        <orientation> -0 -1 -0</orientation>
    </port>
    <port>
        <center> 22.2601 53.7399 2.43187e-05</center>
        <orientation> 0.707107 0.707107 9.55965e-08</orientation>
    </port>
</component>
```

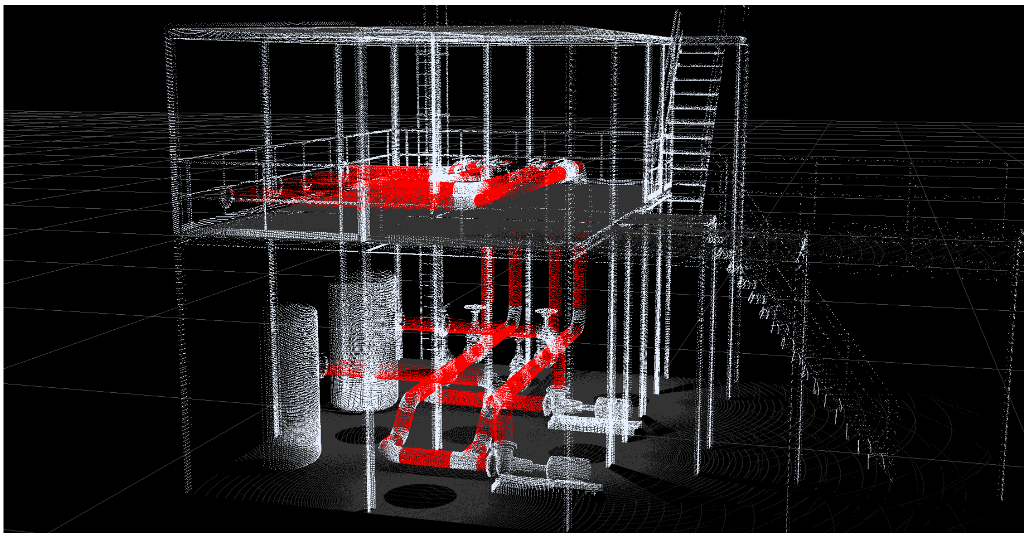
Source: The Authors

Figure 4.10 – Examples of scenes modelled with AutoCAD Plant 3D using our dataset of piping system components.



Source: The Authors

Figure 4.11 – Point cloud of a scene we have modelled using AutoCAD Plant 3D, scanned using our LiDAR Simulator.



Source: The Authors

5 OPTIMIZING THE ALIGNMENT OF PIPING SYSTEM COMPONENTS

After the detection and pose estimation of pipes and other piping system components, these components might not connect properly, as pose estimation is not perfect and the point clouds from which they are detected are inherently noisy. In this Chapter, we introduce two refinement procedures for the cylinder detection step. Then, we present an optimization-based method to improve the alignment of the piping system components. Lastly, we explain how the user can further improve this result using our software.

5.1 Cylinder Detection Refinement

Using butt-welded fittings, we noticed that sometimes the detected pipes advanced through some fittings. This happens because the transition between the pipe and the fitting is smooth. We propose two improvements to the cylinder detection algorithm to delimit the pipes better.

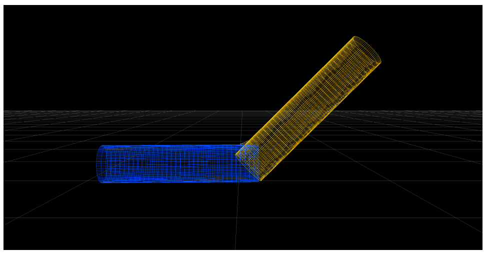
5.1.1 Cylinder Curvature Refinement

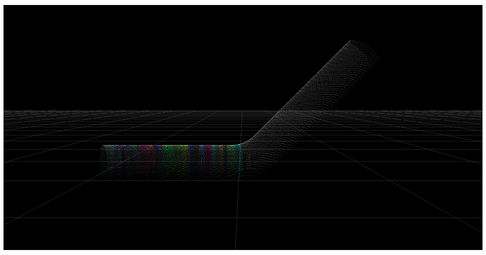
One recurring problem we noticed when detecting cylinders in our test scenes is that a pipe would extend beyond its limit and into the elbow it is connected to. Two issues emerge from this: there are fewer points in the remaining cluster to classify it as an elbow and to estimate its pose; and the estimated orientation and length of the pipe become slightly wrong.

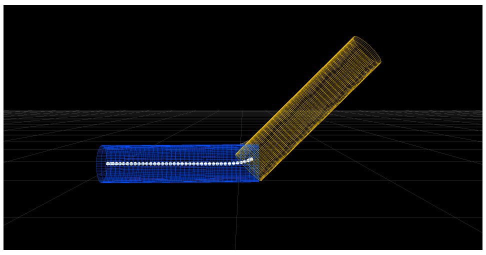
To solve this problem, we developed an algorithm based on (JIN; LEE, 2019), that detects spheres along the axis of the detected cylinders to estimate their medial axes and use this information to determine where the cylinders should end. A pseudocode for this procedure can be seen in Algorithm 1. Figure 5.1 illustrates the steps of the Algorithm. This function is executed for every detected cylinder. First, the points detected as being part of the cylinders are divided into partitions according to their projections into the cylinder's axis (Line 6). Then, in Line 9, we use RANSAC to detect a sphere in each partition. To estimate the center and radius of a sphere from two points and their respective normals, we use the method described in (SCHNABEL; WAHL; KLEIN, 2007).

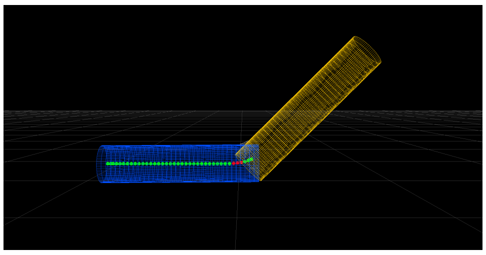
After that, in Line 14, we apply a box filter to the centers of the spheres. Then we

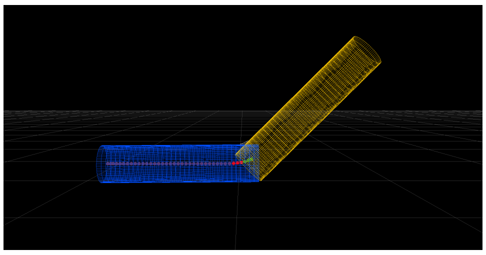
Figure 5.1 – The following images display a scene containing an elbow and two cylinders, but for which the cylinder detection process returned two intersecting cylinders. The images illustrate the steps of our algorithm applied to the blue cylinder. The detected spheres are represented by their centers. Here we can see the original result of the cylinder detection (a), the partitions of the blue cylinder (b), the detected spheres, after filtering (c), the valid spheres highlighted in green, after estimating the curvature at their centers (d), the sets of valid spheres (e) and the result after executing our algorithm (f).

