

https://dspace.org/

Pipeline for the 3D Reconstruction of Rigid, Handheld Objects through the Use of Static Cameras

Kambadkone, Saatwik Ramakrishna

2023-04

University of the Witwatersrand, Johannesburg

https://hdl.handle.net/10539/41923

Kambadkone, Saatwik Ramakrishna. (2023). Pipeline for the 3D Reconstruction of Rigid, Handheld Objects through the Use of Static Cameras. [Master's dissertation, University of the Witwatersrand, Johannesburg]. https://hdl.handle.net/10539/41923

Downloaded from a DSpace repository. https://dspace.org/

Pipeline for the 3D Reconstruction of Rigid, Handheld Objects through the Use of Static Cameras

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Saatwik Kambadkone
1601869

Supervised by Prof Richard Klein

July 4, 2023



A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, South Africa, in fulfilment of the requirements for the degree of Master of Science

Abstract

In this paper, we develop a pipeline for the 3D reconstruction of handheld objects using a single, static RGB-D camera. We also create a general pipeline to describe the process of handheld object reconstruction. This general pipeline suggests the deconstruction of this task into three main constituents: input, where we decide our main method of data capture; segmentation and tracking, where we identify and track the relevant parts of our captured data; and reconstruction where we develop a method for reconstructing our previous information into 3D models.

We successfully create a handheld object reconstruction method using a depth sensor as our input; hand tracking, depth segmentation and optical flow to retrieve relevant information; and reconstruction through the use of ICP and TSDF maps. During this process, we also evaluate other possible variations of this successful method. In one of these variations, we test the effect of using depth-estimation to generate data as the input to our pipeline. While this experimentation helps us quantify our method's robustness to noise in the input data, we do conclude that current depth estimation techniques do not provide adequate detail for the reconstruction of handheld objects.

Declaration

I, Saatwik Kambadkone, hereby declare the contents of this research proposal to be my own work. This proposal is submitted for the degree of Masters in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.



July 4, 2023

Signature

Date

Acknowledgements

I would like to thank my family, my partner and my supervisor for all their support through this dissertation.

Thank you to the Mathematical Sciences Support at the University of the Witwatersrand for access to their facilities to perform training of networks.

I am also extremely grateful to Jason Chalom for all their help with the process of 3D printing.

Contents

Preface
Abstract i
Declaration ii
Acknowledgements iii
Table of Contents iv

1 Introduction **1**

2 Background and Related Work **4**
2.1 Scene Reconstruction 5
2.1.1 Structure-from-Motion 5
2.1.2 Visual Simultaneous Localization and Mapping 6
2.1.3 KinectFusion 9
2.2 Object Reconstruction 10
2.2.1 Assisted Object Reconstruction 10
2.2.2 Object Pose Estimation 11
2.2.3 In Hand Scanning 12
2.3 Related work 16

3 General In-Hand Reconstruction Pipeline **19**
3.1 Input 20
3.2 Tracking and Segmentation 20
3.3 Reconstruction 21
3.4 Evaluation 22

4 Input **24**
4.1 Introduction 24
4.2 Equipment 24
4.2.1 Depth-Sensing 25
4.2.2 Advantages 27
4.2.3 Disadvantages 27

5 Segmentation and Tracking **29**
5.1 Introduction 29
5.2 Segmentation through Learning 29
5.3 Segmentation through Hand Pose Estimation 34

5.3.1 Hand Pose Estimation . . . 35
5.3.2 Skin-Segmentation . . . 37
5.3.3 Noise Removal and Optimizations . . . 38

6 Reconstruction **41**
6.1 Introduction . . . 41
6.2 Frame-to-Frame Comparisons . . . 41
6.2.1 Multi-Resolution ICP . . . 43
6.2.2 Point-to-Plane . . . 44
6.2.3 Coloured ICP . . . 45
6.2.4 Applied Approach . . . 45
6.3 Model Connectivity . . . 46
6.4 Model Integration . . . 48
6.4.1 Evaluation Metrics . . . 49
6.5 Testing and Experimental Setup . . . 50
6.5.1 Optimal Conditions for Target . . . 51
6.5.2 Object Augmentation . . . 51
6.6 Results . . . 53

7 Depth Estimation **58**
7.1 Introduction . . . 58
7.2 Monocular Depth Estimation . . . 58
7.3 Estimation Noise Simulation . . . 60
7.4 Evaluation and Comparison . . . 61

8 Conclusion and Future Work **64**
8.1 Conclusion . . . 64
8.2 Future Work . . . 65

References **74**

Chapter 1

Introduction

The act of digitizing structures and scenes in the real world is extremely useful for tasks such as the preservation of historical landmarks and medical research. By producing accurate representations of different objects or locations, we can interact with their digital versions in a manner that was previously restricted. Introducing digital versions of archaeologically important structures and locales allows us to interact with them without the worry of damage while also circumventing location-based restrictions.

By using a combination of both current VR technology and 3D scanning, it should not be difficult to simulate visiting places out of our reach. These digital scenes can be further filled in with separately scanned objects to fill the environment with more user-friendly interactions. Scenarios like lockdowns further gear us to be less mobile and in turn, less likely to travel than usual. We can overcome the effect of these restrictions through the use of simulated experiences such as virtual museums and tours [Carrozzino and Bergamasco 2010].

In the medical field, creating 3D reconstructions of different parts of the body has many uses. A 3D model of biological structures can provide leagues more information than 2D information. Performing complex surgeries remotely is much more reliable when we have an accurate representation of the point of interest. Depth information is vital when performing complicated surgeries that require pinpoint accuracy and building a 3D model of the premise can help this process become a reality.

By making use of depth sensors, we can record the spatial information of a scene. Using this, we can start building a 3D model of the subject. This entails aligning each pixel with a position in 3D space and over time updating this information until we have established a collection of 3D points. Then between each of the established points, we can create a collection of surfaces that describe a 3D model [Remondino 2003]. As opposed to the reliance on depth sensors there are also well-researched methods of estimating depth and point cloud positions from 2D information [Remondino and El-Hakim 2006].

In recent years, there has been a slew of applications that bring this complex capability to commercial devices such as mobile phones. Applications such as SCANN3D and Trnio allow us to create 3D models of objects by simply moving a smartphone around the object. The applications take multiple images from different views and then process this information into a 3D reconstruction of the object. Several restrictions arise due to the inability of commercial devices to incorporate variable methods: Among others, the objects cannot be too dynamic, the objects in the scene need to be well lit and the background is considered part of the object.

In order to introduce variability to our considered approach, we propose the use of a modular pipeline. This pipeline aims to provide constant access to the overarching needs of our system while allowing us to consider multiple approaches and possible scenarios that might arise in an uncontrolled environment.

Along with this, one of the main goals of our research is to highlight an alternative approach for recording the shape of an object. When attempting to record this shape we will be working with a dynamic object and a fixed monocular camera. The dynamic object, in our case, is a rigid handheld object that introduces the challenge of movement and rotation to the scene. These rotations and transformations allow the camera to view different surfaces of the object. As the object is moved around the scene, we want to be able to track and record the shape of the object.

While working through this method we need to keep in mind the challenges we will face while attempting to track our object. The object is both unknown and also exists in a scene with constant hand occlusions. Often methods that need to record a shape cannot differentiate between different elements in a scene. All objects that have been recorded are usually described as part of the same structure. To segment different elements, the method needs to have an understanding of the different objects it will interact with. This understanding is often introduced in a categorical manner where groups such as tables, chairs, or cars are learned. In our case, cannot learn the type of object we need to segment as this information is unknown.

Instead we will be using our occlusion in the scene to understand where our object is. Our initial approach to this problem seeks to utilize RGB-D data to attempt learn the concept of a handheld object. We first attempt to train a neural network to understand this concept by showing it annotations of hands and handheld objects in scenes. Following this, we also attempt to track and segment the handheld object using its proximity and interactions with the hand. Once this is done and we establish where the object we are tracking is, we need to understand what part of the object we are modelling in each frame. Through this process we iteratively build our understanding of the object's shape to perform reconstruction.

The following chapters, will describe our approach in detail while providing further information on our proposed pipeline. Chapter 2 describes 3D Reconstruction and its intricacies as we move from the reconstruction of a scene to the reconstruction of an object. Understanding this transition helps in formulating the different possible approaches that can be implemented. This chapter also discusses other methods that perform this task. Chapter 3 provides a deeper understanding of our proposed research. This chapter describes the pipeline structure we will be implementing as well as a summarized breakdown of each phases of the pipeline. Chapter 4, 5 and 6 provide a more in-depth look at their respective phase in the pipeline. In these sections we detail the different approaches tested and the results recorded by the end of each phase. Chapter 7 discusses an alternate approach to the method of input in chapter 4 and the results of our testing with this change. Lastly, Chapter 8 discusses the permutability of the pipeline further and introduces a few more experiments that could be implemented with this in mind. Along with this, key aspects of the pipeline are summarized while also providing a few conclusive thoughts.

Chapter 2

Background and Related Work

3D reconstruction is a complex task that has many different existing approaches and uses. This variation exists to cater to the different situations and their corresponding goals. The possible approaches range from those that aim to create dynamic models of humans to those that aim to model an entire room.

The main overarching goal of 3D reconstruction, regardless of the target, is to work out the shape of an object. Using the information we are given, we need to understand the structure of our subject. By capturing shape information, we can generate a wide range of unique views for an object, even if we do not have physical access. This can allow us to interact with previously inaccessible objects in new ways. Having a digitized version of an object allows us to place them in virtual environments such as games. These digitized versions can then also be used to create replicas through the use of 3D printing.

The introduction of commodity depth sensors to the market has largely exploded the amount of research done to perform 3D reconstruction. Models such as the Azure Kinect DK and the Intel RealSense are now largely available to the public and due to this, a large amount of research that makes use of RGB-D data has been performed. RGB-D data can be split up into two components — the RGB information stored in regular images and the depth map of the scene.

A depth map is a method of 3D representation that stores information of a point's location in 3D space. From this, we're able to derive information regarding how far each point is from the camera. The colour of each pixel dictates how far this area is from the camera. The brighter the pixel, the closer it is to the camera. This representation allows us to derive important relations between the different objects in the scene. We can use this information to accurately segment different objects and understand how far they are from each other. This information can also be supplemented by the RGB data to gather a better understanding and even more information on the scene. If two points are close together on the depth map, we can rely on RGB data to provide more information on if these points lie on the same object.

3D reconstruction is a well-established field with numerous benefits for society. For example, it can be used to explore new environments and find more efficient routes

[Nikoohemat *et al.* 2020], to digitize and preserve important subjects such as crime scenes [Raneri 2018], and to document historical architecture and artefacts [Xu *et al.* 2017]. 3D reconstructions enable us to interact with and study these subjects in ways that would not be possible due to the risk of physical degradation.

2.1 Scene Reconstruction

The research done on digitizing larger structures has always been a reliable base in which to test 3D scanning methods. Larger scenes contain a lot of interesting features, making them ideal targets for using shape analysis techniques on. The following techniques are used to decipher shape information. These techniques also translate well to the smaller scale that we work with for our research.

2.1.1 Structure-from-Motion

Structure-from-Motion (SfM) is one of the first, widely-used approaches to recreating shape information. By taking visual information from a series of images with different views of the same subject, SfM aims to triangulate 3D coordinates [Longuet-Higgins 1981]. This in turn, over time, allows us to calculate shape information. During this process, we want to perform 3 steps — feature matching, camera motion estimation and lastly 3D reconstruction.

By matching features between different scenes, we can analyse the motion of the camera between different views. Using this information, we can retrieve the 3D coordinates of different points in the scene. As we retrieve more matching points between different scenes, we can further update a 3D space with more coordinates.

Instead of analyzing every point in several images, feature extraction allows us to reduce the number of points we need to compare. An often-used feature extraction method is Scale Invariant Feature Transform (SIFT). SIFT allows us to find local features in our images that are not only invariant to several image transformations such as rotation and scaling but are also robust to occlusions [Lowe 1999]. This means that we can repeatedly and reliably find the same features in different images of the same subject and all of this can be done in real time.

The found features are converted to gradient descriptors that record elements of their respective key point's appearance in the scene. A feature is found by highlighting peak values after the Difference of Gaussian filter is applied over several scale values. Once the feature is found, a descriptor is generated by creating a histogram of gradients surrounding the feature's center. We only take the orientation of the gradients as including the magnitude would cause the feature to be influenced by the brightness of the pixels.

We use the principal gradient bin of this histogram as the normalizing direction. By normalizing all these features according to this rule, we can achieve rotation invariance. We further achieve scale invariance by scaling all the descriptors according to the size used when detecting the feature with the Difference of Gaussian filter. As a result

of the normalization, the SfM solution has an easier time when comparing different histograms together.

$$\mathbf{d}(\mathbf{H}_1, \mathbf{H}_2) = \sqrt{\sum_{i=1}^{k} (\mathbf{H}_1(i) - \mathbf{H}_2(i))^2} \tag{2.1}$$

A simple way of comparing two descriptors would be to find the L2 distance, $\mathbf{d}$, between histograms, as shown in equation 2.1. We compare each bin of histogram $\mathbf{H}_1$ against its corresponding bin in histogram $\mathbf{H}_2$. Since the descriptor has been normalized, the corresponding bin of a matching feature should have the same index i. The differences of these bins are summed up which then acts as the loss. As a result, the closer the value of $\mathbf{d}$ is to zero, the more likely it is that the descriptors are of the same feature.

