Master Project

Object-in-Hand Scanning

Shreya Deshmukh

University Examiner: Prof. Thomas Brox
University Adviser: Max Argus

Company Name: Optonic Gmbh
Company Adviser: Lars Schiller

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
November 17th, 2025

Abstract

Accurate 3D models play an important part in Robotic applications. It is a crucial requirement for achieving precise object manipulation and robust perception tasks. This report details core methodologies necessary for Surface reconstruction. The core concept of this project is the process that is required to convert raw sensor data into a coherent 3D mesh model. The goal is to achieve an accurate 3D model of an object placed in a Robot gripper. This process starts with data acquisition through a stereo camera integrated with ROS framework. These stereo images are then used to generate dense Point Clouds. This conversion is essential for the subsequent pre-processing steps and providing necessary input for surface reconstruction. The processed point clouds are then used to generate the 3D mesh. The resultant 3D mesh can be used for other robotic applications that require accurate spatial information thereby promoting precise gripping and improved decision making capabilities.

Contents

1 Introduction

1.1 Project Objective

In the fast growing field of Robotics, perception and interaction with the environment plays a pivotal role. Perception is a means for the robot to understand the environment. Therefore, in order to have a better understanding of the environment, the robot requires an accurate knowledge about the position and orientation of objects. Perception includes tasks such as object detection, 3D object modeling etc. An accurate 3D model helps to get accurate geometric and semantic information about any given objects. This information is necessary for other crucial tasks in robotics like Manipulation and Control. To get an accurate object model one of the crucial information to have is depth information. Depth perception is necessary while dealing with distances of objects from robots. This includes tasks like grasping, obstacle avoidance etc. This report details a comprehensive methodology to generate accurate 3D model by using the depth perception of stereo camera. This includes utilizing Robot Operating System (ROS) as discussed in Section 3.3, for robust communication and other tools for data handling and processing. Object scanning and modeling pipeline begins with acquiring raw image data from a stereo camera. These images are received through ROS and then transformed into 3D point clouds (discussed in Section 3.4). Pre-processing of these point clouds is essential to reduce noise and remove outliers to improve the data quality. Once done, it is easier to segment the required object model. Finally, these processed object point clouds are converted into 3D meshes for surface reconstruction.

1.2 Camera and Robot setup

The Ensenso camera is a stereo camera set up on Mikado robot[2]. Mikado is a bin picking robot and a part of Robotics team at Optonic GmbH. The project setup consisted of a stationary stereo camera and a moving robot gripper, as shown in fig [1]. The process to get a 3D object model starts with checking the internal and external

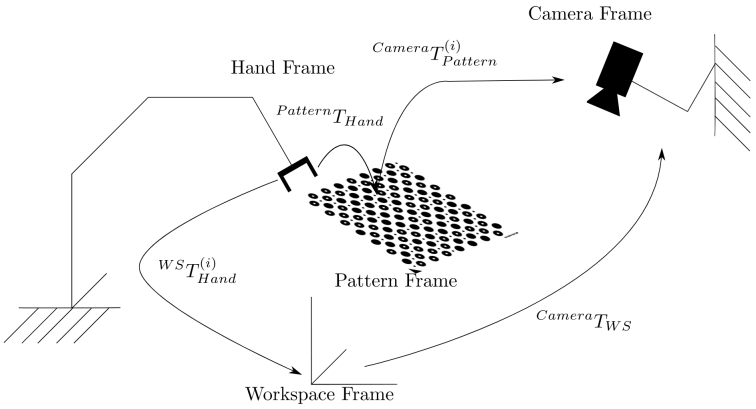


Figure 1: Ensenso Camera calibration
[1]

camera calibration. Camera calibration is discussed in detail in Section 3.2. The ensenso cameras are usually pre-calibrated at the factory. This internal calibration discussed in Section 3.2.1, is done to determine the initial parameters and stereo baselines. This simplifies the initial setup. In case of any changes, the Ensenso SDK has calibration wizard to make any changes in the calibration parameters. For extrinsic calibration, the process is called Hand-Eye calibration. The rigid body transformation is given by $\mathbf{R}_{cam\_to\_robot}$ and $\mathbf{T}_{cam\_to\_robot}$ between camera coordinate system and robot base coordinate system. This process involves a moving robot-mounted camera with a fixed calibration target. Using the correlation between camera's calibrated pose relative to target with robot's known pose for each position, the Hand-Eye transformation can be solved.

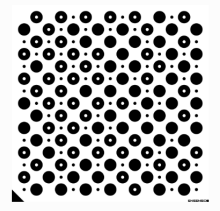


Figure 2: Checker Board Pattern
[1]

Hand-Eye calibration determines the position of camera relative to robot. The calibration process involves collecting multiple observations of calibration pattern with different robot poses. In this setup, the camera is fixed relative to the robot's workspace and the robot hand moves the calibration pattern. This is shown in the figure [1] where $^{\mathrm{WS}}\mathbf{T}_{\mathrm{Camera}}$ is the transformation from workspace to camera. $^{\mathrm{WS}}\mathbf{T}^{(i)}_{\mathrm{Hand}}$

represents the robot's end-effector pose relative to World Frame. ${}^{\text{Hand}}\mathbf{T}_{\text{Pattern}}$ defines the pose of calibration pattern relative to robot's hand frame and ${}^{\text{Camera}}\mathbf{T}^{(i)}_{\text{Pattern}}$ represents the pose of the Pattern Frame relative to Camera Frame at pose i.

The calibration patterns (fig. [2]) have the ability to encode information(like grid spacing). This pattern is a custom one used in the in-house library called NxLib[1]. The grid spacing is to be specified manually before using the pattern[1].

2 Related Work

Object in hand scanning for robotics is an important area of research. Applications that demand precise manipulation and interaction with objects make use of these approaches. Traditional approaches that do not employ neural networks have significantly contributed to the field. These methods have a lot of techniques, each with its unique advantages and limitations.

Geometric methods such as ICP algorithm discussed in Section 3.5, introduced by Besl and McKay in 1992[3] and its variants have become a foundational method for aligning 3D point clouds. This method is robust and accurate in aligning and merging point clouds without the neural networks. Another integral part of the object in-hand scanning is the use of stereo vision. This brings the requirement for accurate camera calibration[4]. This paper discusses the camera calibration in detail to achieve precise depth perception. Another contribution in this domain is [5] which discusses the robustness of structured light methods to produce 3D scans. Camera calibration is discussed in more detail in Section 3.2.

Many 3D object reconstruction methods assume that the objects have distinctive geometric features or textures. This helps in feature matching for multiple views. But many objects might have coherent textures or simple geometry. This creates problem for object scanning and reconstruction. When a hand manipulates an object, additional cues are added. [6] takes leverage of this property and creates watertight 3D mesh using RGB-D video sequence as input. Figure 3 shows the hand tracker used in the in-hand scanning pipeline. The left image shows the raw depth input map, the middle image shows the hand pose overlaid on top of the RGB-D data, while the right image shows just the hand pose.

Object in-hand scanning usually has small registration errors when revisiting previously seen parts of an object. This leads to loop closure problems. [7] addresses this issue. It employs this loop closure online. They employ "what you see is what you get" (WYSIWYG) model. The final 3D model is high quality and does not need any post processing. The drawback however with this is that it needs rigid objects that are manipulated by hand.