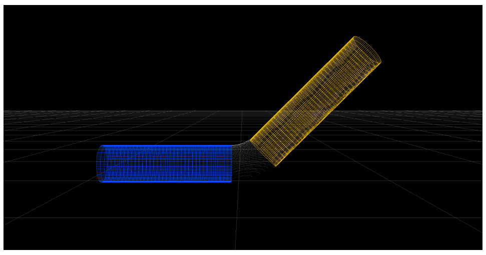

(a)


(b)


(c)


(d)


(e)


(f)

Source: The Authors

compute the curvature of the medial axis at each sphere by finding its neighborhood and computing the sum of the distances of the centers of the spheres to the line between the first and last sphere of the neighborhood. If this curvature is lower than the radius of the original cylinder multiplied by a threshold, the sphere is added to the set of valid spheres (Lines 16-21).

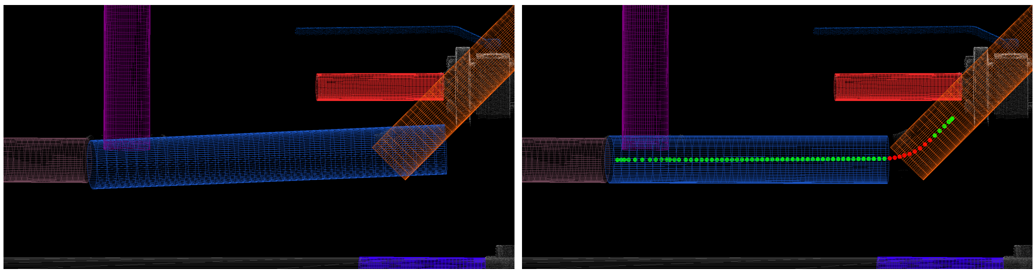
We then need to find sets of consecutive, valid spheres, and estimate the orientation of each set by computing the PCA of the centers of the spheres. Then, we merge sets whose orientations are approximately the same (Line 25). This step is necessary as the sphere detection may produce a result that is significantly distant from the medial axis of the cylinder, which would make some of the spheres around it invalid. If this happens in the middle of the cylinder, there would be two disjoint sets of valid spheres in the same cylinder.

Lastly, if there is no set of valid spheres, the cylinder is removed. Otherwise, we find the largest set and fit the cylinder to the points that belong to the partitions from which the spheres of the set were detected (Lines 27 and 28). Figure 5.2 shows the result of this algorithm.

Algorithm 1: Cylinder Curvature Refinement

```
Input: $pc \in \mathbb{R}^{N\times 3}$          ▷ Point cloud of the cylinder
       $cc \in \mathbb{R}^3$                    ▷ Center of the cylinder
       $ca \in \mathbb{R}^3$                    ▷ Axis of the cylinder
       $cl \in \mathbb{R}$                      ▷ Length of the cylinder
       $cr \in \mathbb{R}$                      ▷ Radius of the cylinder
Function cylinderCurvatureRefinement($pc, cc, ca, cl, cr$):
    $partitions \longleftarrow createPartitions(pc, cc, ca, cl, cr)$;
    $spheres \longleftarrow []$; $sphereSets \longleftarrow []$;
    for $i \longleftarrow 0; i < len(partitions); i \longleftarrow i + 1$; do
        $sc, sr \longleftarrow processPartition(part)$;
        $spheres.add(i, sc, sr)$;
    end
    $filterSpheres(spheres)$;
    $validSpheres = []$;
    while $i \longleftarrow len(spheres) - 1; i >= 0; i \longleftarrow i - 1$; do
        $j, sc, sr \longleftarrow spheres[i]$;
        $neighbourhood \longleftarrow getNeighbourhood(j, spheres)$;
        if $isStraightLine(neighbourhood, cr)$; then
            $validSpheres.add(spheres[i])$;
        end
    end
    $sphereSets \longleftarrow findSphereSets(spheres, validSpheres)$;
    if $len(sphereSets) = 0$; then
        $removeCylinder()$;
    else
        $largestSet \longleftarrow findLargestSet(sphereSets)$;
        $fitCylinder(largestSet, cc, ca, cl, cr)$;
    end
End Function
```

Figure 5.2 – Before (a) and after (b) executing the curvature refinement of the blue cylinder. In (b) we also have the detected spheres represented by their centers. The sphere centers in green represent the points in which the curvature is below a specific threshold, whereas the sphere centers in red represent points in which the curvature is above this threshold. Although there are two disjoint sets of consecutive, valid spheres, our algorithm correctly selects the largest one to delimit the cylinder.



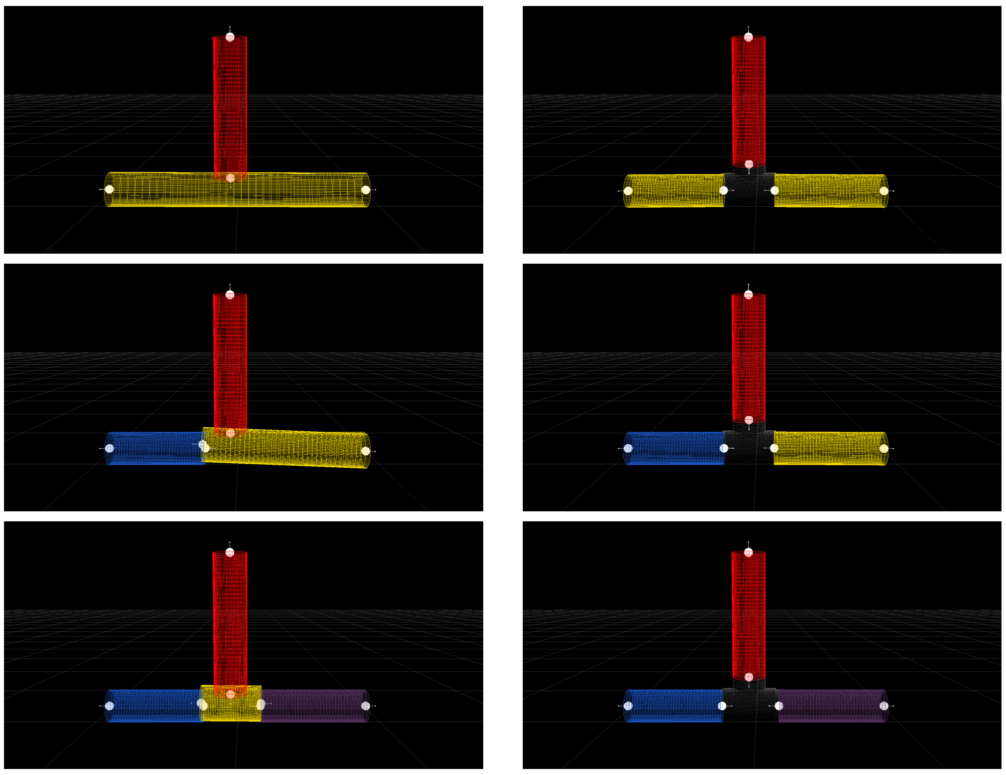
Source: The Authors

5.1.2 Perpendicular Cylinders Refinement

Another issue we noticed is that sometimes the detected pipes extended through tees or crosses, which leads to similar problems to the previous case. To solve that, we propose a simple algorithm that detects close perpendicular pipes and removes the intersection points between them from both cylinders. A pseudocode for this procedure can be seen in Algorithm 2. This function is executed once for the entire scene. First, we find pairs of perpendicular cylinders, that is, cylinders in which the dot product of their orientations is approximately zero, and push them into a queue (Lines 3-8). Then, while this queue is not empty, we get the first pair of cylinders from the queue and, if the distance between the axes of the cylinders is greater than a threshold, we continue to the next element of the queue (Lines 11-13). Otherwise, in Lines 14 and 15 we remove from both cylinders the samples close to the intersection point of the cylinders, that is, the point in the axis of one of the cylinders closest to the axis of the other cylinder. If the intersection point is approximately in the middle of one of these cylinders, it will be split in two; if it is near one of the ends of a cylinder, it will result in a single cylinder; and if one cylinder is small enough, it will be removed. Figure 5.3 shows examples of these three scenarios.