Whenever a new view is introduced, these descriptors need to be compared to every other view we have previously stored. As a result, this method is quite expensive when applied to a very large scene in its base state but there are methods that solve these issues. If we have information of the camera's position between frames, we can use this to determine when we look for correspondences. By grouping views based on their distance from each other, we can reduce the number of frames we need to check such as done by Jared *et al.* [2015].

Next, correspondences between different views have to be checked to verify if they actually describe similar scenes. To do this, we take every pair and try to estimate a transformation that will convert one of the views to match the other. If there are valid correspondences between frames, when the transformation is applied, the matching descriptors will align over each other. Over time, using this information, we build a graph of different views. The different views of the subject act as the nodes and the confirmed correspondences are the edges of a graph. Once we have compared and verified every view, the final graph is a web of views that together can build a 3D view of the subject.

At this point, we can start the reconstruction of the model. As we traverse through the graph, we have to estimate the camera pose of each view we use. By making use of the feature correspondences, we estimate the camera's pose in each frame to stitch together a 3D view of the structure. With every pose, we determine we have another set of correspondences overlapping. These overlapping points then become the scene points that describe the 3D position of key points. Under perfect circumstances, this is enough but due to the nature of estimations, there is some degree of uncertainty in these assigned poses.

2.1.2 Visual Simultaneous Localization and Mapping

Another method of building a 3D scene is through the process of Simultaneous Localization and Mapping (SLAM) introduced in Durrant-Whyte and Bailey [2006]. Through SLAM we attempt to solve two complex questions concurrently. The authors are both

aiming to map a room with the use of an agent as well as attempting to determine the location of the agent in the map.

As the agent traverses through the area, one of its main goals is to pick up features and iteratively build a list of locations it recognizes. As it explores more, it can more reliably notice features that it has seen before and as a result determine its position in the map it has created. Parallel to this, when the agent moves to a novel location it is also tasked with recognizing it is in an unknown space.

We can describe the SLAM problem by explaining the relationship between the data we have and the information we want. In the base application of SLAM, we have a list of the completed actions and their corresponding observations. These are denoted by $\mathbf{u}_k$ and $\mathbf{z}_k$, each performed at time k.

From this information, we want to retrieve an overarching representation of the map, denoted by $\mathbf{m}$, and the position of an agent at each point in time, $\mathbf{x}_k$. While $\mathbf{u}_k$ and $\mathbf{z}_k$ give us the details of what occurred at each point in time, we need to remember that the $\mathbf{x}_k$ retains an extra data point meant to record the initial position of the agent.

As shown in figure 2.1, position $\mathbf{x}_k$ is calculated by using the information from the previous position $\mathbf{x}_{k-1}$ and then simulating the result of a completed action $\mathbf{u}_k$. Once the current action has been completed, the agent records information regarding its surroundings as an observation in $\mathbf{z}_k$. This information, along with the other known values and the relationships we build between observations and their corresponding positions, can build a map $\mathbf{m}$. As we build this map $\mathbf{m}$, we can then in turn give context to a new observation to determine where in the map this information lies.

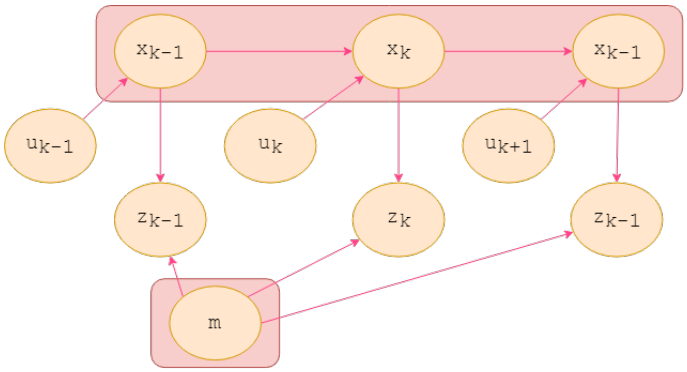


Figure 2.1: Diagram showcasing the process of offline loop closure. The red block highlights which nodes of information are the unknowns to be calculated.

Taking this into consideration, we can further describe the SLAM problem as shown in equation 2.2. This probabilistic distribution of SLAM is read as an estimate of the path an agent takes and an estimate of the map given the controls executed and the observations recorded up until time K.

$$\mathbf{p}(\mathbf{x}_{0:K}, \mathbf{m} | \mathbf{z}_{1:K}, \mathbf{u}_{1:K}) \tag{2.2}$$

If time K refers to the end of the data recording time, we can assume we are making use of an offline SLAM solution. In an offline solution, we make use of all the data we have to determine the shape of a completed map as well as the entire path the agent has taken.

Another approach to the SLAM problem is the online approach. Here, we assume that we need active estimates of the object's position in the scene. This type of solution where we need to constantly update the $\mathbf{x}_k$ variable and the $\mathbf{m}$ representation, is the more widely used approach when agents are expected to be autonomous. In these cases, the agent needs to have access to a constantly updated map.

$$\mathbf{p}(\mathbf{x}_k, \mathbf{m} | \mathbf{z}_{1:k}, \mathbf{u}_{1:k}) \tag{2.3}$$

Online solutions introduce the flexibility of being able to understand when there is an obstacle and choose actions as a result of this information [Anderson *et al.* 2015]. We can consider a new probabilistic model to describe this actively changing position at the current time k, as shown in equation 2.3. This probabilistic distribution of online SLAM is read as an estimated current position at time k and an estimate of the map given the controls executed and the observations recorded up until current time k as shown in diagram 2.2.

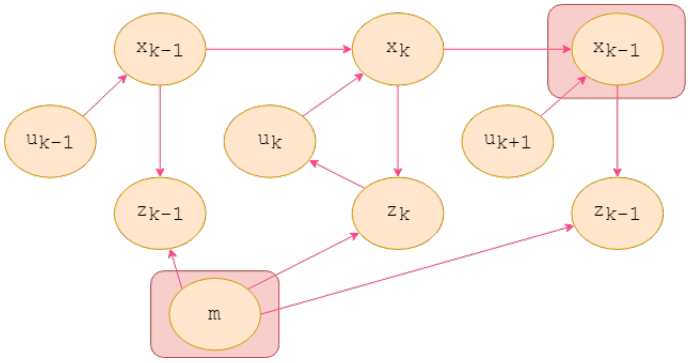


Figure 2.2: Diagram showcasing the process of online loop closure. The red block highlights which nodes of information are the unknowns to be calculated. The previous positions of the agent are considered known information in this approach.

Our online SLAM approach can be run in real time as we are often more concerned with the current position. This means that we do not have to be concerned about the entire previous path of the object as this information has often already been incorporated into the map. This also contributes to the reason why online SLAM can be run in real time. Here, we do not need to optimize the path of the agent's previous positions in the map as we would in offline SLAM. We can instead calculate where in the map we are by

using the last few positions and observations and their interactions with the currently recorded map.

The observations recorded in $\mathbf{z}_k$ can largely vary depending on the type of agent we are using. Depending on the type of sensors that the agent has access to, observations can include images, depth data or even motion information[Kabzan *et al.* 2020]. When we have access to simple data such as RGB information, we can create more observation information by using feature descriptors such as SIFT to generate descriptors. We can then use this information to easily compare different observations.

Then when reconstructing the shape representation in $\mathbf{m}$, the output can be derived from a method similar to SfM. We determine the location of key points in 3D space by comparing their change in position between different frames. In the case of SLAM, we also have the added advantage over SfM of knowing that all these frames are sequential. Through this information, we can make assumptions on which frames will have similar key points and as a result, reduce the number of frames we need to check for matches.

Visual SLAM and SfM both contribute to the reconstructions of their respective subjects but when our main understanding of their shape is derived from specific key points the reconstruction is sparse [Taketomi *et al.* 2017]. We can estimate the shape of the rest of the points in between but to gain a more accurate dense reconstruction of the subject, we can make use of RGB-D data. RGB-D data gives us a more in-depth description of the 3D structure of the object. Instead of just relying on a few key points to determine our homography transformations between frames, we have a larger array of dependable points that we can compare and triangulate more accurate transformations from as further explored in chapter 6.

2.1.3 KinectFusion

KinectFusion, introduced in Newcombe *et al.* [2011], has been a pivotal paper in the field of 3D reconstruction. The paper introduces a unique method for generating dense 3D reconstructions. Through the use of a commodity depth sensor that generates RGB-D data, the approach can build accurate representations in real time. In this paper, the depth sensor is moved throughout the scene and sequential depth information is obtained during which an overall model is built and updated.

KinectFusion works by using sequential frames from the live feed of a depth sensor. By comparing the differences between the two frames, we can determine the shift in the camera's position. We record this shift in the camera's position as a transformation. If we apply these minor transformations between frames to the camera, we should be able to accurately estimate the current position of the camera in the scene. As done in SLAM, we can use this tracking of the camera's position to determine which points are new features. If we align the views according to the camera's position, we can easily recognize information that still needs to be included in the overall representation.

As opposed to just layering each depth map on top of each other, after alignment the depth maps are changed to a different form of representation that better suits the idea of stacking information. This representation is known as the Signed Distance Function

(SDF) of the scene and it can be likened to occupancy mapping. Occupancy mapping is a method often used in computer vision problems for robotics and is characterized by splitting up a scene into a grid [Elfes and Matthies 1987]. Each cell in the grid has a probabilistic measure of whether its space is occupied. As more information is provided over time, we perform Bayesian updates on each cell's value.

The SDF representation was introduced in Curless and Levoy [1996]. As opposed to the occupancy mapping, each cell in the volumetric grid stores the distance to the closest surface. This means that a surface is given the value zero and as a cell gets further from a surface, its value gets larger. Using an SDF gives us access to the surface in a much more efficient manner than the occupancy map as we simply need to retrieve the zero crossings of the SDF. The SDF alone can cause some anomalies due to the quality of input it is given but Zach *et al.* [2007] shows that by performing λ_1 normalization on the truncated version of the SDF along with total variation regularization we can obtain a global optimal surface reconstruction known as the TSDF (Truncated SDF).

While KinectFusion can efficiently reproduce a scene there are several restrictions. The objects that can be scanned need to be static. If the object is dynamic then KinectFusion cannot rely on the features in the background to determine what part of the object we are modelling. Since it is safer to derive the camera's position from background landmarks, we ignore smaller dynamic objects that get phased out as their corresponding SDF values change to reflect this. Another restriction is the reliance on the changes in viewed depth information which allow us to keep track of the camera's position. As a result, if the depth information does not contain interesting features that are easy to track then the record of the camera's position becomes unreliable. This becomes further relevant when the focus of our reconstruction becomes a specific object in the scene.

2.2 Object Reconstruction

When reconstructing room-sized models, there is a vast amount of information to consider. We often have access to distinct and persistent landmarks that we can use effectively. As mentioned earlier, knowing the camera's position in the scene correlates with knowing which new parts of the target need to be recorded. These approaches work when considering a dynamic camera as they can provide us with further details about the environment that we can use for tracking. In cases where the camera is static, we cannot expect to use the details of the background as we have to derive pose information from the dynamic parts of the scene.

2.2.1 Assisted Object Reconstruction

The limited information available when only focusing on the moving object in the scene may not be sufficient. Due to this, we may need additional support from supplemental sources to increase the accuracy of our estimate. The following are some methods that take advantage of further aid from external factors that are not inherently part of the reconstruction process:

Equipment-based Assistance

Using specific equipment allows us to derive this pose information that we need. When performing photogrammetry on single objects, a turntable is an often-used solution. The object is placed in the center of the turntable's platform while a static camera observes the equipment. The platform can be rotated at a predetermined speed and may come with software that automates the entire scanning process. Other than this, if we know the angle of the platform's rotation, we can easily determine which part of the object we are seeing without having to do any calculations.

If we do not have access to the platform's rotation we can still calculate this value by comparing the previous frame with the current frame. This along with knowing the parameters of the rotation speed still give us an easier problem to solve as a result of the equipment. Still, the accuracy of the estimate may be lower due to the lack of information when we remove background landmarks. Methods such as Fremont and Chellali [2004] and Gonzalez-Barbosa *et al.* [2015] rely on distinctive features of the object, like corners and edges, to track its rotation. Using the trajectories of these landmarks in 3D space we can track the rotation quite well. By analyzing the multiple images taken of the object at different angles of the platform's rotation, we can generate a highly accurate convex hull of the object using only the RGB information.