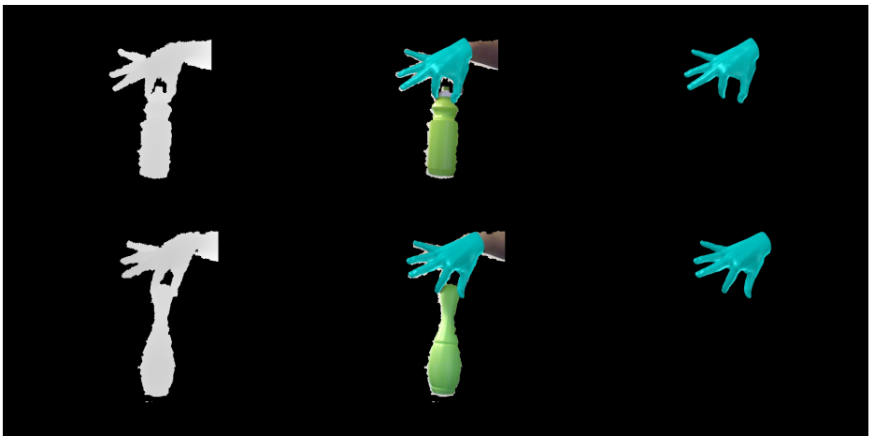


Figure 3: 3D Object Reconstruction from Hand-Object Interactions [6]

Figure 4: In-hand Scanning with Online Loop Closure[7]

Apart from the traditional methods discussed above, there are also techniques that use neural networks for Object reconstruction from in-hand scanning. Although, using neural networks was not an objective for this project, they are nonetheless helpful for any future work. [8] uses MLP that learns the 3D geometry and color of the object directly from the RGB video. There are also other works that use neural networks for similar topics[9][10].

3 Background

3.1 Stereo Vision

Robots are systems that are designed to accomplish a certain task with little to no human interaction. In order to do that, it must perceive the world around which involves identifying the shape, size, distance between the robot and the object and other physical characteristics. To achieve this, the robot should be able to estimate the objects in 3D. Stereo vision enables the robot to achieve this. It has played an important role in the development of computer vision and 3D reconstruction. Stereo vision works similar to how humans perceive the environment. It uses two cameras placed side-by-side. Both cameras capture the scene at a given instance. Finding matches in those two images should follow the rules of epipolar geometry. Each pixel's match in another image can be found on the line called epipolar line. If two images are coplanar, the right camera is only shifted horizontally in comparison to the left camera. In these situations, Image rectification is used. It warps both images such that they appear as if they were taken with horizontal displacement. Therefore, all the epipolar lines are horizontal, thus simplifying the stereo matching process.

The main objective of stereo vision is depth estimation. This can be achieved by a process called Triangulation. In this setup, each camera captures an image from different viewpoints. Therefore, a point lying on a 3D plane is shifted in the two images. This shift is called disparity. Objects that are closer to the camera have higher disparity whereas the objects that are farther away have lower disparity. Considering the vertical co-ordinate y as same, the depth then can be calculated using equation1.

$$Z = \frac{f \cdot B}{d} \tag{1}$$

The raw images from the Stereo camera have two different viewpoints. For each pixel in these images, the corresponding scene point's depth is determined. This is done by finding matching pixels in the two images and then applying triangulation to these matches to determine the depth. In computer vision, triangulation is a process

to determine a point in 3D given its projections onto two or more images. In order to solve this problem, it is necessary to know the parameters of the camera projection function from 3D to 2D for the cameras involved. Even after going through a process like image rectification(which aligns the images) and utilizing the epipolar constraint, there might still be a need to accurately find the correct matching pixels for every point in the reference image. Issues like occlusions and repetitive patterns might make this difficult. To tackle this, there are various stereo matching algorithms which implement different methods to calculate pixel similarity.

3.2 Camera Calibration

Camera calibration is one of the essential steps in 3D reconstruction, robotic applications and so on, that requires accurate, quantitative information from an image. The reasons for inaccurate measurements can be because of distortion in lens, sensor issues and camera positions. The camera calibration aims to determine the geometric parameters of the image formation process. This is a crucial step in many computer vision applications especially when metric information about the scene is required. In these applications, the camera is generally modeled with a set of intrinsic parameters (focal length, principal point, skew of axis) and its orientation is expressed by extrinsic parameters (rotation and translation)[11]. It is basically to form a camera model to have a bridge between 3D points in the real world co-ordinates and their corresponding 2D projection. The most common method uses calibration pattern like checkerboard with known dimensions. Multiple images of the pattern are taken from different angles and distances.

Firstly, its important to understand the 3D to 2D projection. Let there be a point P(X,Y,Z) in world coordinate system. We want to convert it to a 2D image point p=(x,y). Here a pinhole camera is considered.

$$s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{A}[\mathbf{R} \mid \mathbf{t}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

where s is an arbitrary scale factor, (R,t) called the extrinsic parameters is the rotation and translation which relates the world coordinate system to the camera coordinate system, and A is called the camera intrinsic matrix.

3.2.1 Intrinsic Camera Parameters

Intrinsic parameters define the internal geometry and optics. The intrinsics are given by eq.3 where u_0 and v_0 are the coordinates of the principal point, α and β are the scale factor and the parameter describing the skew of the two image axes respectively[4]. This matrix is same for a given camera and lens, regardless of the environment. Accurately estimating this matrix is necessary for 3D reconstruction.

$$\mathbf{A} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

3.2.2 Extrinsic Camera Parameters

The extrinsic parameters is the matrix that controls the pose(rotation and translation) of the camera in world co-ordinate system. As shown in the matrix eq.4,R is a 3x3 rotation matrix and t is the 3x1 translation matrix. For every image, this matrix is calculated.

$$[\mathbf{R} \mid \mathbf{t}] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{4}$$

3.2.3 Lens Distortion

Lenses often have non-linear distortions. This can give rise to issues wherein the straight lines can appear curved and displace image points. There are majorly two types of distortions.

Radial Distortion

In Radial distortion, image points are either towards or away from the image center. Here $r^2 = x^2 + y^2$

$$\begin{aligned} x_d &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_d &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{5}$$

Tangential Distortion

Tangential distortion arises when the lens and image sensor are not perfectly aligned.

$$x_d = x + [2p_1xy + p_2(r^2 + 2x^2)]$$
$$y_d = y + [p_1(r^2 + 2y^2) + 2p_2xy] \tag{6}$$

3.3 ROS

The Robot Operating System (ROS or ros) is a set of software libraries and tools that help build robot applications[12]. ROS is an open-source robotics middleware. ROS is not an operating system in itself but its a set of software frameworks for robotics applications. It follows graph architecture wherein the processing takes place in a subsystem called nodes. ROS is open source and therefore gives users the independence to choose the configurations of tools and libraries.

Nodes

A node is an executable program that performs a given task in ROS. Every node has a unique name, which it registers with the ROS master. Multiple nodes with different names can exist under different namespaces, or a node can be defined as anonymous, in which case it will randomly generate an additional identifier to add to its given name. Nodes are at the center of ROS programming. Most of the client coding consists of nodes.

Topics

Topics are a type of buses over which nodes can send and receive messages. The names of these buses need to be unique. In order to send messages to a topic, a node must publish and to receive it should subscribe. This publish and subscribe model is hidden from the other nodes. That means the information transferred between the corresponding nodes are accessible only to them.

Services

A node can also advertise services. A service represents an action that a node can take. These services have a defined start and end, such as capturing a one-frame image. Nodes advertise services and call services from one another.

ROS Master

A ROS Master is a central system that is responsible for coordinating communication between the nodes. It has other tasks as well, such as maintaining registry of active nodes, managing topic connections and facilitating the discovery and commu-

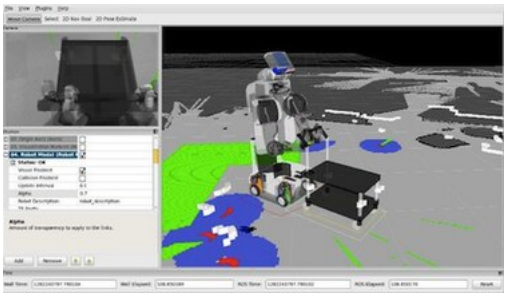


Figure 5: ROS[12]

nication of nodes within the ROS network.