To calculate the distance between the axes of the cylinders and obtain their intersection points, we use the GeometricTools library (EBERLY, 2020). The size of the intersection region is directly proportional to the largest radius between the two cylinders

Figure 5.3 – A scene containing one tee and three cylinders. The cylinders in the left column were manually detected, and the right column shows the result after executing our algorithm. In the first row, the intersection point between the red and yellow cylinders is approximately in the middle of the yellow cylinder, so it is split in two. In the second row, the intersection point is near one end of the yellow cylinder, so it results in a single cylinder. In the last row, the yellow cylinder is removed.



Source: The Authors

and inversely proportional to the cylinder's length.

Lastly, we must traverse the queue of perpendicular cylinders, replacing the cylinders being processed by the resulting cylinders after removing the points from the intersection region (Lines 16 and 17). This is necessary as one cylinder might initially be perpendicular to more than one cylinder. Figure 5.4 shows the result of this algorithm.

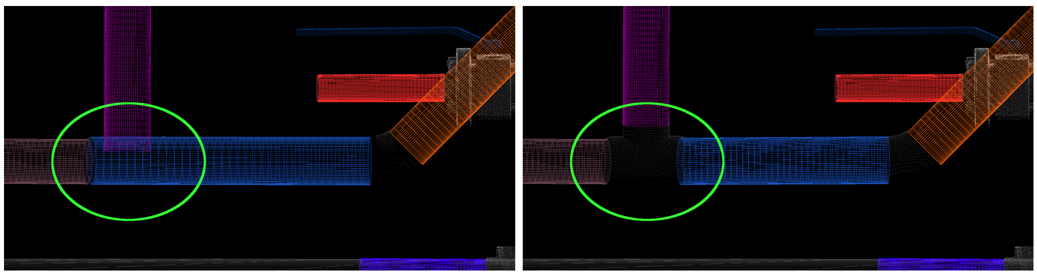
Algorithm 2: Perpendicular Cylinders Refinement

```
Input: $cylinders$    ▷ Array with the parameters and inliers of the cylinders
Function perpendicularCylindersRefinement($cylinders$):
    $perpendicularCylinders = []$;
    for $i \longleftarrow 0; i < len(cylinders) - 1; i+ = 1$; do
        for $j \longleftarrow i; j < len(cylinders); j+ = 1$; do
            if $areCylindersPerpendicular(cylinders[i], cylinders[j])$;
             then
                $perpendicularCylinders.add(cylinders[i], cylinders[j])$;
            end
        end
    end
    while $\neg perpendicularCylinders.empty()$; do
        $c0, c1 \longleftarrow perpendicularCylinders.front()$;
        $perpendicularCylinders.pop\_front()$;
        if $areCylindersClose(c0, c1)$; then
            $c00, c01 \longleftarrow removeIntersectionPoints(c0, c1)$;
            $c10, c11 \longleftarrow removeIntersectionPoints(c1, c0)$;
            $updateCylindersPairs(c0, c00, c01, perpendicularCylinders)$;
            $updateCylindersPairs(c1, c10, c11, perpendicularCylinders)$;
        end
    end
End Function
```

5.2 Detecting the Connection between Piping System Components

After having a better delimitation of the pipes and using the estimated poses of the other components, we now need to find connections among the piping system components in the scene. To achieve that, we developed a greedy algorithm that searches, for every port, a corresponding port that is closest to it. Besides, the distance between the ports must be below a certain threshold, and the angle between the orientations of the ports must be above some angular threshold.

Figure 5.4 – Before (a) and after (b) executing the perpendicular cylinders refinement of the blue and purple cylinders; the green ellipse highlights the region where the cylinders intersect. As the intersection point is near the end of both cylinders, after removing the intersection points, each cylinder generates a single cylinder.



Source: The Authors

5.3 The Ceres Solver

As an optimization library for our system, we chose the Ceres Solver (AGARWAL; MIERLE et al., 2010). This library can be used to solve non-linear least squares problems, which are represented as:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}\sum_{i} \rho_i(\|f_i(x_{i_1},\dots,x_{i_k})\|^2) \\ \text{s.t.} \quad & l_j \le x_j \le u_j, \end{aligned} \tag{5.1}$$

where $f_i(x_{i_1},\dots,x_{i_k})$ represents a cost function, $\{x_{i_1},\dots,x_{i_k}\}$ is the parameter block of the cost function f_i, l_j is the lower bound of the parameter block x_j, u_j is the upper bound of the parameter block x_j, and ρ_i represents a loss function. As we use the identity function as our loss function and we set no upper or lower bounds for any of our parameter blocks, we have an unconstrained non-linear least squares problem. We used the default options from Ceres for the solver. We also used automatic differentiation for our cost functions.

5.3.1 Our Objective Function

We now describe the function that we try to minimize. Let C be the set of poses of piping components, with $(cp, cr) \in C$, where $cp \in \mathbb{R}^3$ is the position of the component and $cr \in \mathbb{R}^4$ is a quaternion that represents the rotation of the component. Let P be the set of pipes, with $(pp, pa, pl) \in P$, where $pp \in \mathbb{R}^3$ is the position of the pipe, $pa \in \mathbb{R}^3$ is the axis of the pipe and $pl \in \mathbb{R}$ is the length of the pipe. Our objective function can be represented as:

$$\begin{aligned}
\min_{(cp,cr)\in C,(pp,pa,pl)\in P} \quad & \sum_{(p,c)\in CPC} \|CPCD(c.cp, c.cr, p.pp, p.pa, p.pl)\|^2 + \|CPOS(c.cr, p.pa)\|^2 \\
& + \sum_{(c_0,c_1)\in CCC} \|CCCD(c_0.cp, c_0.cr, c_1.cp, c_1.cr)\|^2 + \|CCOS(c_0.cr, c_1.cr)\|^2 \\
& + \sum_{(p_0,p_1)\in CollPR} \|CollP(p_0.pp, p_0.pa, p_1.pp, p_1.pa)\|^2 \\
& + \sum_{(p_0,p_1)\in PerPR} \|PerP(p_0.pa, p_1.pa)\|^2 \\
& + \sum_{(p_0,p_1)\in ParPR} \|ParP(p_0.pa, p_1.pa)\|^2, \qquad (5.2)
\end{aligned}$$

where $CPC \in C \times P$ is the set of connections between components and pipes, $CCC \in C \times C$ is the set of connections between two components, $CollPR \in P \times P$ is the set of collinearity restrictions between pipes, $PerPR \in P \times P$ is the set of perpendicularity restrictions between pipes, and $ParP \in P \times P$ is the set of parallelism restrictions between pipes.