Learning-based Assistance

Identifying an object in the scene and accurately segmenting out this dynamic object is also part of the object reconstruction problem. If we had prior information on the type of objects that would be reconstructed, we could use this to augment the reconstruction process. Methods such as Runz *et al.* [2018] make use of learned labels to classify and segment out known objects in the scene. This can more accurately segment out and track the pose of an object. As a result of this, we can track the object with higher confidence in object classes that have been trained on

We can also use class labels to help us further complete parts of the object we have not seen by the end of the reconstruction. Some single-view reconstruction methods such as Mescheder *et al.* [2019] that take a deep learning approach can make a 3D reconstruction of an object based on their learned understanding of the object. Given an input image or an incomplete point cloud in the case of Wang *et al.* [2022], they can predict a total reconstruction of the object.

These methods are unfortunately dependent on their training data. They are limited to only supplementing reconstruction for labels that they have been trained on. This does not work for our goal as the objects we work with are unknown. Due to this, we cannot use a method that is reliant on a specific class label being present in our target.

2.2.2 Object Pose Estimation

To perform reconstruction, understanding the pose information of either our target or our camera is vital to the process. When we remove the element of the background with a static camera, our main goal is to simulate camera motion around the object to

accurately shape the model as we would with camera pose information. As mentioned before, we derive what parts of the target are new and need to be added to our overall shape through this information. As a result, estimating the bounding box and using this to influence the 6-DOF pose of the target can largely improve our chance of success.

There are several methods like Xiang *et al.* [2017] that can make use of learned features to estimate the 6-DOF pose of objects. The success of methods only increases when we consider access to RGB-D information such as in Wang *et al.* [2019]. Having depth information is very helpful when determining the 6-DOF pose of the object because it allows us to use shape information to improve the accuracy of the pose estimation.

While these methods help to find the pose of objects that have been previously seen, finding the 6-DOF pose of an unseen object is a harder problem. Without knowing the current shape of the object, finding the corresponding 3D bounding box is a difficult task to perform. To solve this, Wen and Bekris [2021] uses a combination of object segmentation, feature matching, graph optimization and a set of constantly updated keyframes to decipher a pose for the object in real-time.

The object is first designated with a mask in the first frame. The frames that follow can track the pose based on this initial demarcation. This view of the object is considered a keyframe and if the object moves to a point where there is a novel view shown, the method records a new keyframe that is added to a list. The method develops a structure known as a posegraph where the poses are retained at recorded keyframes and establish relationships between them. The relationship describes the transformation required to move between different nodes in the graph. Whenever a new frame is received, the frame is checked to see if it can be considered a novel view. If it is, it is added to the keyframe list and if not the closest keyframe is added to the current pose to derive a pose in comparison to this.

This paper shows that if we implement this level of object pose tracking, we should be able to replace the camera motion around the target. The object's features can then be explored by introducing interaction with robots as done in Wen and Bekris [2021]. However, interaction with programmed systems introduces another level of difficulty when interacting with unknown objects. When faced with the goal of scanning objects, the solution is to instead replace this with natural interaction by introducing in-hand scanning.

2.2.3 In Hand Scanning

In-Hand Scanning is a specific field of 3D reconstruction research that deals with objects that are handheld. In most cases, these objects are shown to a static camera. The object is reoriented and rotated to show all the different features so our reconstruction can retain all the information required. Using hands allows for natural motion and precise movements that can be customized to fit the needs of scanning different types of objects.

We propose that in-hand scanning methods can be described as the collaboration between 3 different steps, i.e. **Input**, **Segmentation and Tracking**, and lastly **Recon-**

struction. Most methods focus on the Segmentation, Tracking and Reconstruction of the problem but we propose that the problem begins when we discuss what forms of Input are available. The process of in-hand scanning is flexible due to the number of options available to us. This is further highlighted by the multitude of ways it has been performed and improved in the following papers.



Figure 2.3: Images of methods and results as shown in Rusinkiewicz *et al.* [2002]

Rusinkiewicz *et al.* [2002], as shown in figure 2.3, is one of the earliest papers that attempts in-hand scanning. A lot of the methods used in this paper become reoccurring staples in the process of handheld object reconstruction. In this paper, depth information is generated using the structured light depth-sensing method. During this time, the commercially available depth-sensing equipment we currently have access to would not have been created. As a result of this, the depth information is generated using a DLP (Digital Light Projector) that displays visible structured patterns onto the object [Hall-Holt and Rusinkiewicz 2001]. As further discussed in chapter 4, we use this information to perform triangulation which in turn generates depth information.

The segmentation of the target in this paper is assisted through the use of a controlled environment. The object is held with gloves and recorded against a simple background, both of which are black. The paper uses this to focus on the reconstruction of the target. This paper performs reconstruction by aligning different frames together using a method known as Iterative Closest Point (ICP). As further discussed in chapter 6, this method can overlap each new frame of information to incrementally build up the shape of the object.

And lastly, due to the memory constraints of the time, retaining the 3D information of thousands of points was not an option. To solve this, the paper uses applying a merging solution that quantizes the 3D points in a grid-based manner. If there are multiple 3D points in a single block of the grid, they are reduced then reduced to a single point to decrease the amount of information that needs to be stored. Over time we can build up the model and retrieve a successful reconstruction of the target.

This paper proved to be pivotal in handheld object reconstruction research. This method introduced the freedom of being able to actively rotate and show a camera the new features of an object using your hands. And while this method applied quite a few commonly utilized methods of modern in-hand scanning, there were still critical errors that had not been dealt with. By circumventing the entire process of segmentation, the method is restricted to only working in specific environments. Furthermore, while this method is able to run in real time, the scanning process has to be restarted every time there is an alignment error.

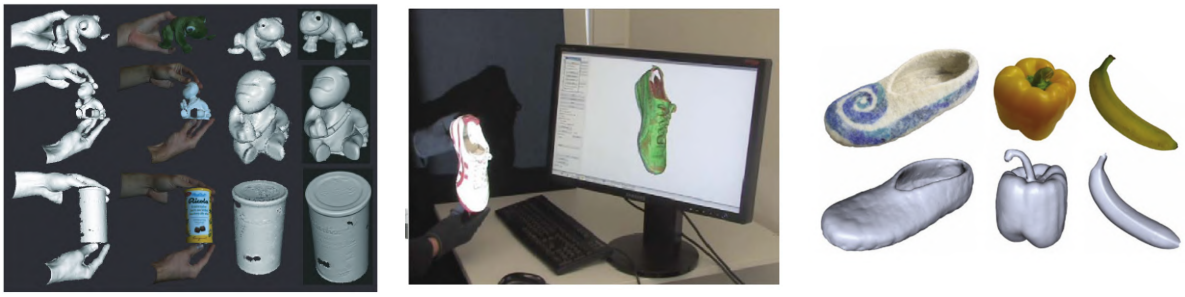


Figure 2.4: Image and results as shown in Weise *et al.* [2008] (Left) and Weise *et al.* [2011] (Middle and Right)

This method is then improved upon by Weise *et al.* [2008], as shown in figure 2.4, in several aspects. This paper introduces an improved segmentation implementation for both the background and the hands in the scene. The background is first removed through the use of the change detection algorithm described in Mester *et al.* [2001]. A frame without the hand and the object is compared to the following frames. The change detection algorithm is then used to determine which parts of the frame are new. The new information is then kept and the unchanged data which would signify elements of the background is segmented out.

These frames then use the skin colour detection approach, described in Jones and Rehg [2002], to determine which pixels are considered skin. This is done through the application of a statistical model trained on several images and while this method is efficient, it is not too robust to several conditions. The hand segmentation approach applied here does not work well with a variation in lighting conditions or skin tones. This method also introduces improved failure detection to test the success of each alignment attempt. Elements of the object's geometry and its texture are used to determine if the object has achieved successful alignment. Furthermore, in addition to using ICP here, this method also checks the texture and fine geometric features of the object for further alignment. This helps further align symmetrical objects by using the texture details on the surface of objects however, loop closure is still an issue.

As the structure shape gets built over time, the tracking error builds up. As this tracking error builds up and we reach parts of the structure that we have already scanned, the familiar points are instead treated as new points. This causes the model to build in on itself over time and as a result, the output has artefacts known as loop closure errors.

Weise *et al.* [2011], as shown in figure 2.4, introduces a method that introduces online loop closure for handheld object scanning. Due to the focus on being able to perform reconstruction in real-time, this paper reduces some of its segmentation capabilities. While this paper reintroduces the restrictions of the controlled environment in Rusinkiewicz *et al.* [2002], the advantages of being able to interact and reorient the object according to the current model are far greater. To perform this reconstruction in real-time, we need to be able to constantly look for loop closure cases while also actively deforming the model to fit the correct shape.

This is done by first adopting a surface element (surfel) representation for the storage of 3D information as described in Pfister *et al.* [2000]. Surfels are small flat disks that are used to describe the surface of an object. Each surface retains information regarding its position in 3D space, its normal vector and its radius. In this paper, each surfel also records its visibility confidence. This describes how reliable the surfel is in its information and this reliability increases as the same information is viewed and confirmed from multiple perspectives. With every new frame added, the current model's surfels are updated. New surfels are added, existing surfel's radii are updated and surfels are removed if their confidence is low enough.

Furthermore, each surfel also retains a list of other surfels it is often observed with. This creates a graph that represents the different connections between surfels and allows us to introduce solutions for when loop closure cases are detected. Once loop failure has been recorded using the loop failure detection method described in Angeli *et al.* [2008], we can apply a deformation to the model that makes the loop closure edges fit together. Since we can treat the connected surfels as a graph of nodes, we can apply a transformation that bends the appropriate surfels towards a detected loop closure edge. Through these adaptions, this method performs reconstruction in real-time.

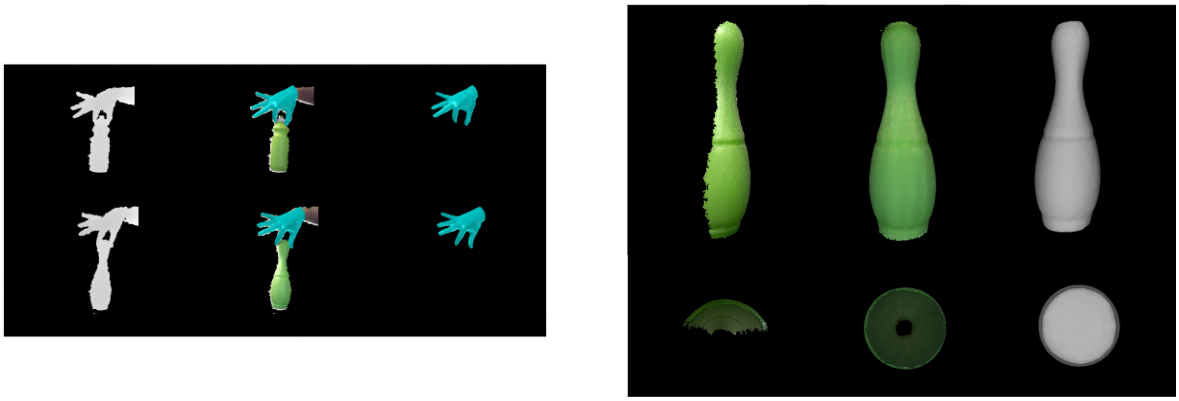


Figure 2.5: Image and results as shown in Tzionas and Gall [2015]

Another implementation that makes use of the inherent elements of handheld object reconstruction is Tzionas and Gall [2015], as shown in figure 2.5. In this paper, the hands in the scene are reintroduced as valuable sources of information. This method introduces the problem of reconstructing featureless symmetrical objects. While previous methods rely on information found on the target's surface, these approaches do not work if we interact with a class of objects that has a severe lack of this information. To solve this, this paper proposes we record the motion of the hands in the scene.

This paper shows how tracking the motion of the hands can introduce new features to keep track of. This means the alignment of the object's frames is affected by both visual elements as well as the movement of the hands in the scene. The tracking of the hands is done through the use of the method described in Tzionas *et al.* [2014]. This paper tracks the motion of hands through the use of a generative model that reacts to collision detection. This tracking of the hands is then in turn utilized to calculate contact points with the object. The movement of these contact points in the scene can

then be used to determine the motion of the object and works well in cases where there are no trackable features on our object.

This method is a good example of the variation we can expect in different parts of the pipeline. While the first two papers restrict their information to the reconstruction section by removing hands from the scene, Tzionas and Gall [2015] utilizes this information extensively. Variation like this often introduces new views on the field and can heavily change the direction of research. While this method is restricted by its initial grasp and is reliant on the tracking of the hands working, exploiting the information provided by the hands in the scene has become a staple of recent in-hand scanning methods.

2.3 Related work

There have been several recent additions to the field of handheld object reconstruction that provide valuable insight into the progress the field has made as a whole. As mentioned earlier, these variations actively highlight the multitude of different approaches that can be tested in this field.

There has been an increase in papers that actively perform hand-tracking as an equal task on par with object tracking. As earlier mentioned, the relation between the hand and the object in the scene has been further utilized to gain a further understanding of both in the process. Papers such as Zhang *et al.* [2019] and Zhang *et al.* [2021] have followed the task of handheld object reconstruction to create a subset that seeks to reconstruct hand-object instructions effectively.