Tools

ROS has a variety of tools that allows visualization and recording of data. These tools help navigate the packages and create scripts to automate complex configuration and setup processes. These tools are provided in packages that provide task and robot agnostic tools. Some of these tools are mentioned below.

1. **rviz**: rviz(Robot Visualization tool) is a 3D visualizer. It is used to visualize robots actions, the environment they work in and sensor data. It is configurable and can adjust to a users needs with many different visualizations and plugins.

2. **rosbag**: rosbag is a command line tool used to record and playback ROS messages. rosbag uses file format called bags. These bags log messages by listening to topics and recording messages. Playing messages back from a bag is largely the same as having the original nodes that produced the data in the ROS computation graph, making bags a useful tool for recording data to be used in later development[12].

3. **catkin**: catkin is a build system based on CMake. Similar to CMake, it is cross-platform, opensource and language independent. Catkin simplifies the process of building complex ROS projects.

4. **rosbash**: rosbash package provides tools that augment the functionality of the bash shell. These tools include rosls, roscd and roscp that have similar

funnction as that of ls,cd and cp.

5. **roslaunch**: roslaunch is a tool that can be used by multiple ROS nodes locally and remotely. It can also set parameters on ROS parameter server. roslaunch configuration files, which are written using XML can easily automate a complex startup and configuration process into a single command. roslaunch scripts can include other roslaunch scripts, launch nodes on specific machines, and even restart processes that die during execution[12]

3.4 Point Clouds

Point Clouds play a pivotal role in several fields like computer vision, robotics, autonomous vehicles, augmented reality, and many more. In computer vision, point clouds enable the precise modeling of real-world scenes, facilitating tasks such as object detection, scene understanding, and 3D reconstruction. Similarly, point clouds play a crucial role in perception and navigation in robotics and autonomous vehicles, aiding in obstacle detection, environment mapping, and path planning[13]. A point cloud is a discrete set of data points in 3D space. Each point in this space has a set of Cartesian co-ordinates(X,Y,Z). Other properties that a point cloud has are RGB colors, normals and timestamp and so on. Point clouds are generated using 3D scanning. LiDAR(Light Detection and Ranging) is one of these techniques which is commonly used in Robotics and Computer Vision. LiDAR sensors emit pulsed laser light and measure the time-of-flight(TOF) to calculate the distance to the target. LiDAR is pretty accurate and effective over long distances. It has also proved effective in different lighting conditions. This technique is mostly used in terrestrial, airborne, and mobile mapping. Another technique used is with the help of a stereo camera. 3D point clouds play an important role in the domain of spatial data representation. In the field of machine learning, point cloud classification method has good applicability for in-vehicle laser point cloud processing and has high research value for improving the automatic and intelligent processing of point clouds. The point cloud feature vector constructed by the contextual semantic environment, local geometric features, and spatial distribution of the vehicle-mounted laser point cloud improves the automation and intelligence level of point cloud classification[14]. Point clouds can also simulate the 3D structure of biological organs, tissues, etc. Algorithms such as classification based on medical point cloud data can assist doctors in more accurate diagnosis and treatment and have important application value in clinical

medicine and medical device-aided design[14]

3.5 ICP(Iterative Closest Point)

ICP is a widely used technique for geometric alignment of 3D models. It is used for point cloud registration that aligns a source point cloud to a target point cloud. The process of determining the spatial relationship between 3D data is called 3D registration. In case of Stereo cameras, we get two sets of point clouds. These point clouds are taken from different viewpoints and exist in different co-ordinate frames. 3D registration transforms the first point cloud to align with the other. This transformation is basically to find the rotation and translation in order to minimize the sum of squared differences between the coordinates of matched pairs. This process is iterated until the two point clouds are aligned. There are a lot of ICP variants, and point-to-point and point-to-plane are the most used. There are a lot of implementations of ICP in online libraries such as Open3D, PCL, Meshlab and so on.

3.5.1 Point-to-point ICP

The objective of Point-to-point ICP is to minimize the squared Euclidean distance between the corresponding points.

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \|\mathbf{p} - \mathbf{T}\mathbf{q}\|^2 \tag{7}$$

The steps to achieve this objective are as below:

1. Find the correspondence set $\mathcal{K} = \{(\mathbf{p}, \mathbf{q})\}$ from the target point cloud P and the source Q transformed with current transformation matrix T[15].

2. Update the transform T by minimizing an objective function E(T) defined over the correspondence set K[15].

3.5.2 Point-to-plane ICP

The objective of Point-to-plane ICP is to make a point lie on the same surface plane as the target cloud. As shown in eq. 8[15], we can formulate the objective such as to minimize the objective function from transformed source point Tq to the tangent plane of target point P with normal $\mathbf{n_p}$. The point-to-plane ICP algorithm has a faster convergence speed than the point-to-point ICP algorithm[15].

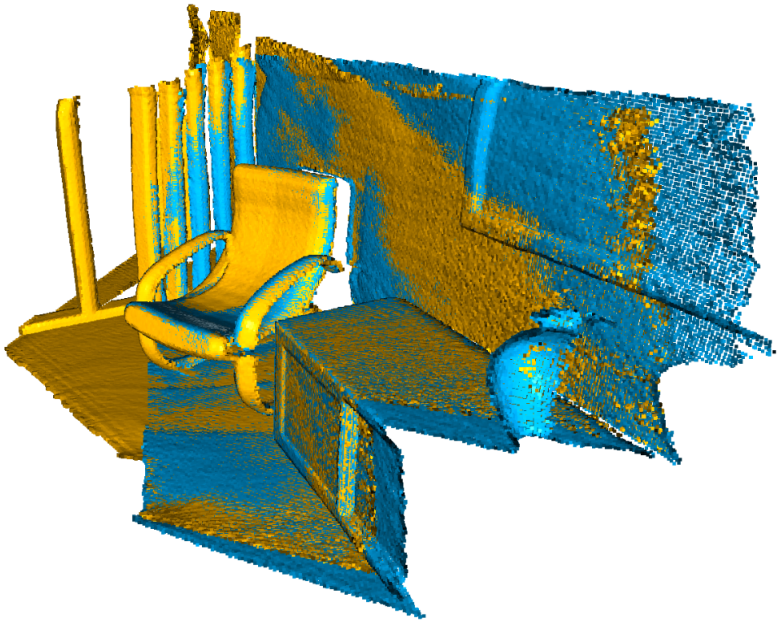


Figure 6: Point to point ICP[15]

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \big((\mathbf{p} - \mathbf{T}\mathbf{q}) \cdot \mathbf{n}_\mathbf{p}\big)^2 \tag{8}$$

3.6 Outlier removal in Point Clouds

Outlier removal is an important pre-processing step in 3D data pipeline. Outliers are points that deviate from the distribution making the points look noisy and deformed. Outliers in point clouds can be sparse and/or isolated. They are mostly caused by sensor issues, occlusions, poor calibration and reflective surface. It is necessary to remove outliers as they can cause problems while doing tasks such as surface reconstruction, registration and feature extraction. There are several outlier removal technique. Some most commonly used methods are as follows:

3.6.1 Statistical Outlier Removal(SOR)

Statistical Outlier Removal technique is one of the most widely used and effective methods of outlier removal. It is based on the k-nearest neighbors principle which states that the points whose mean distance deviates from the average beyond a certain value, are outliers. It is effective against random noise and doesn't affect the original