$CPCD$, $CPOS$, $CCCD$ and $CCOS$ are the cost functions associated with the connections between pipes and piping system components, which we describe in Section 5.4. $CollP$, $PerP$ and $ParP$ are the cost functions associated with restrictions between pipes, which we describe in Section 5.5. As the Ceres library does not provide restrictions, the orientation vectors and the rotation quaternions need to be normalized inside all cost functions, but this was omitted from our notation for readability.

5.4 Connection Cost Functions

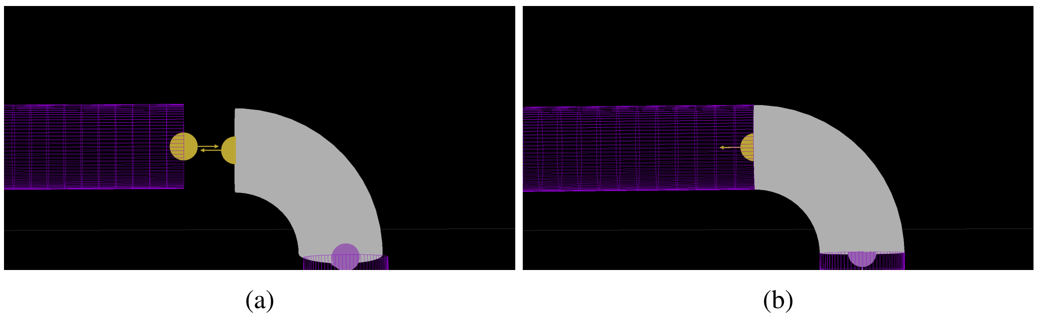
To ensure that the piping system elements are correctly aligned, we have implemented cost functions that calculate the distance between the ports of two elements and the sum of their orientations. Figure 5.5 shows an elbow disconnected from a pipe, and the expected result after executing the optimization. As the position and orientation of a pipe port are calculated differently from how they are calculated for a component port, we also needed to implement different cost functions for the connection between pipe and component, and the connection between component and component. However, as they are very similar, we will only present the cost functions associated with the connection of a pipe and a component. They can be represented as:

$$CPCD(cp, cr, pp, pa, pl) = (cp + cr * CPP) - (pp + \frac{PPD * pl * pa}{2}), \tag{5.3}$$

$$CPOS(cr, pa) = (cr * CPO) + (PPD * pa), \tag{5.4}$$

where $CPCD$ stands for Component Pipe Connection Distance, $CPOS$ stands for Component Pipe Orientation Sum, CPP stands for Component Port Position, CPO stands for Component Port Orientation, and PPD stands for Pipe Port Direction. The PPD of a pipe port is either 1 or -1, and it is already known for each port. The CPP and CPO are the canonical position and orientation of the port, and were obtained as described in Chapter 4.

Figure 5.5 – A scene containing an elbow and a pipe (a), and the result after optimization (b). Notice that, for the components to be correctly aligned, the distance between their ports must be zero, and the length of the sum of the orientations of their ports must also be zero.



Source: The Authors

In Equation 5.3, the position of the component's port is calculated by applying the rotation of the component to the port's canonical position and then adding the component's position. The position of the pipe's port is calculated by adding the axis of the pipe multiplied by half of its length to the position of the pipe.

Similarly, in Equation 5.4, the orientation of the component's port is calculated by applying the rotation of the component to the canonical orientation of the port. The orientation of the pipe's port is calculated by multiplying its axis by PPD.

5.5 User Interaction

After executing our optimization method, the result might still not be perfect, so we added some functionalities to help improve it. The user can manually add and remove connections between pipes and piping system components, change the weight that multiplies each type of cost function, and set the parameters of a pipe or a piping system component constant.

Besides, the user can add restrictions between pipes to force parallelism, perpendicularity, or collinearity, which are also implemented as Ceres cost functions. The cost functions for perpendicular, parallel, and collinear pipe restrictions are, respectively:

$$\begin{aligned} PerP(pa0, pa1) &= pa0 \cdot pa1, && (5.5) \\ ParP(pa0, pa1) &= 1 - (pa0 \cdot pa1)^2, && (5.6) \\ CollP(pp0, pa0, pp1, pa1) &= 1 - [(pp1 - pp0).normalized() \cdot pa0]^2 \\ &\quad + 1 - [(pp1 - pp0).normalized() \cdot pa1]^2. && (5.7) \end{aligned}$$

One interpretation for the perpendicular pipe restriction function is that two pipes are perpendicular if the dot product of their axes is zero. If the dot product between the orientation of two pipes is either 1 or -1, they are parallel. In the case of collinear pipes, both axes must be parallel to the vector that starts at the center of one pipe and ends at the center of the other pipe.

These cost functions were tested by applying them to pairs of cylinders with random orientations and positions. In our tests, the perpendicular and parallel cost functions worked consistently. However, when applying the collinearity restriction, our method sometimes converged to a local minimum, where the cylinders were not collinear.

5.6 Summary

This Chapter discussed the algorithms we proposed to improve the cylinder detection and how we used optimization to fix misalignments among piping system components. We also described the actions one can perform to improve the result of the optimization, which include determining that a pair of cylinders must be perpendicular, parallel, or collinear.

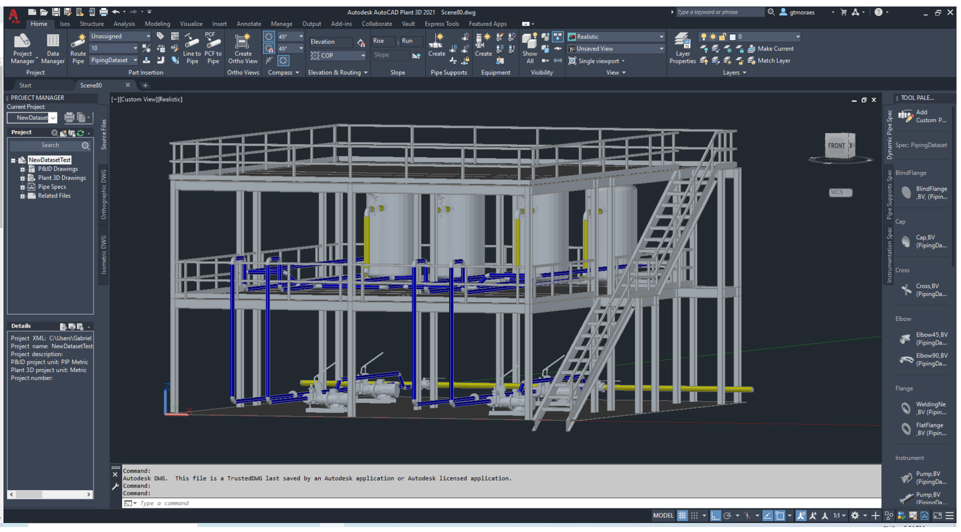
6 INTEGRATION WITH CAD SOFTWARE

One essential aspect of our system is the integration with a CAD software. The user must be able to modify the reconstructed model as needed, and instead of implementing all functionalities provided by a CAD software, we allow the user to export the reconstructed CAD model to AutoCAD Plant 3D, a widely used software for this purpose. This Chapter describes a plug-in we developed for AutoCAD Plant 3D and the necessary steps to use our dataset inside this CAD software.