InteractFusion in Zhang *et al.* [2019] uses two synchronized depth cameras on opposite sides of the scene to capture valuable information from scenes. The two depth frames from this information are first used to perform segmentation of the hand and the object in the scene. The segmented hand data is then run through a trained LSTM (Long Short-Term Memory) network to predict the corresponding hand pose. An LSTM network is quite effective at learning information that has a temporal aspect and as a result is well suited for hand pose estimation [Graves and Graves 2012]. After predicting the motion of the hands and the current segmented object in the scene, the object is merged into the current object's reconstruction.

Another amazing feature of this research is its ability to interact with deformable objects. Working with a handheld object already is difficult and then to further work with deformable adds on a layer of complexity. This complex task is addressed in DynamicFusion from Newcombe *et al.* [2015], where the objects recorded are flexible and have moving parts. DynamicFusion deals with the deformation of an object by forcing the main object into a rigid state initially. At the very beginning of the scene, an initial model is captured and stored as the canonical model.

During the following frames, the main task of the method is to estimate how the canonical model has been warped. This information is stored separately and if applied to the canonical model, it should give the current state of the structure. When the model warps to a region that has not been recorded, comparisons of the canonical model and

the currently recorded deformations can be used to recognize that this is newly introduced information. As more of the structure is shown, more information is added to the canonical model. This means the volumetric warp field can be more reliably tracked. During the time the scene is rigid it can simply be dealt with using similar principles to KinectFusion.

Through the application of these methods, InteractFusion is able to generate a model for the segmented handheld object. The tracking of its deformations is done during the tracking of the object and hand pose. The hand's placement also contributes as a filter to determine when the depth of the object has been incorrectly recorded.

In Zhang *et al.* [2021], the authors present have a method that reconstructs hand-object interactions through the use of a single depth camera. To do this, this method first performs the task of predicting 3D key points for the hand alongside a segmented object mask for a frame. This information is then used to estimate the hand pose and the motion of the object as done in Zhang *et al.* [2019]. Again similarly to [Zhang *et al.* 2019] and Newcombe *et al.* [2015], the information of the segmented object is recorded to reconstruct the object's model over time while also recording any deformations.

Huang *et al.* [2022] is another recent paper that produces state-of-the-art results in handheld object reconstruction. Through the use of a static RGB camera, this method aims to reconstruct the shape of the object. The first step of this process includes the tracking of the hand pose in the scene, followed by a segmentation of both the object and the hand from the background. Next, the object's geometry is recreated by simulating multiple views through the interaction of the hand with the object.

This method implements the neural surface reconstruction method from Wang *et al.* [2021] to recover shape information of the object and the hand in the scene. This is comprised of a network architecture trained on a large number of images to output an SDF field for a given input image. Through this, a model is generated for an image without the use of depth information. Lastly, by using 3D semantic information the generated model is split into a model for the hand and the object. While this method is limited in its ability to change grasps during the tracking of the video as this would entail a large degree of deformation, this method produces very clearly accurate results and pushes forward the state-of-the-art in handheld object reconstruction

Another interesting approach to the handheld object reconstruction problem is shown in Ye *et al.* [2022]. This method aims to recreate the shape of an object from a single frame of RGB data. This is done through another element of information that can be derived from hand-object interactions, the grasp information. This method proposes that the way a human hand holds an object has important implications for the shape of the held object. This interesting approach to the reconstruction of the handheld object first entails estimation of the hand's pose in the scene.

This information is then used to generate the possible shape of the held object as an SDF field using a network with articulation-aware encodings. This is done by first introducing a visual encoder to the image that recognizes the features of the object. This information is then supplemented by the articulation information retrieved from hand pose tracking. All this information together is then decoded to produce an SDF

field that represents the estimated shape of the object. While it relies on the training data available to perform this task, this is a very interesting approach to the problem of handheld reconstruction.

We can clearly see our pipeline's description of the handheld object reconstruction problem in all of these papers. They can all be split according to our three proposed sections. The input changes range from two depth cameras to RGB video frames to simply a single frame of RGB information. The varies between tracking the object, the hand or even both. While the reconstruction method changes between using neural representations of SDF to TSDF maps with deformation fields. There are multiple approaches that each of these methods takes but they can all be broken down according to our pipeline for the reconstruction of handheld objects.

Chapter 3

General In-Hand Reconstruction Pipeline

While the main focus of research is often at the reconstruction phase of the pipeline, viewing the process as a set of independent stages allows us to expand the prospective improvements considered. Just as done in Tzionas and Gall [2015], where it is shown that handheld objects can be more effectively tracked using the location of hand annotations. In order to further examine the possible approaches that can be considered for each step in the process, we have broken down our pipeline into 3 main phases. A combination of input, tracking and segmentation, and reconstruction gives us the general workflow needed to recreate the shape of a handheld object.

The definition of this pipeline further helps to define that there are multiple approaches that can be tested for each section. Due to the nature of working in an uncontrolled environment, we propose that we should initially work with broad categorical goals for each phase. This allows us to switch out the approach currently being tested while still keeping track of what our current goals for the pipeline are.

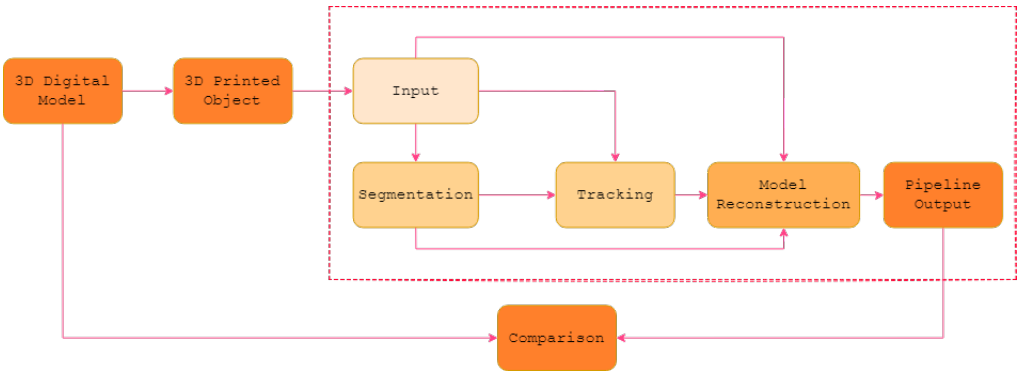


Figure 3.1: Diagram showcasing our suggested pipeline. Diagram colours are used to group according to phases (i.e. Input, tracking and segmentation, reconstruction and evaluation).

Our implemented methodology can be described through figure 3.1. In this diagram, we also display a phase for the evaluation of our experimentation. For the evaluation of our pipeline we aim to make use of 3D modelling to create an object to be scanned in with our pipeline. We then compare the distance between our scanned object and the original model used to create the object. By using this overall distance from the original model, we can define how successful our pipeline is.

Through the several years of research done in the field of computer vision, there are several permutations of different approaches for each stage that can be aligned to produce a specific result. By changing a specific attribute such as our method of tracking, we can make our process more robust to the absence of object features. This chapter details the different tested permutations of the pipeline and a brief description of the methods used in each section.

3.1 Input

The input to our pipeline heavily influences the quality of our end result. The features and details that are initially captured and retained during this phase can determine the approaches that are available for our use later down the pipeline. At this stage, we often need to make use of colour and depth data. This is often packaged as RGB information from cameras or as RGBD data from our depth sensors. These sensors introduce our first bottleneck into the system.

Depending on the resolution of our sensors and in turn the quality of the information we receive, we can estimate how successful our pipeline will be. A high-resolution, high-frame rate sensor that uses a depth-sensing method with high accuracy can entail an easier solution a few steps down the line. And naturally, as the quality of our input decreases, we need to apply separate methods such as interpolation or even super-resolution that allow us to reduce these predicted disadvantages to a minimum.

Our experimentation makes use of the Kinect V1 depth sensor. Chapter 4 provides further information on the details of this sensor as well as its role in our experimentation. While this remains our main method of input, we also perform further testing of different methods of input in the case of missing depth information. Through the use of depth estimation techniques, we aim to derive the depth of each pixel in an RGB image. This information can then be used as our input to the next part of the pipeline. The details of our depth estimation experimentation are further discussed in chapter 7.

3.2 Tracking and Segmentation

This phase has a large degree of variation that can be implemented. For our particular case of in-hand reconstruction, there are several factors to take into consideration. The object we are tracking is often unknown, there will be a degree of occlusion as the object is handheld and the object will be dynamic in the scene. We also need to be able to decipher the pose of the object in the current frame to a certain extent as this

information informs us about what part of our object we look at. These conditions shape the solutions we have to make use of in this phase.

There are several different paths that we could take to reach the same solution. If we have a very controlled environment where our object is always the main focus of our input, we can use methods such as Lee *et al.* [2022] or Kim *et al.* [2022] for salient object detection to segment out our object of interest. Another simpler method would be to make use of our depth information to segment out the closest object to the camera. However, these solutions are not robust to cases of input where the rest of the scene is populated or where multiple objects in the scene are closer to the camera.

To solve these issues, we can make use of other reliable information from our input. We know that since our object is handheld, it is in close proximity to our hands. Since we have this common factor throughout all our data, we can use this to determine what our unknown object is and where it is in the scene. Using this approach to recognize our object works well and allows us to track our objects well. In this case, however, we can still identify more issues that the system might not be robust to, such as multiple objects in the scene that resemble hands.

There is a multitude of different methods that could be applied in this section depending on how lax we are with the conditions of our input. Our main concern is getting the subject of our reconstruction segmented out from the rest of the scene. Since our camera is static in these scenarios, our background features do not provide data that will be too useful during the reconstruction phase. As a result, we have to generate this information from the dynamic objects in our scene. To do this, we have to select the approach that is most effective for our circumstances.

As described in chapter 5, we first test the success of learning-based approaches for segmentation. For our purposes, we theorized that we could train a neural network to segment out hands in our scene. Once we segment out the hands in the scene, it should be simple to further segment out our object of interest by performing depth thresholding as the object should be close to the hands. Furthermore, we also test the success of using hand pose information to segment out our object. By using the positions of each node in the hand pose of the scene, we can once again perform depth thresholding to segment out our object. The theorized solutions are tested in further detail in chapter 5.

3.3 Reconstruction

The reconstruction phase of the pipeline ties together all the information we have accumulated so far. We need to make use of our annotations from the previous steps to iteratively build up our understanding of the shape of the object. As with the other stages, there are multiple approaches that can be applied. The chosen method is often dictated by several factors that stem from the previous stages. The quality of our input, the physical properties of the object and the amount of information gleaned from the type of object tracking all contribute to our decision.

One of the base approaches, as used in Newcombe *et al.* [2011], is the TSDF map. As we track our object's pose and view new parts of the object, we add new points to the TSDF map and integrate new points into our overall shape. This approach is naive but safe. It often works well when there are a lot of features to keep track of as our estimate of the pose is often correct. Unfortunately, this does not cater to the case when loop closure error occurs.

To solve this, we implemented and tested the posegraph approach of Grisetti *et al.* [2010]. We build our model over time while also checking to see if we recognize any relation to previous frames. If we do, we change the shape of the graph. We do this by optimizing the connections between our different nodes to make sure the connected segments match each other. We can either take the online approach or the offline approach depending on the type of data we have available. Using an offline approach gives us more accuracy to our initial model while the online approach allows us to create our model in real-time.

As further described in chapter 6, we implement an offline approach to perform the reconstruction of our object. Through this approach, we also test the effectiveness of a series of different point cloud comparison methods. We apply several variations of the well-known Iterative Closest Point (ICP) algorithm to determine the most successful version for our needs. Once we have created a successful posegraph and optimized the shape of the graph, we generate a model through the application of the TSDF map.

3.4 Evaluation

During the experimentation of each of our previous phases, it is possible to estimate the success of the method through visual cues. We can display how well our segmentation map works by viewing the segmented sections. We can also determine if our reconstruction is successful by applying the pipeline to any object and viewing the resultant point cloud. As a result of this, we had a variety of different objects that we tested our approach on. But in order to further record the degree of our reconstruction's success, we need to have an initial model to compare our final point cloud to.

As shown in figure 3.1, our evaluation of the pipeline begins with the creation of a 3D model. By printing 3D models of different objects, we can gain access to both the physical object as well as retain data of the original 3D model for comparison. For our pipeline's evaluation, we selected 3 different models with varying degrees of surface details and repeating features as shown in figure 3.2. We select these models to test the effectiveness of our pipeline on different surface textures and shapes. Between these 3 models, we ensure the inclusion of smooth surfaces, rough pointed textures as well as repeating patterns.

The Batman model has a relatively smooth surface occasionally interrupted by cloth-like striations. The Dino model has a rough texture with an abundance of unique features while also allowing us to test how effectively our depth camera can pick up thinner features such as the horns. Lastly, the Owl model provides us with a repeating



Figure 3.2: 3D printed models used for the evaluation of our pipeline. Referenced in text as Batman[Presl 2022], Dino[Orest 2018] and Owl[Patel 2020]

patterned surface. The feathers provide our main source of surface detail and as a result allow us test how to effectively deal with these cases.