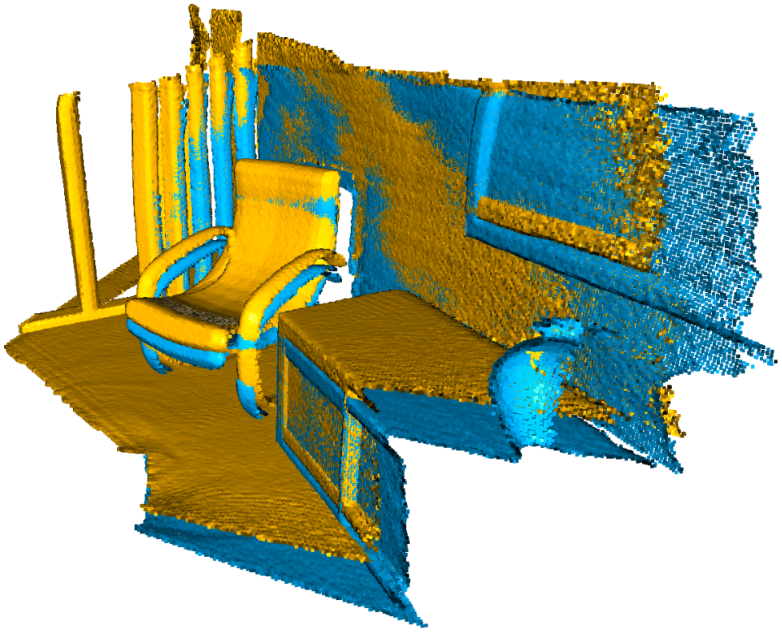


Figure 7: Point to plane ICP[15]

structure. One shortcoming of this method is that it can remove the relevant points near sharp edges.

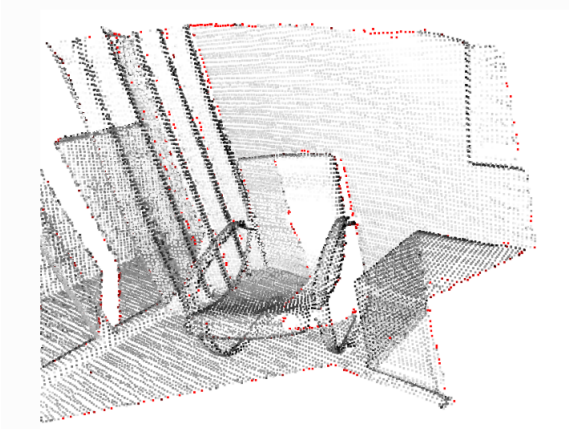


Figure 8: Statistical Outlier Removal[16]

3.6.2 Radius Outlier Removal(ROR)

Radius outlier removal is based on density-based filter. It is therefore mostly suitable where points are dense. It needs to fix a search radius R around each point. The point remains valid as long as it has atleast a given number of neighbors(M). The

remaining points that have less than the given amount of points within the its radius, are then removed. One advantage would be that it can remove the outliers fast and effectively as long as the points are highly isolated. However, an appropriate value for radius(R) and neighbors(M) is necessary. Also, if the density varies a lot then it might end up removing good data in sparse regions and leave noisy points in dense regions.

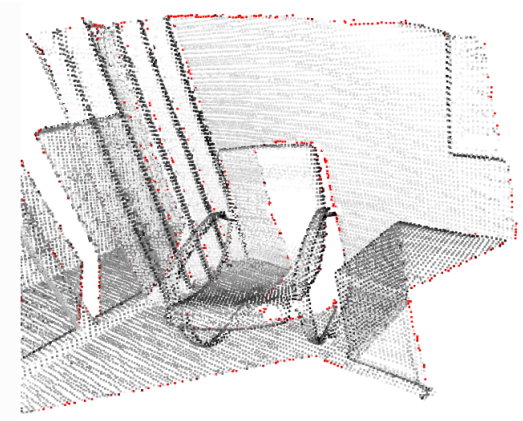


Figure 9: Radius Outlier Removal [16]

3.6.3 Local outlier factor(LOF)

The local outlier factor relies on local density. Local density in this context, is based on the reachability distance. This states that if the average reachability distance from neighbors to a given point A is short, then A has higher density.

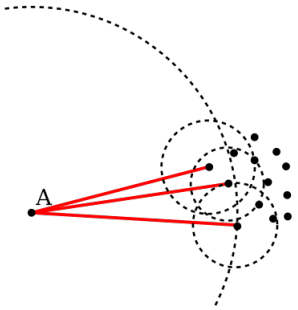


Figure 10: Local outlier factor [16]

Fig.10 shows the comparative analysis of LOF wherein A has a much lower density than its neighbors. The advantage of this method is that, it can identify clustered outliers. It is not ideal for real-time or larger data as it is computationally expensive compared to other methods.

3.7 Surface Reconstruction

3.7.1 Alpha Shapes

Alpha shape is a generalization of the convex hull of a finite set of points in a plane[17]. It creates a crude shape out of straight-line graphs called "α shapes". α shapes are a subgraph of closest points or furthest point in Delaunay triangulation. This algorithm has proven to be helpful in the field of pattern recognition, cluster analysis and computer vision.

Alpha shapes is governed by parameter α that denotes the geometric details of the shapes. By choosing an appropriate alpha, we can get shapes according to how much coarse to fine details we want. The coarse and finer detailing is closely related to convexity. Alpha shapes is an algorithm to materialize a shape out of finite set of points for a real α. It is a generalization of convex hull. A convex hull of a set S on n points in a plane(n being a positive integer) is an intersection of all closed half-planes which contain every point of S. A half space in 2D is the set of points on or to one side of a line[18].For 3D, half-space is the set of points on or to one side of a plane. Convex hull creates a 2D or 3D shapes encapsulating all the set of given points into a polyhedron/polygon. It defines the boundary of the region containing all the points.

$$\text{conv}(S) = \left\{ \sum_{i=1}^{k} \lambda_i p_i \;\middle|\; p_i \in S,\ \lambda_i \geq 0,\ \sum_{i=1}^{k} \lambda_i = 1 \right\} \tag{9}$$

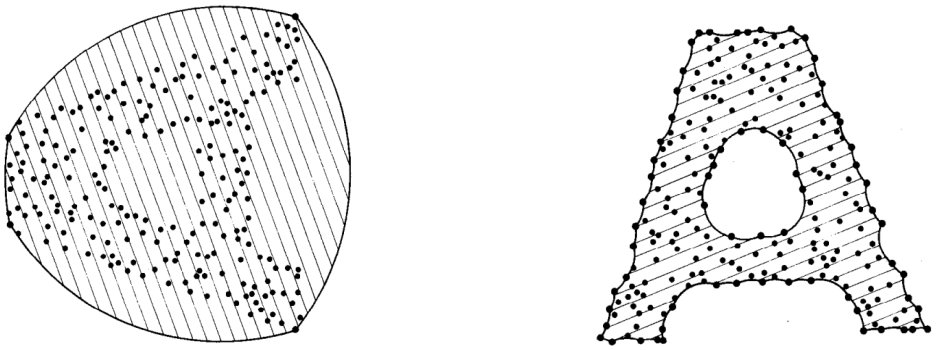
Alpha shapes leverages this quality of convex hull and generalizes it to have better control over how much tighter we want the shape over a given set of points. Therefore, the smaller α gets, the further away it is from being convex. And as α tends to ∞, it becomes a full convex hull. The paper[17] states the properties of α hull as follows:

Given a set S of n points in a plane, the convex hull of S can be defined as an intersection of all the circles that contain all points of S. It studies different formation as the value of alpha changes.

1. Given an arbitrary and positive α, the α hull of S is the intersection of all circles with radius $1/\alpha$ that contain all points of S. It states that large α gives rise to hulls that have only some points that lie on the boundary. As α approaches zero($\alpha \to 0$), the α-hull approximates a convex hull. For a large α, it is equal to the entire plane.