6.1 AutoCAD Plant 3D Catalog

In order to create scenes using our dataset described in Chapter 3, we first needed to create a Plant 3D catalog of piping components that represents our dataset. To do that, we imported all models from our dataset into Plant 3D using a third-party plug-in (SYSTEMS, 2021), and used the information about the ports of the components, as also described in Chapter 3, to add ports to the models and convert them to piping components. After that, we created a catalog and piping spec from these piping components using AutoCAD Plant 3D Spec Editor, a software from Autodesk that comes with AutoCAD Plant 3D. Figure 6.1 shows AutoCAD Plant 3D's interface, with a scene that uses models from our catalog.

Figure 6.1 – The AutoCAD Plant 3D interface, with which the user can instantiate the models of our dataset, displaying one of the scenes modelled by us.



Details

Project XML: C:\Users\Gabriel
Project name: NewDatasetTest
Project description:
P&ID project unit: PIP Metric
Plant 3D project unit: Metric
Project number:

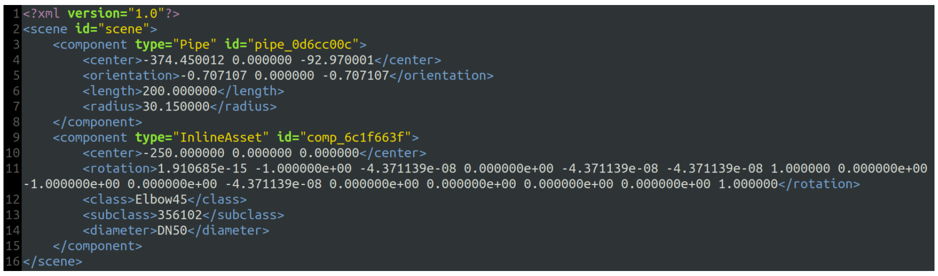
```
Command:
Autodesk DWG.  This file is a TrustedDWG last saved by an Autodesk application or Autodesk licensed application.
Command:
Command:
```

Source: AutoCAD Plant 3D

6.2 The Reverse Engineering Plugin

To integrate our results with the AutoCAD Plant 3D, we developed a C# plug-in using the ObjectARX SDK (AUTODESK, 2021b). It provides two functionalities: import an XML file that contains a description of a scene; and export a scene into an XML file. An example of such a file can be seen in Figure 6.2, which can also be imported into or exported from the Point Cloud Editor.

Figure 6.2 – Example of an XML file that represents the output of the reconstruction of a scene using the Point Cloud Editor.



```xml
<?xml version="1.0"?>
<scene id="scene">
    <component type="Pipe" id="pipe_0d6cc00c">
        <center>-374.450012 0.000000 -92.970001</center>
        <orientation>-0.707107 0.000000 -0.707107</orientation>
        <length>200.000000</length>
        <radius>30.150000</radius>
    </component>
    <component type="InlineAsset" id="comp_6c1f663f">
        <center>-250.000000 0.000000 0.000000</center>
        <rotation>1.910685e-15 -1.000000e+00 -4.371139e-08 0.000000e+00 -4.371139e-08 -4.371139e-08 1.000000 0.000000e+00
-1.000000e+00 0.000000e+00 -4.371139e-08 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.000000</rotation>
        <class>Elbow45</class>
        <subclass>356102</subclass>
        <diameter>DN50</diameter>
    </component>
</scene>
```

Source: The Authors

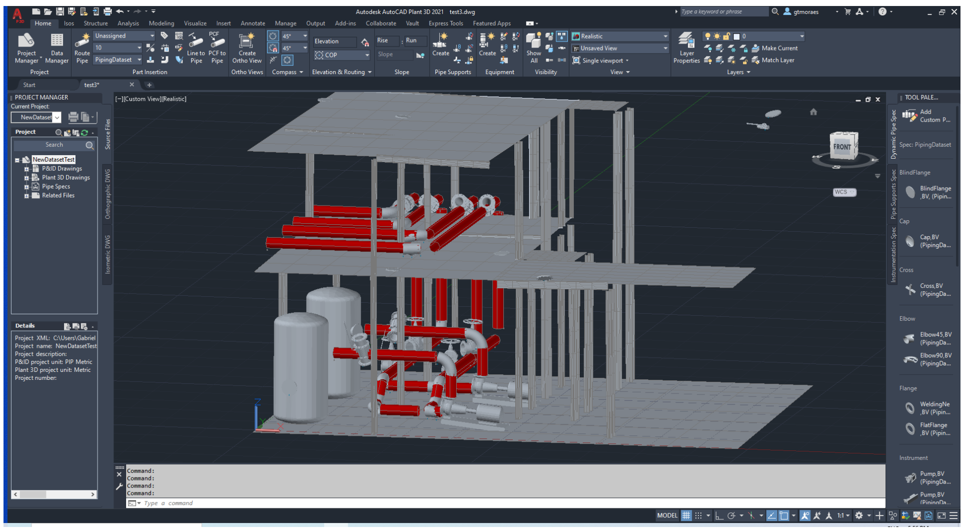
The pipes and piping system components are represented using the catalog we created. Although there are plates, which could represent planes, and structural members in Plant 3D, it is not possible to create them using the provided SDK. To represent planes inside AutoCAD Plant 3D, we create a lofted surface between two nonconsecutive edges of the plane. Structural elements (i.e., beams) are represented as three planes. Figure 6.3 shows a scene imported into Plant 3D using our plugin.

Another detail worth mentioning about this plug-in is that AutoCAD Plant 3D uses a different world coordinate system's convention than the Point Cloud Editor. As the XML file that represents a scene adopts the Point Cloud Editor's world coordinate system, when importing the scene into Plant 3D, the plug-in must also convert everything into its world coordinate system.

6.3 Summary

This Chapter discussed the integration of our reverse engineering system with AutoCAD Plant 3D, allowing us to obtain CAD representations for scanned piping system installations.

Figure 6.3 – An industrial site reconstructed using our system, using a preliminary result of the pose estimation, before applying the pose optimization.



```
Command:
Command:
Command:
Command:
```

Source: AutoCAD Plant 3D

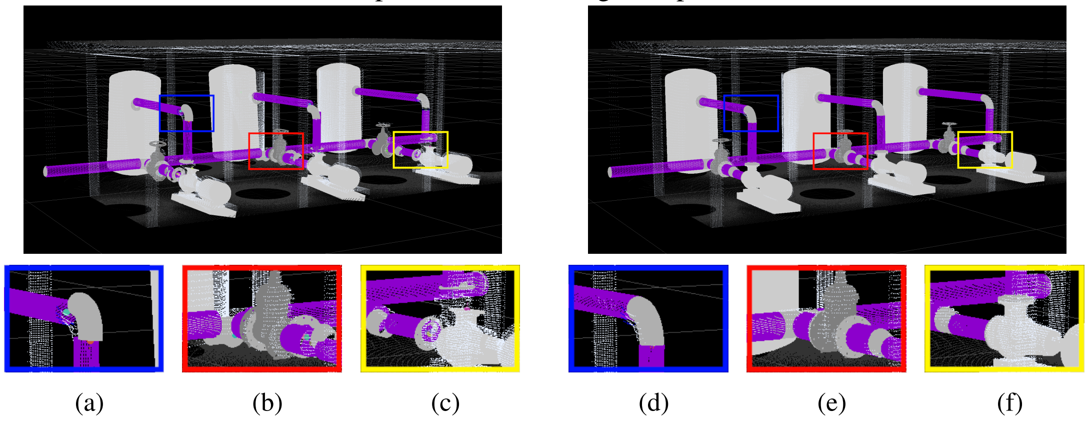
7 RESULTS

In this Chapter, we show the results of our proposed method for the alignment of piping system components. The tests presented here were executed in a computer with a *Intel Core i5-9400F* processor and 16GB of RAM.