Our evaluation method seeks to scan the 3D printed object and compare the shape of this scan to the original 3D model used for printing. By comparing the distances between our scanned result and the original object, we can determine our pipeline's degree of success.

The following chapters will describe our experimentation in further detail. In these chapters, each of the steps taken will be broken down to explain the process of testing and achieving a successful reconstruction for our situation.

Chapter 4

Input

4.1 Introduction

This chapter focuses on our method of input. We will go into detail about the different features of the sensor we used and how it affects the rest of the pipeline. Our main goal during experimentation for this section was to first retrieve ground-truth depth information. Having reliable depth information is vital to both the segmentation of our object as well the reconstruction of surface details. The following sections detail our choice of sensor and the manner in which it reliably derives depth data. We also detail a few advantages and disadvantages of our chosen medium. The importance of ground-truth depth data for our pipeline is also further shown by the discussion of using depth estimation data in chapter 7.

Table 4.1: A list of hardware specifications for the Kinect V1

Colour Resolution/Rate	1280x960@12Hz or 640x480 @ 30Hz
Infrared Resolution/Rate	640x480 @ 30Hz
Depth Resolution/Rate	320x240 @ 30Hz
Effective Range	0.4 m - 3.0 m
Field of View(Horizontal)	58°

4.2 Equipment

During our experimentation, we made use of the **Kinect V1**. This commodity depth sensor is equipped with both an Infrared(IR) Camera and an RGB Camera that it uses to provide visual information regarding the scene, as shown in figure 4.1. The sensor also has an IR Projector that works in tandem with the IR Camera to generate depth information. As depicted in table 4.1, the resolution and the frame rate of the camera are mediocre. However, there are a few conveniences on the software side that help with the deployment of the system. The available software automatically caters to the

physical difference in the placement of the RGB and IR cameras. Any issues that might arise as a result of a difference in focal length are also resolved when we gain access to the depth and colour information from the component.

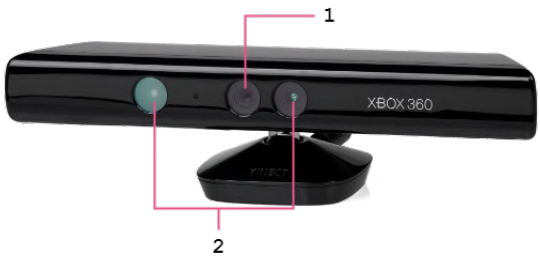


Figure 4.1: Diagram displaying the Kinect V1. Label 1 is the RGB Camera. The components in label 2 work together to become the depth-sensing component. The component on the left is the IR Projector and the component on the right is the IR Camera. Original image by Amos [2011].

4.2.1 Depth-Sensing

If we have two parallel cameras aimed at a similar scene and a known corresponding point that is projected on both cameras, we can use stereo triangulation [Hartley and Zisserman 2003]. When a point in world space $\mathbf{p}$ is projected onto our two cameras as the points $\mathbf{p}_l$ and $\mathbf{p}_r$, we can use several properties from this setup to determine the distance $\mathbf{d}$. As shown in figure 4.2, $\mathbf{d}$ describes the distance from $\mathbf{p}$ to the perpendicular line connecting the optical centers of our two cameras. This distance $\mathbf{d}$ also includes the focal length $\mathbf{f}_l$ of our cameras while the distance between the optical centers of our cameras is denoted by $\mathbf{b}$.

To begin triangulation, we create a triangle between the point $\mathbf{p}$, the optical center of camera 1 and the optical center of the camera and another triangle between the point $\mathbf{p}$ and the projected points $\mathbf{p}_l$ and $\mathbf{p}_r$. We then describe a relationship between these similar triangles as shown in equation 4.1. This relationship can then be simplified to calculate our distance $\mathbf{d}$ as shown in equation 4.2.

$$\frac{\mathbf{b}}{\mathbf{d}} = \frac{\mathbf{b} + \mathbf{x}_l - \mathbf{x}_r}{\mathbf{d} - \mathbf{fl}} \tag{4.1}$$

$$\mathbf{d} = \frac{\mathbf{fl} \bullet \mathbf{b}}{\mathbf{x}_r - \mathbf{x}_l} \tag{4.2}$$

Through these calculations, we can determine the relative distance $\mathbf{d}$ for multiple points in world space. The difficult part of stereo depth estimation even with the presence of these equations is finding the same point when it is projected in two different views. We can determine sparse correspondences from labelling features, but finding the exact point in another view for a relatively featureless point is much more difficult.

To counter this, the Kinect V1 generates depth information using a method known as structured light depth-sensing as described in Andrews [2011]. This depth-sensing

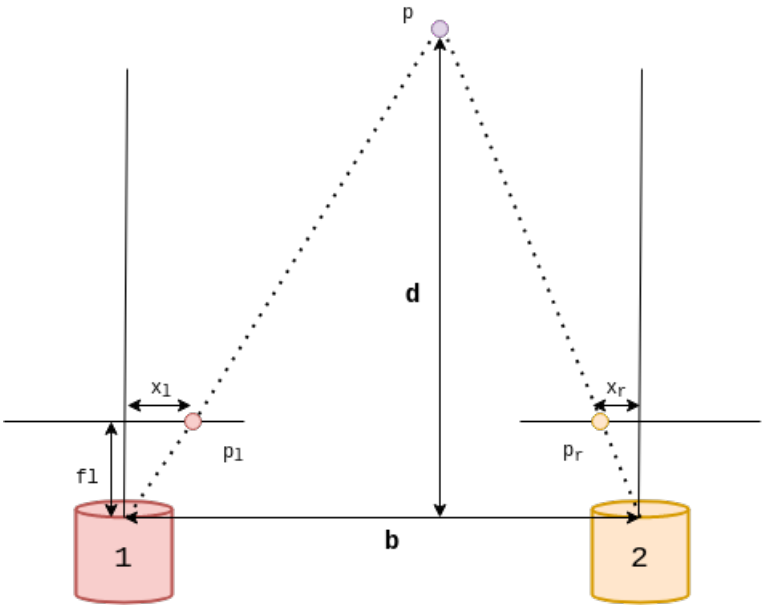


Figure 4.2: Example of stereo-triangulation performed between parallel stereo cameras.

method is inherent to the first generation of Kinect depth cameras while further generations started using the Time-of-Flight method to determine depth. This method when compared to the method described Hall-Holt and Rusinkiewicz [2001] has the added advantage of using a type of light that isn't visible and as a result doesn't affect our colour information.

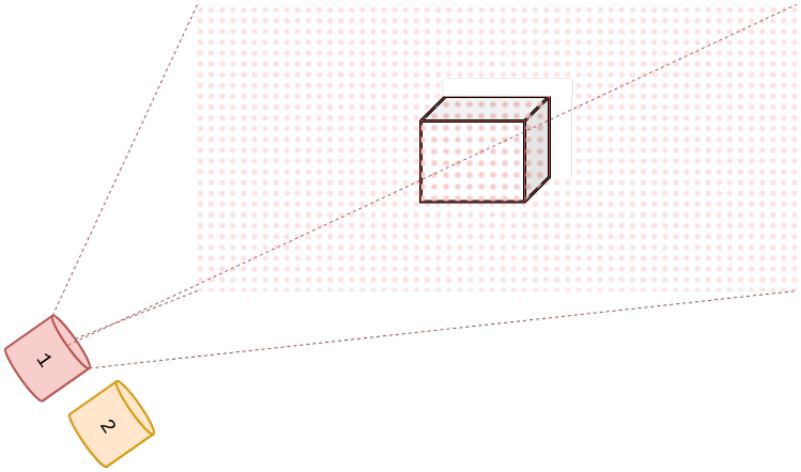


Figure 4.3: Diagram displaying the structured light method that the Kinect V1 implements. Component with label 1 is the IR Projector while the component at label 2 is the IR Camera.

As shown in figure 4.3, the IR Projector displays a speckle pattern on the scene the component is pointed at. This speckle pattern has regions of specifically arranged dots that indicate where on the pattern they lie. This means we inherently have the values for $\mathbf{x}_l$ and $\mathbf{x}_r$ for a large number of points in world space. The intrinsic properties such

as the focal length **fl** of our camera and the distance between the IR Projector and the IR Camera **b** are also inherently known. As a result, we can easily and accurately determine the distance of a multitude of points from our camera, resulting in depth data as shown in figure 4.4.

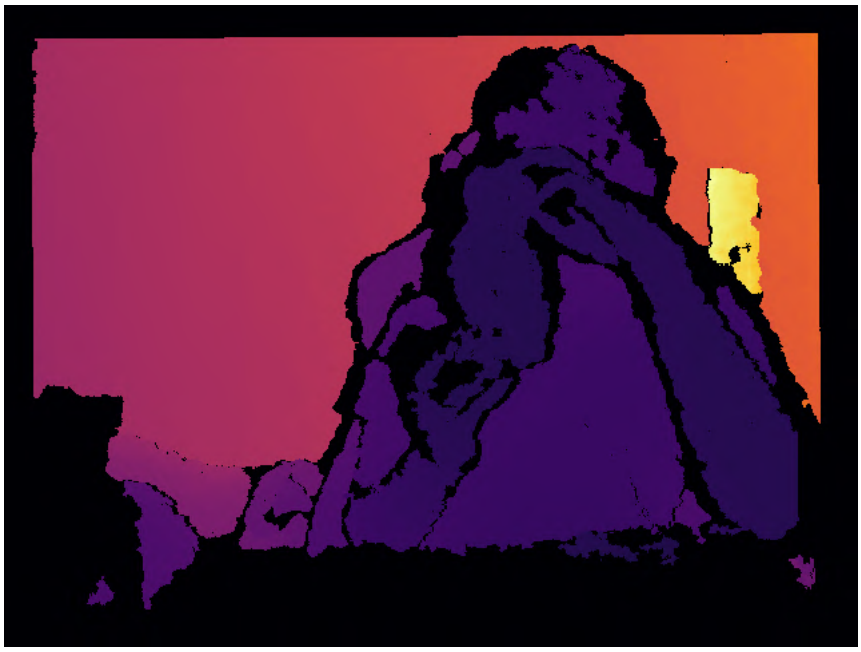


Figure 4.4: Example of colorized output depth retrieved from Kinect V1.

4.2.2 Advantages

The optimal environment for this Kinect Camera to be used is indoors while the best results can be found between distances of 0.8m to 1.2m. Using the Structured Light method in its optimal environment provides us with high-accuracy depth information. This high accuracy depth allows us to track object features quite well.

Displaying a known pattern allows us to bypass one of the biggest factors that cause issues in stereo-based depth estimation. Instead of having to find correspondences between two images, we can immediately recognize the placement of a point on the pattern projected.

4.2.3 Disadvantages

While the depth accuracy works great in the specified optimal environment, the system becomes near unusable in unfavoured situations. We cannot get the depth information of objects closer than 0.8m as the IR projector and IR camera are not able to collaborate at this distance. And while we are still able to get information up to 3.5m, this information is not considered very accurate. There is also a slight decrease in accuracy when we consider distances between 1.2 - 2.0m from the component.

As a result of the system's reliance on being able to detect the pattern projected, there are certain cases where this system becomes unusable. The system cannot be displayed outdoors as the ambient sunlight overpowers the pattern displayed by the projector. The Kinect camera can also not be used in tandem with another Kinect camera overseeing the same scene as the IR projectors will have their patterns overlap. In both of these cases, the IR Camera will not be able to effectively read the pattern displayed and as a result, the 3D shape we retrieve will be filled with artefacts.

Since we rely on a single projector to display the pattern on the scene, there is a chance that the pattern can be displayed onto an unwanted occlusion. These occlusions cover up where the pattern needs to fall and can result in several holes forming in our depth information. The system cannot calculate the depth at these locations and so leave these areas unknown in the output. The numerical value stored as a placeholder for these holes is zero.

The system is also prone to motion artefacts if the subject in the scene moves at pivotal points during the process of depth-sensing. This is because the system projects a series of multiple patterns before it triangulates a single frame of depth information to increase our accuracy. If there is movement between these sequences, the resulting depth information can contain these motion artefacts.

A large amount of these disadvantages do not affect our experimentation. Our reconstruction of the object is done in the optimal environment for the depth sensor. We also only require a single depth sensor for this task and as a result, do not need to be concerned about cross-sensor contamination. The occlusions that occur could lead to an issue but only if we were using single images. As we use multiple frames of data, we can rotate and articulate the object so we can retrieve all the surface details we need. Lastly, we can avoid motion artefacts by making sure we move the object at a certain speed. As shown in table 4.1, the refresh rate of the Kinect V1 is shown to be 30Hz. This should mean we can still move our object at a relatively normal pace without sacrificing accuracy.