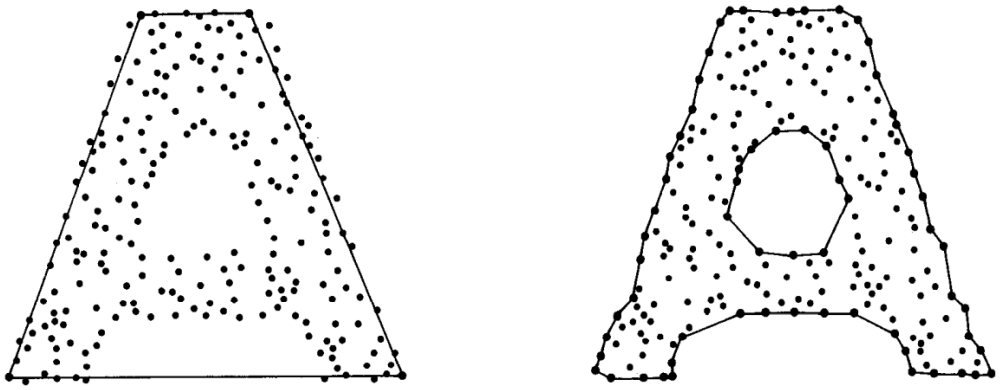
2. For negative α, the α hull is defined as the intersection of all closed complements of circles that contain all points of S. The radius of circle is - $1/\alpha$.

3. For a large $alpha(\alpha \to \infty)$ it equals the entire plane and $(\alpha \to -\infty)$ equals a convex hull.



(a) Positive alpha hull[17] **(b)** Negative alpha hull[17]

Figure 11: Alpha hull



(a) Alpha shape for positive alpha[17] **(b)** Alpha shape for negative alpha[17]

Figure 12: Alpha shapes according to alpha values

After understanding the alpha hull, the paper goes ahead with explaining how to construct an alpha shape. Alpha shapes are basically straight line graph. It is a subgraph of either the closest point or furthest point Delaunay triangulation. In 2D, Delaunay triangulation of a set of points S in the plane subdivides their convex hull into triangles whose circumcircles do not contain any of the points; that is, no point in S lies inside the circumcircle of any triangle in the triangulation[19]. The closest/furthest point Delaunay triangulation is a straight line dual of the Voronoi diagram[20], that is, there is a straight line edge between two points if and only if they are closest/furthest Voronoi neighbors[20]. A Voronoi diagram is a partition of plane into regions(Voronoi cells) close to each of a given set of objects. In other words, every point in a Voronoi cell is at least as close to another point as it is to any other point in a set. Voronoi neighbors are a set of points whose Voronoi cells share a boundary(face/edge). If two triangles share an edge in the Delaunay triangulation, their circumcenters are to be connected with an edge in the Voronoi pattern[20]. The duality of Voronoi diagram and the Delaunay triangulation can be described using the following:

1. If two triangles share an edge in the Delaunay triangulation, their circumcenters are to be connected with an edge in the Voronoi tesselation.

2. Each Voronoi edge is perpendicular to the corresponding Delaunay edge.

Each vertex of an α shape is also a vertex of the respective Delaunay triangulation. Also, if two points are α neighbors, they are adjacent in the respective Delaunay triangulation. The paper[17] discusses in detail about the closest and the furthest Voronoi diagram as it is closely related to the closest and furthest Delaunay Triangulation. The paper gives an in-depth study about both the variations in Delaunay Triangulation. The algorithm to construct an α shape is detailed in the paper[17].

The advantages of this method are, that it gives a scale control using the parameter α to choose a finer or coarser shape. This makes it useful in the areas such as pattern recognition, cluster analysis and computational geometry. It however has some disadvantages also. The freedom of using the value of α, also leaves us with the responsibility to use the appropriate value of α. It has to be tried and tested and requires an understanding of the workings of this algorithm to choose the appropriate value. The finer details might give a non uniform shape with holes and disproportion.

3.7.2 Ball-Pivoting Algorithm(BPA)

Ball pivoting algorithm is surface reconstruction technique that constructs triangle meshes from given set of points from a point cloud. The mechanism is simple. If a ball with a user defined radius ρ touches three points without containing any other point, then the three points form a triangle. It is closely related to the Alpha shapes. It is efficient and scalable to large point clouds. One important thing to note is that the algorithm exhibits linear time performance. It does not require the input data to be loaded into memory simultaneously making it storage efficient as well. The final triangle mesh is incrementally saved to an external storage while running. BPA has also been proven robust against noise.

Consider a ball with radius ρ. Place the ball such that it passes through 3 sample points and doesn't touch any other points in the set. Keep the ball in contact with two initial points, pivot the ball until it touches another point. Pivot the ball around each edge of the triangle. The points that it touches, forms new triangles. Starting with a seed triangle, the algorithm keeps pivoting the ball around the edges of this newly constructed mesh. The set of triangles formed in such a way makes up a coherent mesh. The output mesh constructed with the help of BPA is a manifold subset of an alpha-shape. The circle with radius ρ that passes through the 3 points of a triangle, might have a smaller radius and still not touch any other point. Therefore, it satisfies the "empty ball" condition followed by the Alpha shapes. So BPA cannot have a triangle that isn't present in Alpha shapes, it selects from a subset of Alpha shapes. And both of these are a subset of the Delaunay triangulation. Another assumption of BPA is that the normals can be calculated for the all the points in the set. It is crucial to have this information to distinguish the interior and exterior region. In a situation where there are holes larger than ρ, there might not be a clear distinction of whether the surface is inside or outside. Therefore, the normals are used for consistent orientation. The algorithm assumes that the normals point outward. Incase of missing points, the algorithm can be applied multiple times with increasing radii, making it efficient in the sparser sections. When pivoting around a boundary edge, the ball can touch an unused point lying close to the surface. Again the BPA uses surface normals to decide whether the point touched is valid or not . A triangle is rejected if the dot product of its normal with the surface normal is negative[21].

The advantages of this algorithm include being easy to understand and implement. It is also efficient and scalable as mentioned. And it is also robust to noise. Having said that, it has its disadvantages also. It assumes the point cloud to be dense. Although it can still work with a proper selection of the radius parameter ρ. A

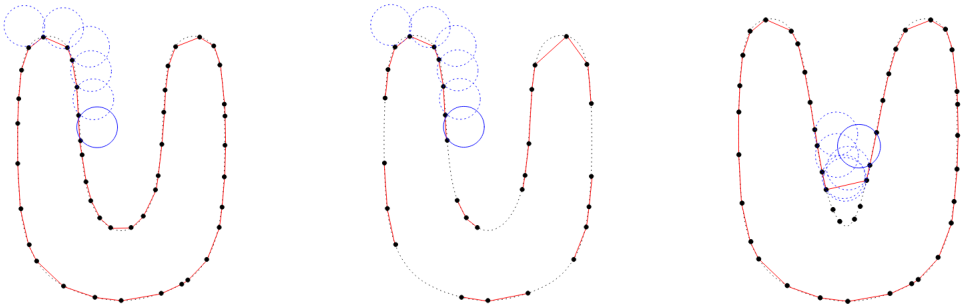


Figure 13: Ball-Pivoting Algorithm[21]

smaller radius circle might cause holes in the shape while on the other hand, if the radius of the circle is large then it causes loss of finer details. Therefore the radius ρ should be chosen appropriately.

3.7.3 Poisson Surface Reconstruction

Poisson surface reconstruction is a technique that uses an implicit function framework for surface reconstruction. It computes a 3D indicator function χ and constructs surface by extracting an appropriate isosurface. Poisson surface reconstruction leverages the relationship between the points and the indicator function[22]. This technique considers all the given points and doesn't need spatial partitioning or blending. It works globally and therefore highly resistant to noise.