Our tests were made using the scenes we modeled with Plant 3D and scanned with our LiDAR simulator. The pipes were semi-automatically detected using our system. To simulate the error in the pose estimation of piping system components, except for pipes, we added Gaussian noise to their original positions and rotations. The standard deviation of the noise added to the components' positions was 50mm, and the standard deviation of the noise added to the components' rotations was 5°. Figures 7.1 and 7.2 show the result after executing the optimization in two of our scenes, and Table 7.1 presents the average execution time and average number of iterations of the optimization procedure after executing our algorithm 10 times for each scene.

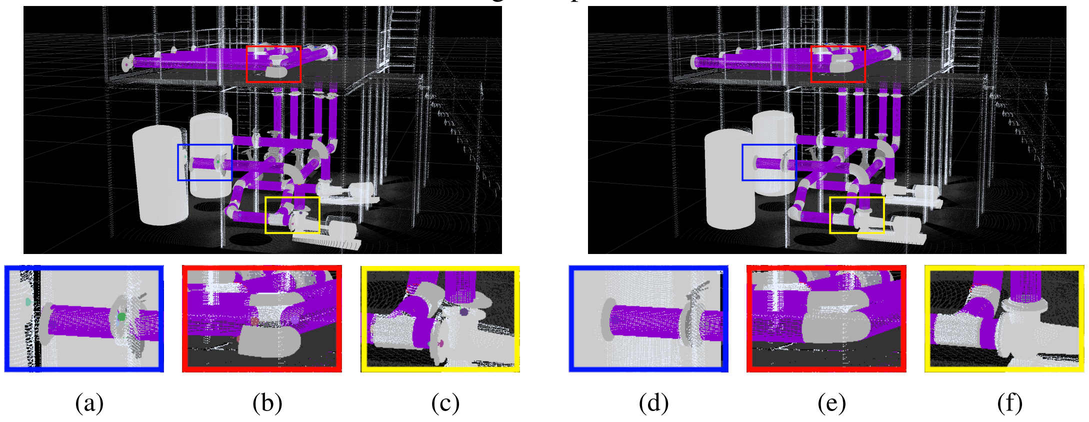
During our tests, we noticed that the algorithm for detecting connections would frequently fail. Figure 7.3 shows an example where the detected connections are wrong. Because of that, in the results shown here, we executed the optimization after manually fixing the connections between the ports.

Figure 7.1 – The piping system of one of our scenes after adding noise to the pose of piping components (left), and after executing optimization (right). In (a), (b) and (c) we highlight some examples of misaligned components, and in (d), (e) and (f) we highlight their correctly aligned counterparts after executing the optimization.



Source: The Authors

Figure 7.2 – Another scene after adding noise to the pose of piping components (left), and after executing optimization (right). In (a), (b) and (c) we highlight some examples of misaligned components, and in (d), (e) and (f) we highlight their correctly aligned counterparts after executing the optimization.



Source: The Authors

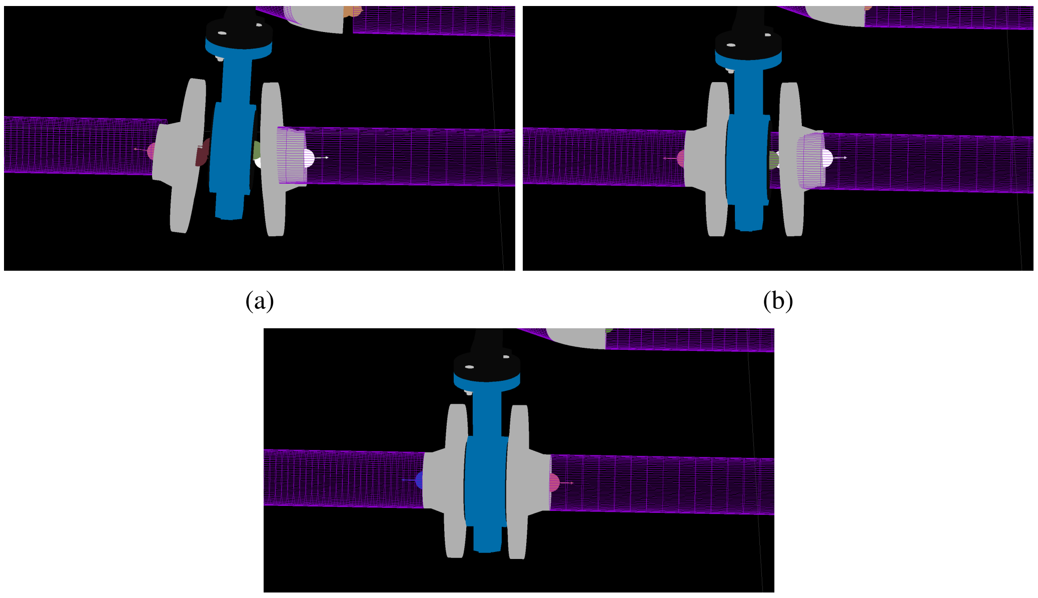
Another issue we noticed is that sometimes the orientation of a valve after the optimization differs from the original orientation, even though the ports of the components and pipes are properly aligned. Figure 7.4 shows an example of this issue. This happens because if we rotate a valve around the vector that represents its flow, the position and orientation of its ports do not change, so it does not affect the result of our optimization problem.

In the tests previously described, the optimization step always produced results similar to the original scenes. We also performed another test, in which we only added a random value in the range [-45°,45°] to the Euler angles that represent the rotation of the components. Similarly to the previous tests, the connections between the ports were fixed manually. This test could be interpreted as the cylinder detection working perfectly, but the pose estimation of components is almost random. In this case, even though the alignment of the ports is corrected, the resulting scene differs from the original one, as shown in Figure 7.5.

Table 7.1 – Average execution time and number of iterations until convergence for each scene, as well as the number of pipes, components and connections.

	Pipes	*Components*	*Connections*	*Iterations*	*Execution Time (s)*
Scene 1	19	35	53	62.0	3.75e-02
Scene 2	42	78	118	56.2	1.52e-01
Scene 3	80	116	195	46.6	2.19e-01
Scene 4	51	76	126	65.4	1.95e-01
Scene 5	44	63	107	76.8	2.12e-01

Figure 7.3 – In the following images, ports with the same color are connected to each other, and ports with the white color are not connected to any other port. Depending on how large is the error of the pose estimation of pipes and components, the process of automatic detection of connections between components might fail. Image (b) shows the result after the optimization using the automatically detected connections, where the right flange is left disconnected from the scene. Image (c) shows the result after manually fixing the connections between pipes and components.