Chapter 5

Segmentation and Tracking

5.1 Introduction

In this chapter, the main goal of our method is to segment out the subject of our reconstruction. Taking into consideration the conditions under which our data was recorded, we could not make use of salient object detection. During our process of experimentation to generate a method that effectively segments out our handheld object, we attempted a few different approaches that would cater to these conditions. With the possible inclusion of multiple other objects in our scene, our main line of thinking was to exploit the relationship between the hand and handheld objects. As mentioned previously, depth information allows us to segment out objects that are in close proximity. This means as long as we know where our hands are in the scene, we know where our target object is by association.

To achieve this degree of segmentation, we experimented with two different approaches. Our first approach was to make use of a learning-based method on our known information regarding the presence of hands. To capitalise on this data, we implemented a neural network with the U-NET architecture [Ronneberger *et al.* 2015] that would find the location of the hands in the scene. Our second set of experiments made use of hand pose estimation. Through these approaches we wanted to obtain the location of the hands in the scene. This information would then be used to determine where in the scene the object could be found.

5.2 Segmentation through Learning

The U-NET architecture is well known for its success in the field of segmentation and as a result, we attempted to apply it to our specific scenario. We trained the network on images with hand annotations to predict how likely it was for each pixel in each image to belong to a hand.

The U-NET architecture is often acknowledged for its ability to learn at an incredible rate even with minuscule amounts of data available. As described in Du *et al.* [2020]

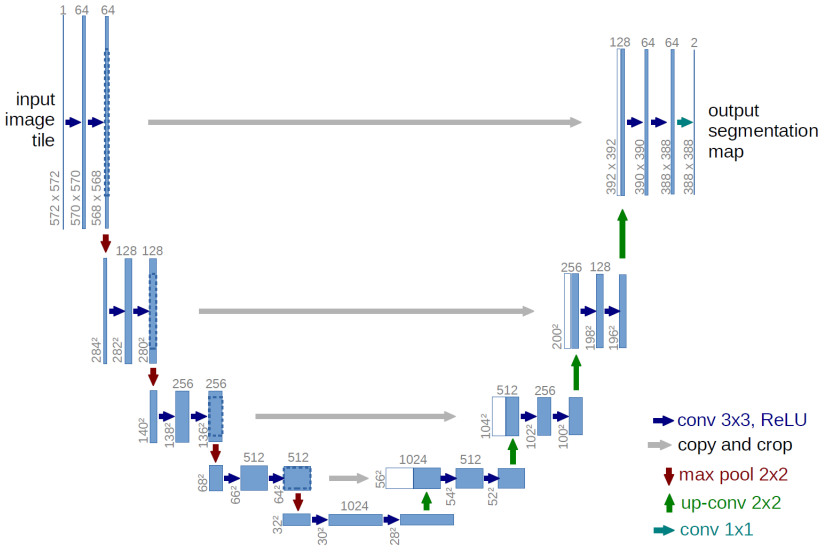


Figure 5.1: Diagram showcasing the U-NET architecture as shown in Ronneberger *et al.* [2015].

and shown in figure 5.1, the structure of the network paired with data augmentation lends itself to adapt robustness even in the presence of a lack of data. Through the use of the encoder section of the architecture, the network is able to learn the important features of the input. The decoder section then interprets these features to generate a segmentation mask for our output. The skip connections that flow between the encoder and decoder allow us to make further use of earlier learned gradients without the worry of losing this information during the process of convolution and upsampling. The U-NET architecture also allows us to process information quickly, which would allow us to perform the segmentation in real time if necessary.

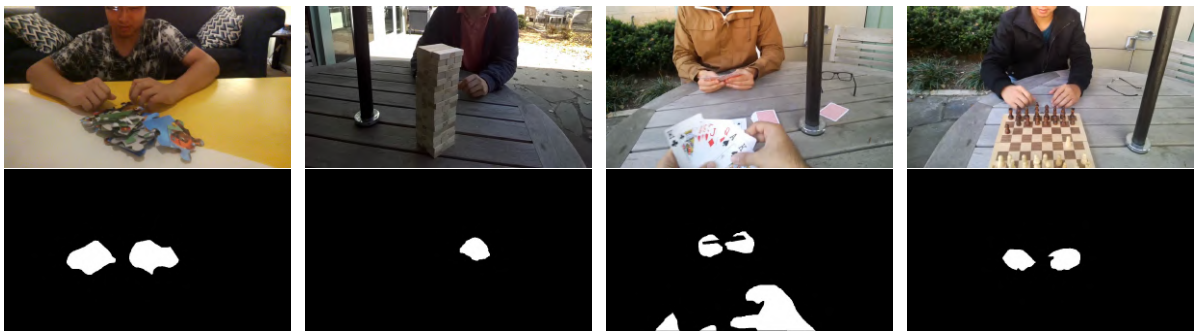


Figure 5.2: Examples of data from the Bambach *et al.* [2015] dataset. The first row includes the colour information provided in the dataset while the second row is the corresponding hand annotations to train on.

Our network was trained on a dataset that included hands interacting with objects. We knew that the hands in the input for our pipeline would be interacting with objects so we wanted to train our network to recognize hands in a way that was robust to the presence of objects. The dataset we used was from Bambach *et al.* [2015] and while the dataset only includes around 4500 images, it contains several different lighting

conditions as shown in figure 5.2. We tested iterations with the vanilla dataset as well as a series of augmentations to see their effect. The augmentations we tested included a series of random rotations and flips to introduce rotation invariance when detecting hands. We also applied a series of colour shifts and smoothing filters to further reduce the effect of noise on our output.

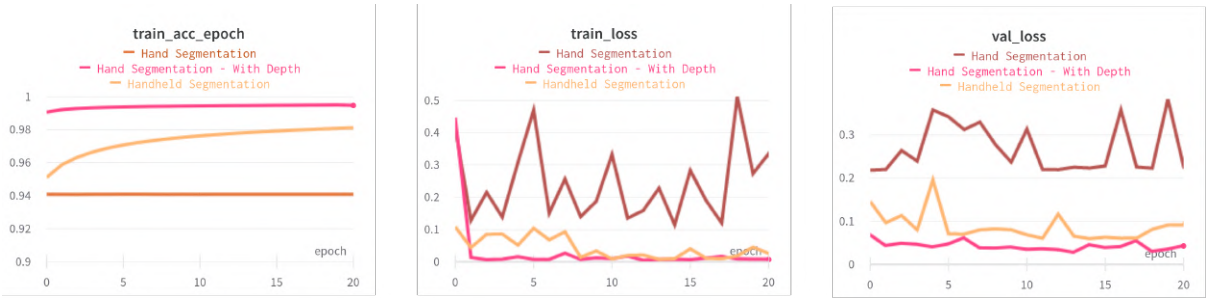


Figure 5.3: Graphs displaying the results our training process.

As shown in figure 5.3, our results for this hand segmentation network on our training sets were acceptable. The training accuracy and loss curves show us that our networks were learning features from our data. However, viewing the validation loss curve for our hand segmentation task unfortunately shows that our network was unable to act with new scenarios. This was echoed further when tested on real-world data. The results were subpar as shown in figure 5.4.

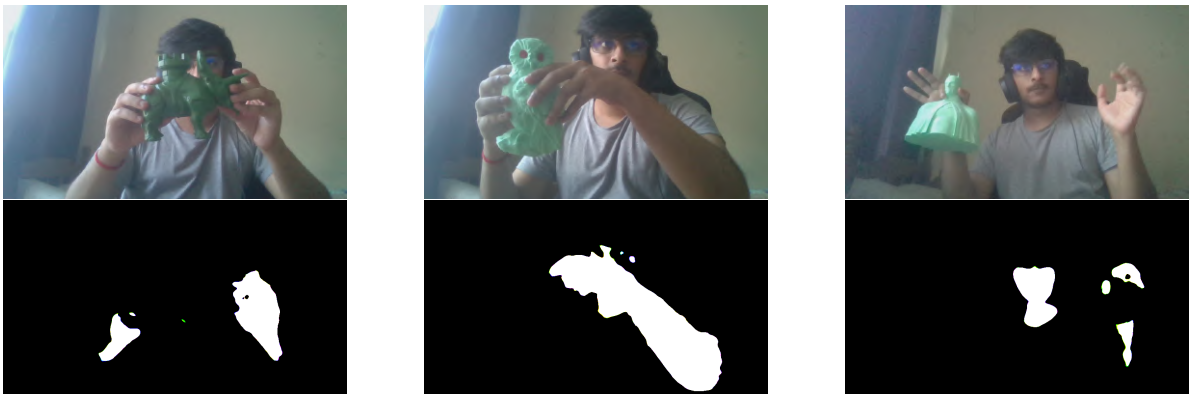


Figure 5.4: Results of our network performing Hand Segmentation using RGB data on real-world data.

Another experiment we performed was the inclusion of depth information for the training of the hand segmentation network. As shown in figure 5.3, the results were far better than the hand segmentation network that only relied on colour information. We supplemented the first dataset, as shown in figure 5.5, by applying the pre-trained depth-estimation network from Niklaus *et al.* [2019] for the task of introducing depth into regular photographs.

This network did not translate to success on real-world data either. We can attribute this to the added complexity that depth estimation introduces to the input. While it does provide more information, we negatively influence our result due to the disparity



Figure 5.5: Dataset augmented with depth information using the reference network from Niklaus *et al.* [2019]

in the clean depth achieved on the dataset and the messy depth achieved when we apply the pre-trained network to real-world data as shown in figure 5.6.

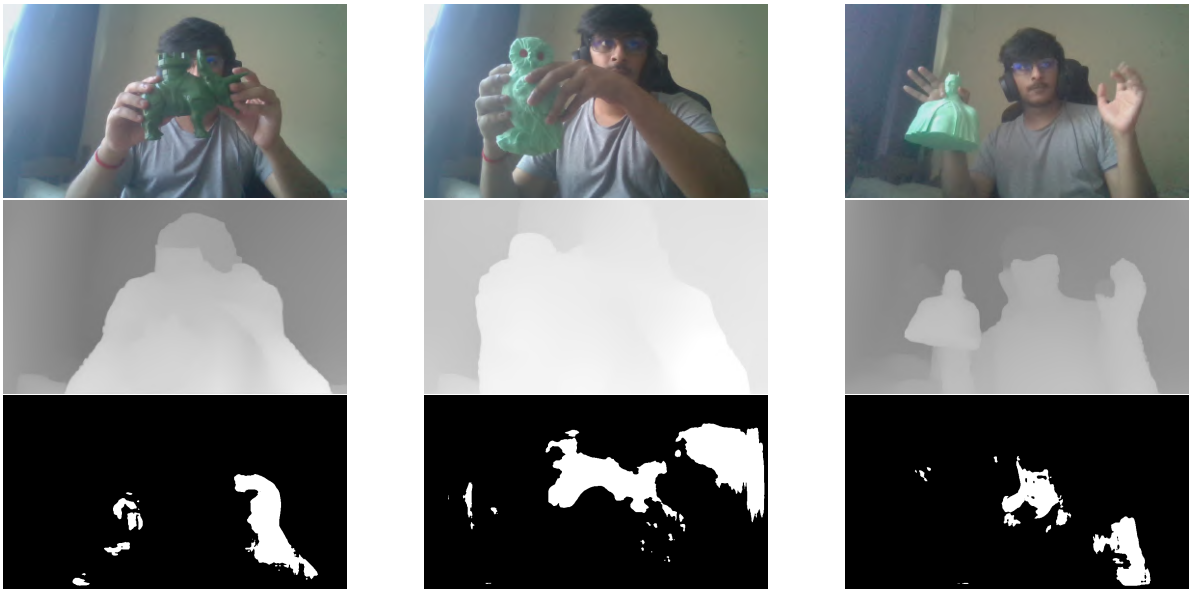


Figure 5.6: Results of our network performing Hand Segmentation using RGB-D data on real-world data. The first row is colour information while the second row is the corresponding depth information generated using the reference network from Niklaus *et al.* [2019]. The last row displays our results.

We still went forward with the idea that knowing the general location of the hands in the scene would be enough to determine the depth of the hands. The next experiment that we tested made use of the output of our first network. Our hope with this approach was to make use of a similar network but instead, output a segmentation mask for our object of interest. Our input during this network's training process included the image of a handheld object, depth information, segmentation masks for the hands in the scene and finally the mask of the object. We were hoping that the network would be able to learn the relationship between the location of the hand segmentation mask and the location of the object.