Unlike [23] that uses Stokes Theorem to define the Fourier co-efficients of the indicator function, it uses Poisson equation. Although both use the indicator function to extract the isosurface. An isosurface is a surface that represents points of constant value within a volume of space[24]. Eq.10explains the indicator function.

$$\chi(p) = \begin{cases} 1, & \text{if } p \text{ is inside the model} \\ 0, & \text{if } p \text{ is outside the model} \end{cases} \tag{10}$$

The indicator function χ is defined such that it is 1 when p is inside and 0 when p is outside. Now, when it comes to the boundary, χ transitions from inside to outside. In interior and exterior regions, χ is constant so $\nabla\chi$ is zero. However at the boundary, the gradient of χ is non-zero. It is equal to the inward surface normals.

The oriented point samples can be seen as the samples of gradient of the indicator function. To get the value of χ, the paper takes the inverse of the gradients. This can be posed as variational problem as below:

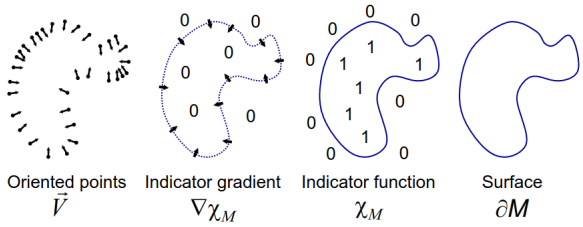


Figure 14: Poisson indicator function[23]

$\min_\chi \|\nabla\chi - \vec{\mathbf{V}}\|$

If we apply the divergence operator, this variational problem transforms into a standard Poisson problem. It can be stated as: To compute the scalar function χ whose Laplacian (divergence of gradient) equals the divergence of the vector field[22] is given by $\Delta\chi = \nabla \cdot \mathbf{V}$.

This algorithm solves the poisson equation globally rather than locally. Solving this equation globally, will give a smooth χ whose gradient is almost similar to the input normals. Hence, it needs to consider the entire 3D grid. But doing so requires a lot of memory and computation power. The main concern here is to convert the continuous poisson equation into a solvable discrete system. For this, it makes use of adaptive Octree. An octree subdivides 3D space recursively. It creates set of nodes where the unknown solution values are computed. The octree nodes define the centers of the locally supported basis functions(B-splines). This solution then is an approximated sum of these functions. After solving the equation, the continuous indicator function is known globally. The final surface mesh is calculated as zero-level set, using the Marching cubes algorithm.

Poisson surface reconstruction therefore, creates a watertight and closed mesh. This makes sure that there are no holes or gaps which is a significant issue in local reconstruction methods. Another advantage here is that the indicator function needs to be detailed only near the surface boundary. Therefore this makes the algorithm more robust to large point clouds. Using an adaptive octree and a hierarchical Multi grid solver, the computational complexity is linear. But there are some disadvantages as well. This method requires accurate and correctly oriented surface normals. It also creates only closed and solid model. So it often tries to close the open boundaries with an artificial cap and therefore produces artifacts near the edges.

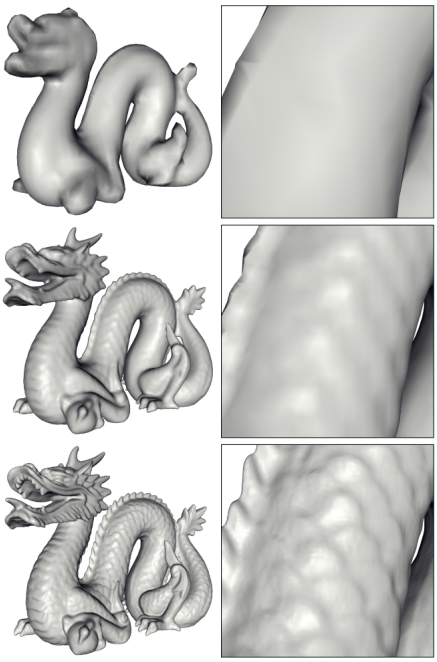


Figure 15: Poisson reconstructions of the dragon model at octree depths 6 (top), 8 (middle), and 10 (bottom)[23]

4 Approach

The approach for this 3D modeling project involves a multi-stage pipeline. The input is data acquisition from Stereo camera using ROS and the output is a 3D model.

4.1 Point Cloud Data Generation and Acquisition

The initial process involves utilizing a stereo camera discussed in Section 3.1, to capture the object in the scene. To get a complete 3D representation from all viewpoints, a dataset of 20 images was captured for each object. The main data representation for this project is point cloud discussed in Section 3.4. These point clouds are extracted and transferred using Ensenso SDK and ROS discussed in Section 3.3. The Ensenso camera node acts as a Server node which publishes the final Point Cloud. This node is responsible for listening to the incoming requests and sending the data back to the requesting client. The client node subscribes to this Master node and initiates the data acquisition.

4.2 Point Cloud Processing Pipeline

Once the point clouds are received, it is necessary to process these point clouds to get a coherent and dense 3D structure. This processing is done using Open3D library. Open3D is a preferable tool over other tools like PCL(Point Cloud Library) because of its robust build-in visualization tools and python language compatibility. Cropping the object can be done by removing the points from the gripper. This can be easily done as the points from the gripper are already available.

4.3 Noise reduction and Alignment

Raw point clouds often contain noise. This noise can be removed using effective outlier removal techniques discussed in Section 3.6. This step is also important so that it does not negatively affect the alignment process. Point clouds captured from

different perspectives often exist in different local coordinate frames. This results in shift between scans. To merge them together into a coherent model, its important to align them. This is done with local registration techniques and ICP(Section 3.5) to get to a final coherent model.

4.4 3D Object Model Creation

The final step is converting this processed point cloud into a 3D mesh model. This can be done in various ways. The algorithms used in this project are based on one of the most effective strategies. Also, these are included in the Open3D library. The three surface reconstruction strategies are:

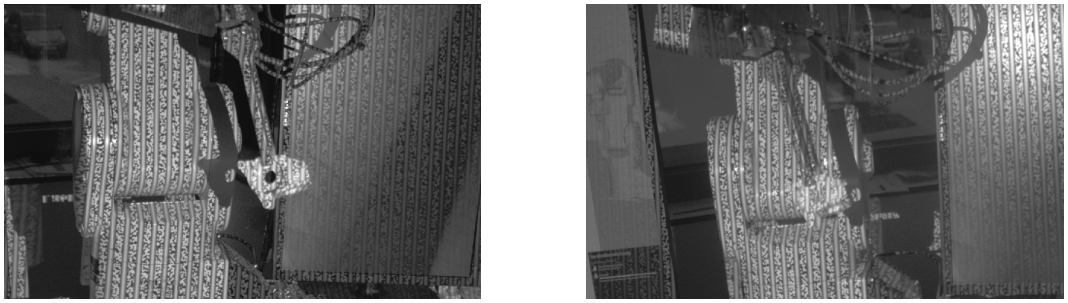
1. Alpha Shapes: This technique defines shapes based on parameter α. It is based on Delaunay triangulation. It is useful in cases where having watertight geometric model is not a requirement(Section 3.7.1).

2. Ball Pivoting Algorithm: This technique creates mesh by rolling a virtual ball over the given points. This works best when the point cloud is dense(Section 3.7.2).