(a) (b)

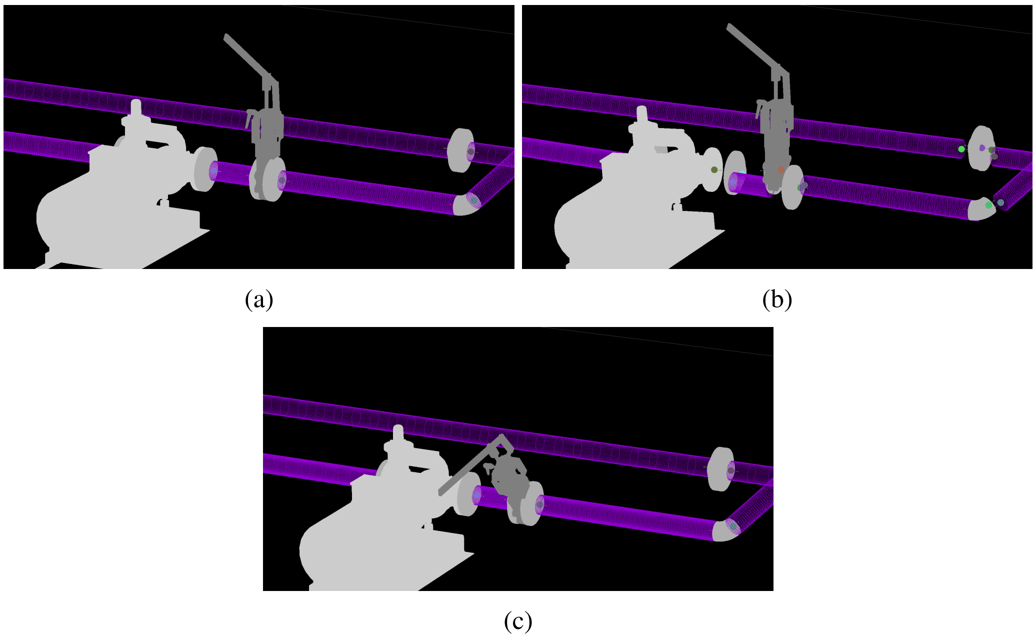
(c)

Source: The Authors

7.1 Summary

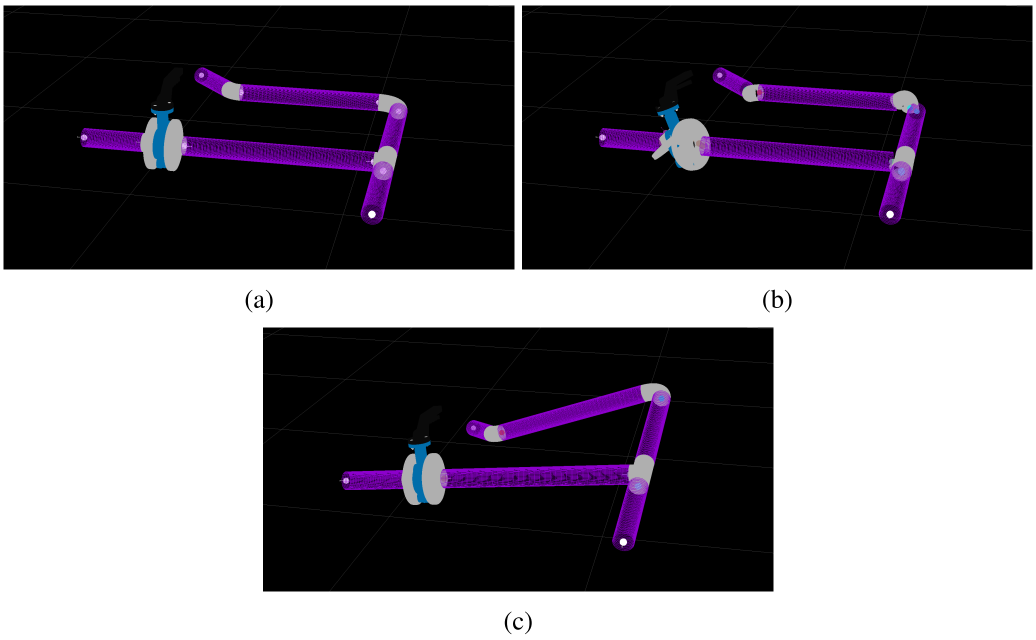
This Chapter presented the results of the pose optimization step of our system. When the connections between components were correctly established, the optimization always resulted in correctly aligned components. However, depending on how large is the error of the pose optimization, the connection detection algorithm might fail, and the result of the optimization might be considerably different than expected.

Figure 7.4 – The orientation of valves sometimes is different from the orientation before adding noise to its pose. In (a), we have the original scene, in (b) the scene after adding noise to the pose of the pipes and piping components, and in (c) the result after executing the optimization. Although the orientation of the valve was not preserved, the result is still plausible.



Source: The Authors

Figure 7.5 – Another example where the resulting scene after optimization differs from the original scene. In (a) we have the original scene, in (b) the scene after adding noise to the rotation of the piping components, and in (c) the result after executing optimization. Just like in the previous example, the result is also plausible.



Source: The Authors

8 CONCLUSION

This work presented several contributions to the reverse engineering of CAD models of industrial plant sites from point clouds. One of our main contribution is an optimization-based procedure to fix the misalignment of the detected piping system components. Another contribution of this work is the integration of our results with a CAD software. Even though the resulting model is not a perfect reconstruction of the underlying industrial plant site, the user can still manually improve this result using either our system or AutoCAD Plant 3D. These solutions contribute towards the goal of obtaining an automatic reverse engineering system.

8.1 Future Work

Our system still has several limitations. First, we can only reconstruct industrial sites that contain models in our dataset. In order to work with other datasets, one would need to detect the position and orientation of the ports of the models in the dataset, which currently is not entirely automatic. It would be desirable to improve the automatic port detection, to minimize the user interaction needed in this step.

As we have seen in Chapter 7, the detection of connections between ports can also fail if the error in the pose estimation or in the pipe detection is too large. The algorithm should also avoid connecting ports of components with different nominal diameters.

Depending on the error in the pose estimation of components, the optimization might still be able to align the components, but potentially modifying the poses of components that were already well aligned. To determine whether this would be a problem in practice, we need to test our solution on real data and use the result of the pose estimation. Alternatively, it could be interesting to test the robustness of our method as the error in the pose estimation is increased, which can be simulated as described in Chapter 7. In this case, it would be necessary to measure how different the result after optimization is from the original scene.

It is also necessary to test the scalability of our proposed method for the alignment of piping system components using scenes with a larger number of connections. This is particularly important as the pose optimization step is executed whenever the user of our system performs any action described in Section 5.5.

REFERENCES

3DSYSTEMS. *Geomagic Design X 3D Reverse Engineering Software*. 2021. <https://www.3dsystems.com/software/geomagic-design-x>. Last accessed: 11/04/2021.

AGARWAL, S.; MIERLE, K.; OTHERS. **Ceres Solver**. 2010. <http://ceres-solver.org>. Last accessed: 11/04/2021.