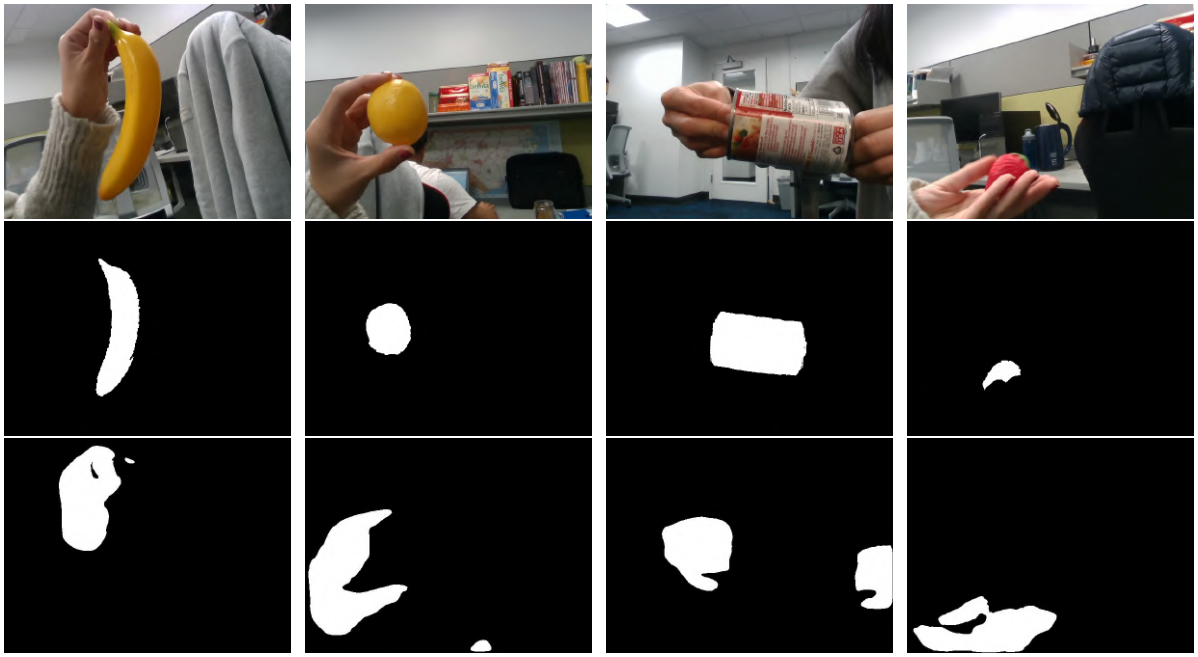


Figure 5.7: Examples of data from the Wang and Hauser [2019] dataset. The first row includes the colour information. The second row is the corresponding object annotations. The third row contains our generated annotations for the hands using our RGB hand segmentation network.

To attempt this section of research we made use of another dataset from Wang and Hauser [2019]. This dataset also included images and depth information for hands that were interacting with objects as depicted in figure 5.7. In this dataset, however, the main annotations we were supplied with were for the objects. To supplement this information we first applied our RGB hand segmentation network to the image of the handheld object. We used our RGB hand segmentation network here as we achieved higher accuracy when testing on real-world data. Once this was done, the network was then trained on this information. We also applied a similar series of augmentations as used in our first network to this data to increase robustness.

The results for this network, while better than our RGB hand segmentation network, were worse than our results for the RGB-D hand segmentation network. This can be attributed to the difference in information provided during training and testing. The depth information we trained our network on included ground-truth depth information from the dataset while our real-world testing made use of the previously noted depth estimation network. While the validation results were on par with the results of our RGB-D hand segmentation network, they still did not reliably decrease. And again, when tested on real-world data as shown in figure 5.8, our network does not achieve success.

While there was some correlation to the position of hands in our hand segmentation networks, there were also false positives that occurred when the network considered pixels that were part of the rest of the body. This meant that the positions of the hands were not dependable. As a result, the associated depth information would not reflect

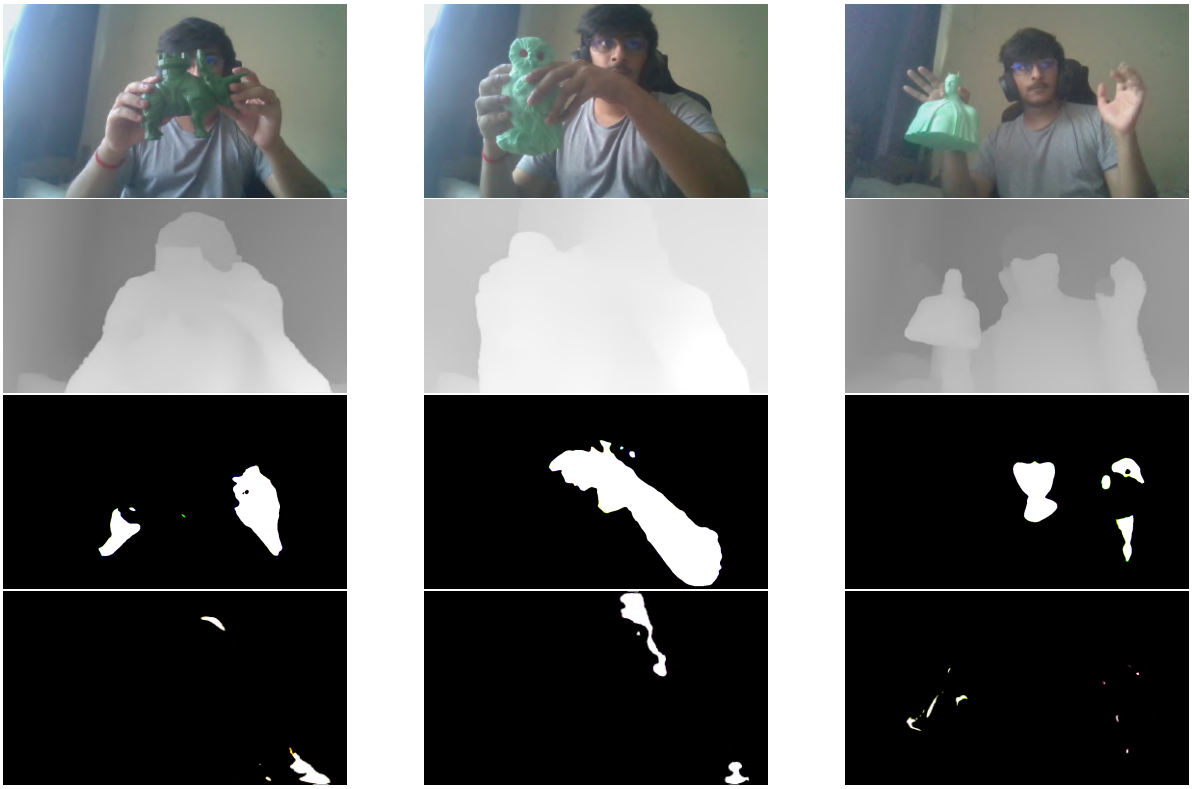


Figure 5.8: Results of our network performing Handheld Object Segmentation using RGB-D data on real-world data. The first row is colour information while the second row is the corresponding depth information generated using the reference network from Niklaus *et al.* [2019]. The third row displays our results from our RGB hand segmentation network's results on our real-world data and our final row includes the results of our final network.

the general position of the object. Furthermore, the handheld object segmentation network was trained with a reliance on the output of the hand segmentation network. Since the first network was already unreliable, the input into the second network could not provide us with the required output. The uncertainties between the two networks stacked and resulted in incorrect output from these networks.

The results for both networks were unfortunately unreliable when applied to real-world input from our camera. The validation set that we use is a subset of the datasets that we are using to train. As a result of this, even though our validation loss is low as shown in the results for our RGB-D hand segmentation and the handheld object segmentation, our networks perform poorly on real-world data. Due to the inaccuracy produced by these approaches on real-world data, we knew we would not achieve the required degree of accuracy in our segmentation.

5.3 Segmentation through Hand Pose Estimation

At this point, we moved onto our next set of experiments. As described earlier, to segment out the hand and the object in the scene, we only need to know the general

depth of the hand in the scene. We can get this information by getting the depth of several points on the hands as long as the points are reliable. The output we receive when we perform hand pose estimation consists of annotations for each important joint in the hand. Looking at the corresponding locations for each annotation on the depth map gives us an array of depth values that lie on the hand. Using these, we can derive a general region of interest that demarcates where our object lies.

5.3.1 Hand Pose Estimation

Mediapipe Hands gives us an easily-deployed solution to use for our case [Zhang *et al.* 2020]. Mediapipe Hands is a machine-learning pipeline that applies pose estimation on RGB input with the need for very few resources. This works well with the current information we receive and allows us to demarcate our region of interest in real-time. Mediapipe Hands is comprised of two networks in a pipeline that work together to provide us with pose information as shown in figure 5.9.

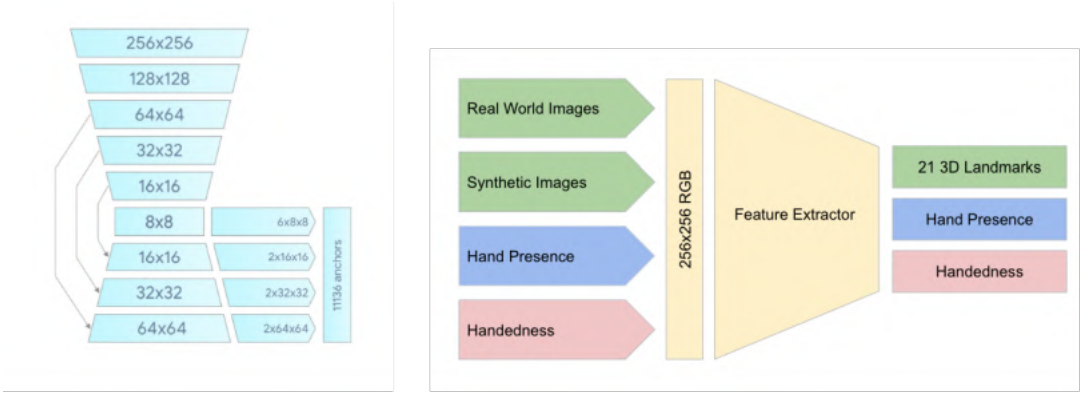


Figure 5.9: Diagram showcasing the Mediapipe Hands Network Architectures as shown in Zhang *et al.* [2020]. On the left is the architecture of the Blaze Hand Detector while on the right is the Hand Landmark Model.

The first network in this pipeline is the Blaze Palm Detector. The goal of this network is to detect the presence of palms in the input it has been given. Placing a bounding box around a hand while considering all the possible iterations of the different connected digits is a time-consuming and complex task. To simplify this task, the network is instead trained to detect and create a bounding box around the much simpler palm of the hand. The naturally simple shape of the palm also allows bounding box selection algorithms such as non-maximum suppression to be applied.

The palm detector acts as the entry point to the rest of the hand pose estimation pipeline. It is only run while there is no current palm recognized in the scene. Once the palm has been detected we move the focus to providing annotations for the rest of the hand as shown in figure 5.10. Before the image is sent to the next part of the pipeline, we crop the scene down to the bounding box placed around the palm. This cropped image is then reoriented to remove the possible effects of transformations on the tracking of the hand landmarks. If we lose track of the hand landmarks, we re-

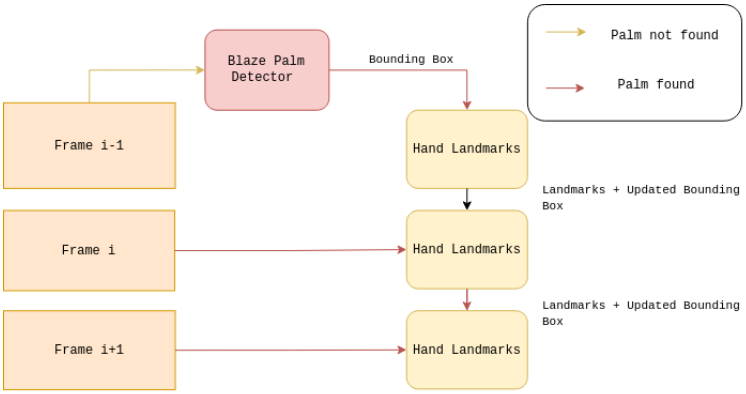


Figure 5.10: Diagram showcasing a summarized version of how the Mediapipe Hands Network processes data.

peat palm detection. Otherwise, our hand landmark model constantly updates both the hand landmarks and the bounding box generated.

This second network receives the cropped image and places 21 landmarks along the hand that together indicate the specific pose for the frame it has received. The second network is a model that has been trained on a large dataset of annotated hand images in several different scenarios with large amounts of variety. The effectiveness of the training is displayed in the model's ability to track landmarks well even in cases of occlusions and partial visibility. These landmarks not only contain 2D information about their position but also hold details regarding their relative 3D position in comparison to the landmarks on the wrist as shown in figure 5.11.



Figure 5.11: Example output from Mediapipe Hands when interacting with an object.

For our overarching reconstruction pipeline, we choose landmarks that predominantly lie on the palm of the hand to determine what our general depth is. Since the palm is larger, there is a smaller chance that the location is in one of the areas where the depth information is affected by occlusion. After we retrieve the depth values at these points,

we calculate the average depth. There is often a risk that the depth at these annotations falls into one of the holes that our ground-truth depth produces. As our depth sensor produces holes around occlusions and our hands often create these situations, we need to make sure we only consider non-zero values when calculating our depth average. We then define a range around this average value as a region of interest. If a depth value lies within this range, we consider it close to the hand. This provides us with a well-segmented mask of our hand and the object being held as shown in figure 5.12.