3. Poisson Surface Reconstruction: This technique creates watertight meshes. It is also robust to noise. This is a preferred technique when volumetric model is desired(Section 3.7.3)

5 Experiments

5.1 Stereo Camera Images



(a) Stereo camera image left

(b) Stereo camera image right

Figure 16: Stereo camera images with Robot gripper and object

5.2 Point Cloud Visualization

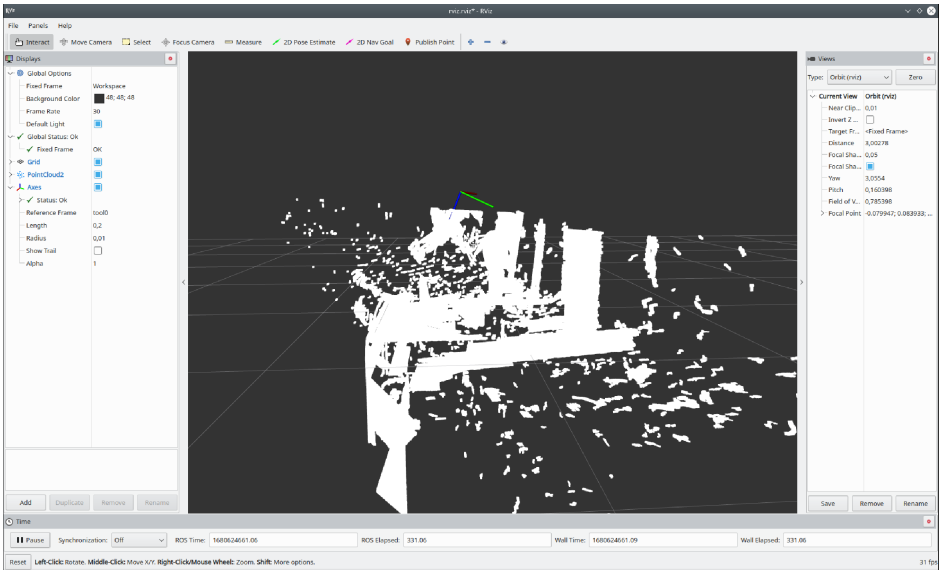
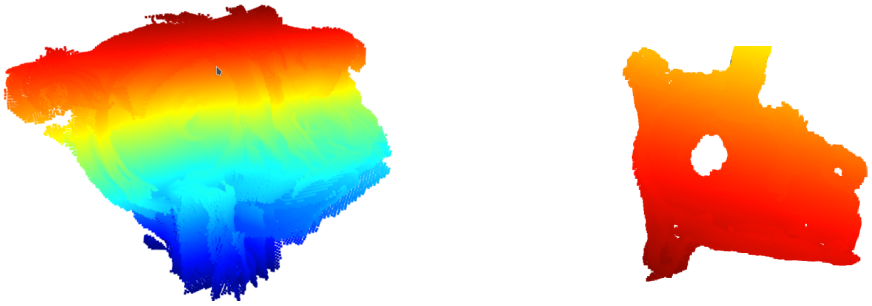


Figure 17: Point Cloud Visualization using RVIZ

Figure 18: Cropped Point Cloud with gripper and object

5.3 ICP registration



(a) Object model before applying registration **(b)** Object model after applying registration