ARAUJO, A. M. C.; OLIVEIRA, M. M. Connectivity-based cylinder detection in unorganized point clouds. **Pattern Recognition**, v. 100, p. 1–12, 2020. ISSN 0031-3203. Article 107161.

ARAUJO, A. M. C.; OLIVEIRA, M. M. A robust statistics approach for plane detection in unorganized point clouds. **Pattern Recognition**, v. 100, p. 1–12, 2020. ISSN 0031-3203. Article 107115.

ARAUJO, A. M. C.; OLIVEIRA, M. M. Towards reverse engineering of industrial site plants. In: **Anais do XXXIII Concurso de Teses e Dissertações**. Porto Alegre, RS, Brasil: SBC, 2020. p. 37–42. ISSN 2763-8820. Available from Internet: <https://sol.sbc.org.br/index.php/ctd/article/view/11366>.

AUTODESK. **AutoCAD**. 2021. <https://www.autodesk.com/products/autocad/overview>. Last accessed: 11/04/2021.

AUTODESK. **AutoCAD ObjectARX SDK**. 2021. <https://www.autodesk.com/developer-network/platform-technologies/autocad/objectarx>. Last accessed: 11/04/2021.

AUTODESK. **AutoCAD Plant 3D**. 2021. <https://www.autodesk.com/products/autocad/included-toolsets/autocad-plant-3d>. Last accessed: 11/04/2021.

AUTODESK. **Autodesk ReCap**. 2021. <https://www.autodesk.com/products/recap/overview>. Last accessed: 11/04/2021.

AVEVA. **AVEVA E3D Design**. 2021. <https://www.aveva.com/en/products/e3d-design/>. Last accessed: 11/04/2021.

BOTSCH, M. et al. Openmesh: A generic and efficient polygon mesh data structure. In: . [S.l.: s.n.], 2002.

CLEAREDGE3D. **EdgeWise - ClearEdge3D**. 2021. <https://www.clearedge3d.com/edgewise/>. Last accessed: 11/04/2021.

COMMUNITY, B. O. **Blender - a 3D modelling and rendering package**. 2018. <http://www.blender.org>. Last accessed: 11/04/2021.

DEVITO, Z. et al. Opt: A domain specific language for non-linear least squares optimization in graphics and imaging. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 36, n. 5, oct. 2017. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/3132188>.

EBERLY, D. **Geometric Tools Engine, Version 5**. 2020. <https://github.com/davideberly/GeometricTools>. Last accessed: 11/04/2021.

FARO. **FARO As-Built for AutoCAD Software**. 2021. <https://knowledge.faro.com/Software/As-Built/As-Built_for_AutoCAD_Software>. Last accessed: 11/04/2021.

FERGUS, R. et al. Removing camera shake from a single photograph. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 25, n. 3, p. 787–794, jul. 2006. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/1141911.1141956>.

FIGUEIREDO, L.; IVSON, P.; CELES, W. Deep learning-based framework for shape instance registration on 3d cad models. **Computers & Graphics**, 2021. ISSN 0097-8493. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S009784932100176X>.

GOMEASURE3D. **QUICKSURFACE: Scan to CAD Software**. 2021. <https://gomeasure3d.com/quicksurface/>. Last accessed: 11/04/2021.

HEIDE, F. et al. Proximal: Efficient image optimization using proximal algorithms. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 35, n. 4, jul. 2016. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/2897824.2925875>.

HEIDE, F. et al. Low-budget transient imaging using photonic mixer devices. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 32, n. 4, jul. 2013. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/2461912.2461945>.

HEIDE, F. et al. High-quality computational imaging through simple lenses. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 32, n. 5, oct. 2013. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/2516971.2516974>.

HEXAGON. **Intergraph Smart 3D**. 2021. <https://hexagonppm.com/offerings/products/intergraph-smart-3d>. Last accessed: 11/04/2021.

IVSON, P.; CELES, W. Instanced rendering of massive cad models using shape matching. In: **2014 27th SIBGRAPI Conference on Graphics, Patterns and Images**. [S.l.: s.n.], 2014. p. 335–342.

JIN, Y.-H.; LEE, W.-H. Fast cylinder shape matching using random sample consensus in large scale point cloud. **Applied Sciences**, v. 9, p. 974, 03 2019.

KOCH, S. et al. Abc: A big cad model dataset for geometric deep learning. In: **The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2019.

LI, L. et al. Supervised fitting of geometric primitives to 3d point clouds. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2019.

LIMBERGER, F. A.; OLIVEIRA, M. M. Real-time detection of planar regions in unorganized point clouds. **Pattern Recognition**, v. 48, n. 6, p. 2043–2053, 2015. ISSN 0031-3203.

PEREZ, P.; GANGNET, M.; BLAKE, A. Poisson image editing. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 22, n. 3, p. 313–318, jul. 2003. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/882262.882269>.

POLYGA. **XTract3D Add-in for SOLIDWORKS**. 2021. <https://www.polyga.com/xtract3d/>. Last accessed: 11/04/2021.

RABBANI, T.; HEUVEL, F. Efficient hough transform for automatic detection of cylinders in point clouds. **Proc ISPRS Workshop Laser Scan 2005, ISPRS Arch**, v. 36, 01 2005.

RAJA, V.; FERNANDES, K. J. **Reverse engineering: an industrial perspective**. [S.l.]: Springer Science & Business Media, 2007.

SCHNABEL, R.; WAHL, R.; KLEIN, R. Efficient ransac for point-cloud shape detection. **Computer Graphics Forum**, Blackwell Publishing, v. 26, n. 2, p. 214–226, jun. 2007.

SORKINE, O.; ALEXA, M. As-rigid-as-possible surface modeling. In: **Proceedings of the Fifth Eurographics Symposium on Geometry Processing**. Goslar, DEU: Eurographics Association, 2007. (SGP '07), p. 109–116. ISBN 9783905673463.

SYSTEMS, A. **OBJ2DWG**. 2021. <https://apps.autodesk.com/ACD/de/Detail/Index?id=3563402343067399513&appLang=en&os=Win32_64>. Last accessed: 11/04/2021.

TECHNODIGIT. **3D Reshaper: The 3D Scanner Software**. 2020. <https://www.3dreshaper.com/en/software-en/>. Last accessed: 11/04/2021.

TRACEPARTS. **TraceParts - Product Content Everywhere**. 2021. <https://traceparts.com/>. Last accessed: 11/04/2021.

TRAN, T.-T.; CAO, V.-T.; LAURENDEAU, D. Extraction of cylinders and estimation of their parameters from point clouds. **Computers & Graphics**, v. 46, p. 345–357, 02 2015.

VISIONWORKPLACE. **OBJ Converter for Autodesk AutoCAD**. 2021. <https://visionworkplace.com/products/obj-converter-for-autodesk-autocad>. Last accessed: 11/04/2021.

WU, C. et al. Real-time shading-based refinement for consumer depth cameras. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 6, nov. 2014. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/2661229.2661232>.

ZORAN, D.; WEISS, Y. From learning models of natural image patches to whole image restoration. In: **2011 International Conference on Computer Vision**. [S.l.: s.n.], 2011. p. 479–486.