Figure 19: Before and after applying ICP registration

5.4 Surface Reconstruction

5.4.1 Alpha Shapes

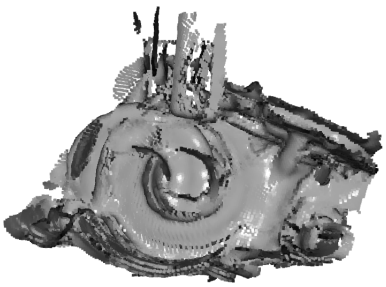


Figure 20: Original Point cloud

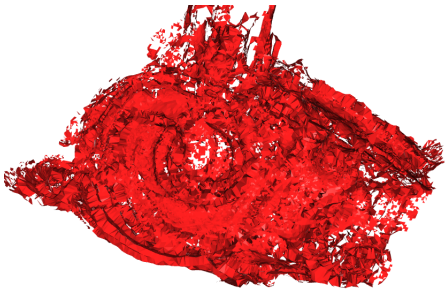


Figure 21: Alpha shapes with α value 0.001215

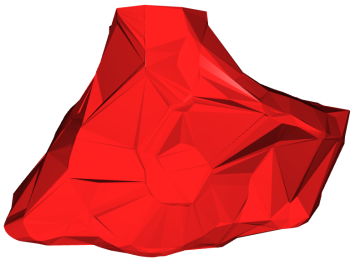


Figure 22: Alpha shapes with α value 0.015

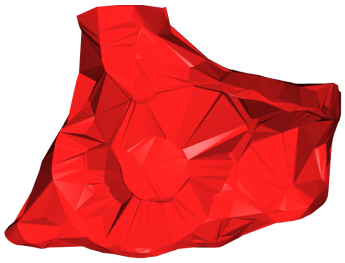


Figure 23: Alpha shapes with α value 0.0098

5.4.2 Ball Pivoting Algorithm

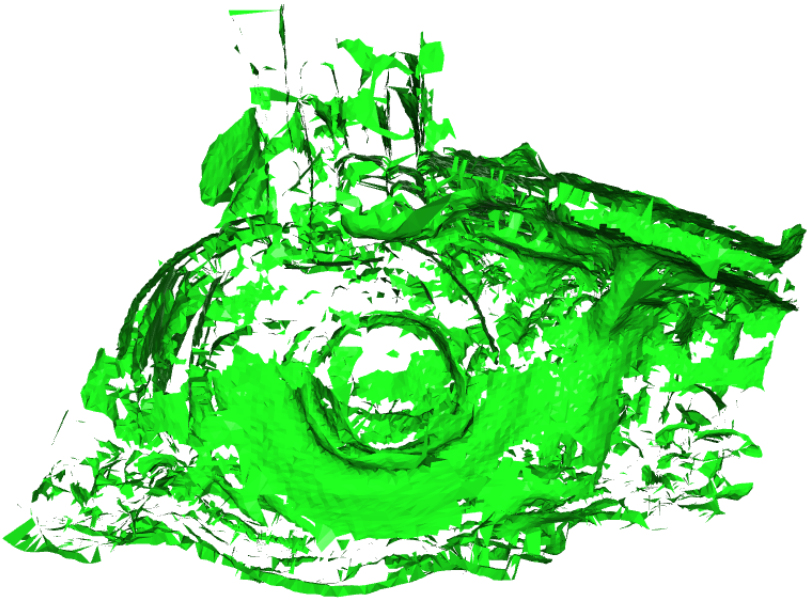


Figure 24: Ball pivoting with radii 0.0010,0.001620,0.0028

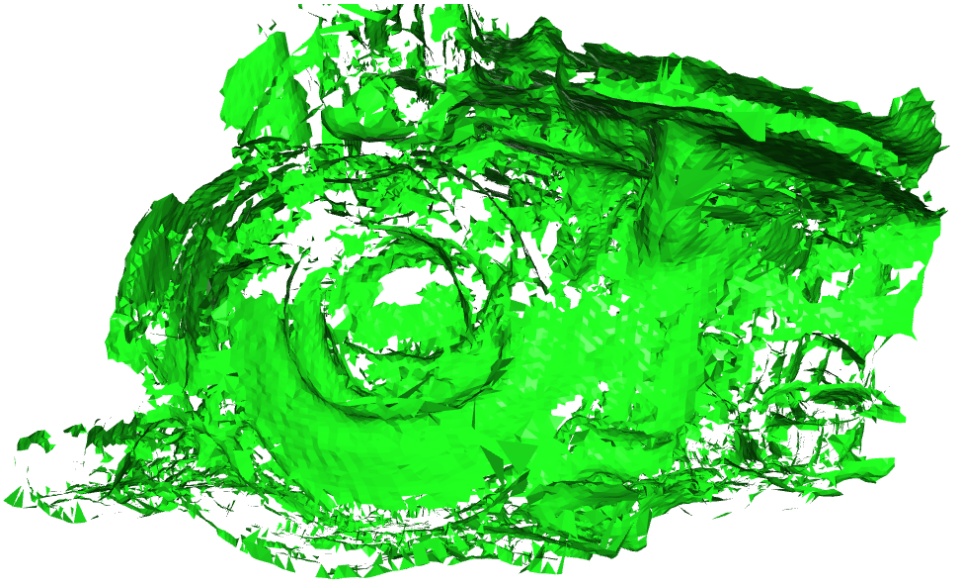


Figure 25: Ball pivoting with radii 0.0006,0.001215,0.0024

5.4.3 Poisson Surface Reconstruction

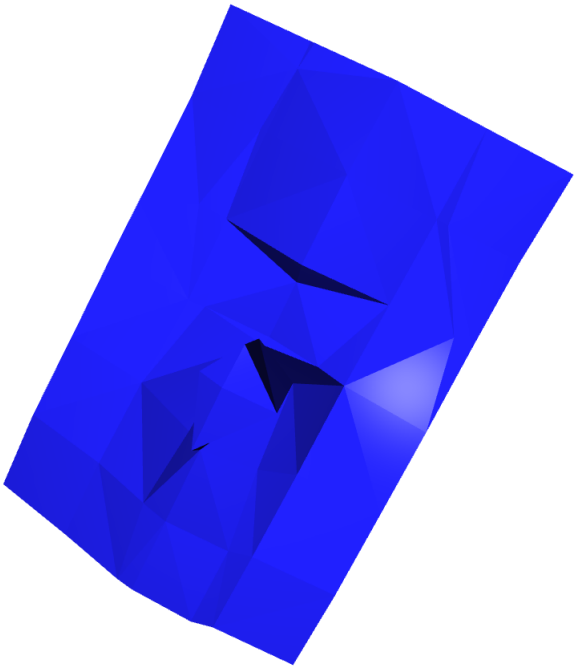
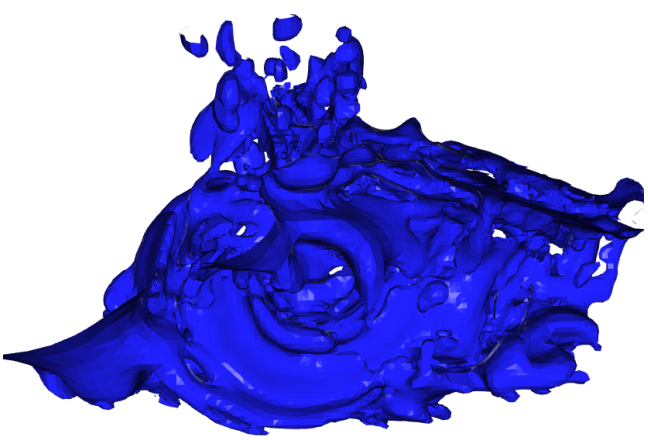
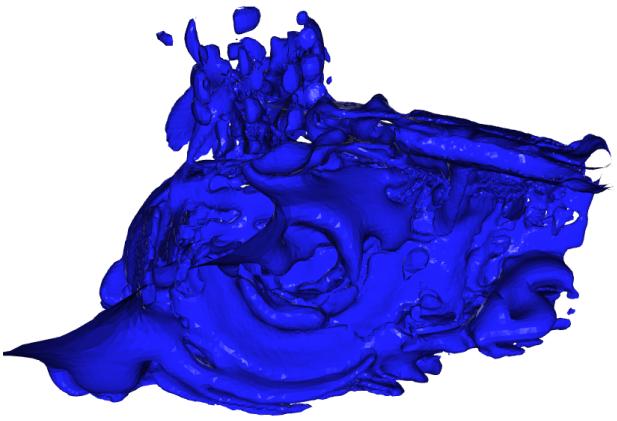


Figure 26: Poisson with octree depth 2



(a) Poisson with octree depth 7



(b) Poisson with octree depth 9

6 Conclusion

The goal of this work was to develop accurate 3D object models using Stereo vision. This goal is achieved by implementing a robust methodology using existing frameworks like ROS and Open3D. The pipeline consists of acquiring the images through a stereo camera. Once done, integrating the ROS framework ensures the system is effective and scalable. It can well adapt to various robotic platforms and sensors. Conversion of these images into dense 3D structures of points gives a flexibility to process the data in a better way and paves the path for efficient Surface reconstruction. Ultimately, different methods were studied and tested to create a reliable 3D object model. This contributes to the advancement of robotic applications for improved dexterity in unknown environments. Future work can focus on optimizing the point cloud registration and utilizing the growing power of neural networks.

Bibliography

[1] Ensenso, "Ensenso manual." https://manual.ensenso.com/3.0/guides/calibration/calibrationpatternsUN85.html, 2025.

[2] Optonic GmbH, "Mikado." https://www.optonic.com/en/brands/mikado/, 2025.

[3] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, pp. 586–606, Spie, 1992.

[4] 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.

[5] J. Salvi, J. Pages, and J. Batlle, "Pattern codification strategies in structured light systems," *Pattern recognition*, vol. 37, no. 4, pp. 827–849, 2004.

[6] D. Tzionas and J. Gall, "3d object reconstruction from hand-object interactions," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 729–737, 2015.

[7] T. Weise, T. Wismer, B. Leibe, and L. Van Gool, "In-hand scanning with online loop closure," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pp. 1630–1637, IEEE, 2009.

[8] S. Hampali, T. Hodan, L. Tran, L. Ma, C. Keskin, and V. Lepetit, "In-hand 3d object scanning from an rgb sequence," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17079–17088, 2023.

[9] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger, "Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3504–3515, 2020.

[10] M. Oechsle, S. Peng, and A. Geiger, "Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5589–5599, 2021.

[11] L. Deligiannidis and H. R. Arabnia, "Emerging trends in image processing, computer vision and pattern recognition," 2014.

[12] Wikipedia, "Robot operating system (ros)." `https://en.wikipedia.org/wiki/Robot_Operating_System`, 2025.

[13] V. Thengane, X. Zhu, S. Bouzerdoum, S. L. Phung, and Y. Li, "Foundational models for 3d point clouds: A survey and outlook," *arXiv preprint arXiv:2501.18594*, 2025.

[14] H. Zhang, C. Wang, S. Tian, B. Lu, L. Zhang, X. Ning, and X. Bai, "Deep learning-based 3d point cloud classification: A systematic survey and outlook," *Displays*, vol. 79, p. 102456, 2023.

[15] Open3D, "Icp." `https://www.open3d.org/docs/release/tutorial/pipelines/icp_registration.html`, 2025.

[16] Open3D, "Outlier removal." `https://www.open3d.org/docs/latest/tutorial/Advanced/pointcloud_outlier_removal.html`, 2025.

[17] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on information theory*, vol. 29, no. 4, pp. 551–559, 2003.

[18] Brilliant, "Convex hull." `https://brilliant.org/wiki/convex-hull/`, 2025.

[19] Wikipedia, "Convex hull." `https://en.wikipedia.org/wiki/Delaunay_triangulation`, 2025.

[20] Wikipedia, "Voronoi diagram." `https://en.wikipedia.org/wiki/Voronoi_diagram`, 2025.

[21] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE transactions on visualization and computer graphics*, vol. 5, no. 4, pp. 349–359, 2002.

[22] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[23] M. Kazhdan, "Reconstruction of solid models from oriented point sets," in *Proceedings of the third Eurographics symposium on Geometry processing*, pp. 73–es, 2005.

[24] Wikipedia, "Isosurface." `https://en.wikipedia.org/wiki/Isosurface`, 2025.