Figure 5.12: Example of depth thresholding image. Range selection for viable depth values is calculated from depth at hand landmarks.

5.3.2 Skin-Segmentation

The information we have at the current stage after depth thresholding is often limited to the hand and the object of interest. To further specify our segmentation to just our object of interest, we can perform one more level of segmentation. Again, as there are no known details about the features of the object itself, we can make use of the known features in our scenes. As mentioned previously with our U-NET approach, one of the issues with performing specifically hand segmentation is that the network can provide false positives for other parts of the body in the scene. The cropped input we receive from our depth thresholding already reduces the chance of this occurring but we can merge the information we have in a way where this becomes a non-issue.

To perform our skin segmentation more accurately than we can with our trained network, we make use of a pre-trained approach that has been trained on a subset of images from the COCO dataset [Lin *et al.* 2014]. This approach makes use of a Fully Convolutional Network(FCN) that uses ResNet101 as its backbone. The utilized dataset has a variety of skin colours and lighting conditions that allow a more robust result than the dataset used for our U-NET approach. The task is also simpler than hand segmentation as the network has more features to learn on the rest of the body. As a result, classifying skin as opposed to specifically the hand is a more lenient task to train for.

Through these steps, we generate the required information to perform viable object segmentation in real-time. By simply multiplying the inverse of the mask we generate from skin segmentation with the mask from depth thresholding, we can calculate an initial mask for our object in the scene as shown in figure 5.13. While this initial

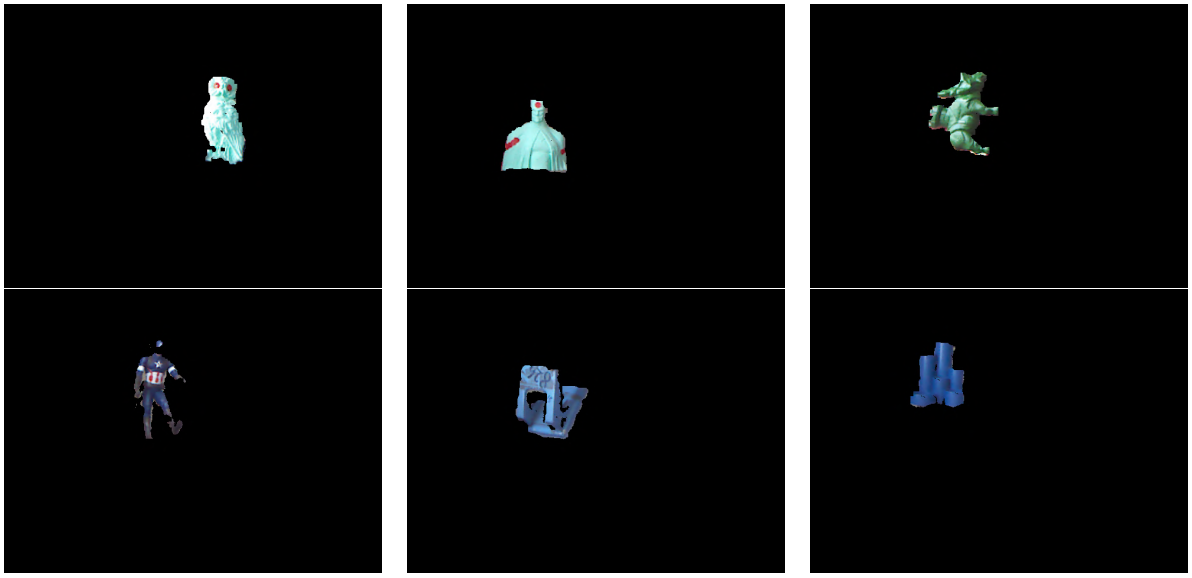


Figure 5.13: Examples of our pipeline output at current stage. Segmentation has been applied to multiple objects.

mask contains valuable data regarding the details of our target, there is still noise in our output. This next section details the steps we take to reduce this noise while also describing the optimizations we perform on this initial mask.

5.3.3 Noise Removal and Optimizations

A certain degree of noise in the accuracy of our input can cause large detrimental effects during the process of reconstruction. We implement a series of solutions to both counter this and improve the performance of our object segmentation.

Due to the nature of the interactions between the hand and the object, there are moments where the hand-tracking solution fails and as a result, we lose track of where our object is in the scene. This results in valuable frames of information being lost. To resolve this we implement a simpler feature tracking solution that utilizes optical flow in the scene [Bouguet and others 2001].

Once our initial approach recognizes the object in the scene, we start tracking features in the scene. This means during the frames where our hand-tracking solution fails, we can rely on our optical flow method to continue tracking the object. By using the corresponding depth at the locations of recorded object features, we can estimate the possible depth range that our object resides in. This approach allows us to recover the few frames that we would have lost during the failure of the hand tracking solution as shown in figure 5.14. To reduce the overall computational cost of the optical flow method, we also make sure that the only features we are considering to be tracked are the ones on the object itself. This can be done by only accepting new points to track on our segmentation output when we have access to the hands in the scene.

Applying skin segmentation to the entire image often results in a much lower frame rate due to the larger amount of information that needs to be processed. There is still

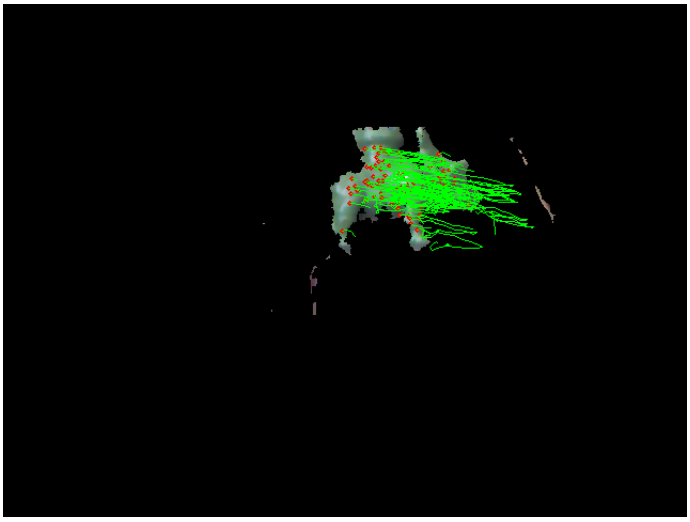


Figure 5.14: Example of optical flow tracking in action. Red points are tracked features while green lines are the trajectories of those points in the scene.

a considerable amount of overlap between data that is processed for skin segmentation and data that is never considered due to depth thresholding. By highlighting a specific area of each frame and applying skin segmentation to these specific subsets of the image, we can extensively increase the speed at which skin segmentation is performed without affecting the accuracy of our result.

Further utilizing our hand landmarks, we actively demarcate the subset of the image that needs to be processed. We draw a boundary box around the detected hand in the scene while also making some reliable assumptions about the object's placement. If we detect a right hand in the scene, we draw our boundary box to include both the hand and some extra space to the left and vice versa if a left hand is detected. If both hands are detected in the scene, our boundary box is drawn to include both hands with the assumption that the object is between them.

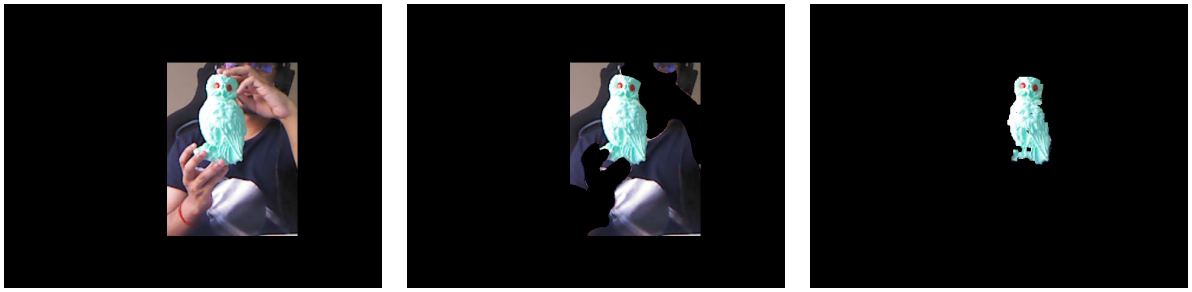


Figure 5.15: Example of our entire optimized segmentation pipeline at work. From left to right: Box crop is implemented based on hand landmark locations, skin segmentation is applied to box crop, depth thresholding mask is applied to previous result.

Once all our segmentation masks have been applied as intended, the output we generate often still has a few pixels surrounding the object that can be considered noise. These issues are easily solved through the use of morphological operations. By applying

an open operation on the merged masks, we can remove the floating pixels while still retaining the edges of our object. By applying all these optimizations to our method, we can generate an accurate object segmentation mask in real time to be used for our next part of the pipeline as shown in figure 5.15.

Chapter 6

Reconstruction

6.1 Introduction

This chapter details the final section of our pipeline. At this point, we have gathered all the information required to perform the reconstruction of the object. By using both the segmentation and depth information in the scene, we can incrementally build up our model of the object. As done before, our implemented method is chosen based on the type of information we receive from the previous parts of the pipeline. Figure 6.1 displays the current state of our input once segmentation and tracking has been applied. This RGB information also has stored corresponding depth information that has also been similarly segmented.



Figure 6.1: Example input for the reconstruction section.

6.2 Frame-to-Frame Comparisons

A large part of the reconstruction process relies on matching a series of different frames. The different information in every frame contributes to our larger understanding of the object's shape. With every new frame, we find what part of the object is being shown, what parts of the object it connects to and over time complete an accurate representation of the object. Once we have a good understanding of the shape, every subsequent frame matches to an already mapped part of our model and further solidifies our interpretation.

To build this shape of the object we need a reliable manner of performing the comparisons between different frames. There are various methods that can be applied but a common mechanism among them is finding correspondences and attempting to find the transformation that makes them overlap. If a viable transformation is found, we can assume that there are common features between the two frames. This entails that our two frames have a connection between them and should contain parts of the object that reliably overlap.

Most SLAM solutions, as described in chapter 2, used today heavily utilize the ICP algorithm. Introduced in Besl and McKay [1992], the main goal of the ICP algorithm is to align two point clouds to make them overlap as much as possible. In the case that we have known correspondences between two point clouds, the alignment has a direct and optimal solution that can be resolved simply. In the case of two hypothetical point clouds, $\mathbf{X}$ and $\mathbf{Y}$, and their paired correspondences, $\mathbf{x}_i$ and $\mathbf{y}_i$, we have to find the transformation that aligns them when applied. The result of this transformation should be a modified version of our correspondences, transformed by some rotation, $\mathbf{R}$, and translation, $\mathbf{t}$. We should aim to minimize the overall distance between the corresponding points in the two point clouds as shown in equation 6.1.

$$\min_{\mathbf{R},\mathbf{t}} \sum_{(x_i,y_i)\in I} \|\mathbf{y}_i - \mathbf{R} * \mathbf{x}_i - \mathbf{t}\|^2, \tag{6.1}$$

where $I \subset X \times Y$ is the set of points (x_i, y_i) that correspond between the point clouds.

Furthermore, if we define an intermediate point cloud, $\overline{\mathbf{X}}$, that contains the points of $\mathbf{X}$, we can calculate the transformation that needs to be applied to $\mathbf{X}$ as shown below.

Algorithm 1 To summarize, the base ICP algorithm is described below:

$\overline{\mathbf{X}} \leftarrow \mathbf{X}$
$\mathbf{E} \leftarrow \infty$
while $\mathbf{E}$ is decreasing and $\mathbf{E} >$ convergence_threshold **do**
  $(\mathbf{y}_i, \overline{\mathbf{x}}_i) \leftarrow$ `FindClosestPoints`$(\mathbf{Y}, \overline{\mathbf{X}})$
  $(\mathbf{t}, \mathbf{R}) \leftarrow$ `FindOptimalTransformation`$(\mathbf{y}_i, \overline{\mathbf{x}}_i)$
  $\overline{\mathbf{X}} \leftarrow \mathbf{R} * \overline{\mathbf{X}} + \mathbf{t}$
  $\mathbf{E} \leftarrow \sum \left\|\mathbf{Y} - \overline{\mathbf{X}}\right\|^2$
end while

There are a series of variants to the ICP method that can be applied to increase the success of our algorithm. These variants contribute to the accuracy and take advantage of the features of the object. The following listed methods are the variants we tested to improve the success of our application. The base ICP method is successful due to the abundance of points and information available when reconstructing surfaces that are the scale of a room. The use of these variants is necessary when attempting to reach convergence under our circumstances.

6.2.1 Multi-Resolution ICP

There are several simple updates we can make to our Vanilla-ICP method to increase the speed at which we reach convergence. Downsampling the point clouds reduces the time it takes to perform ICP but it comes at the cost of losing accuracy in our final alignment. By applying Multi-Resolution ICP we can gain the advantages of downsampling while still retaining the precision we would have by aligning our initial dense point clouds Jost and Hugli [2003].

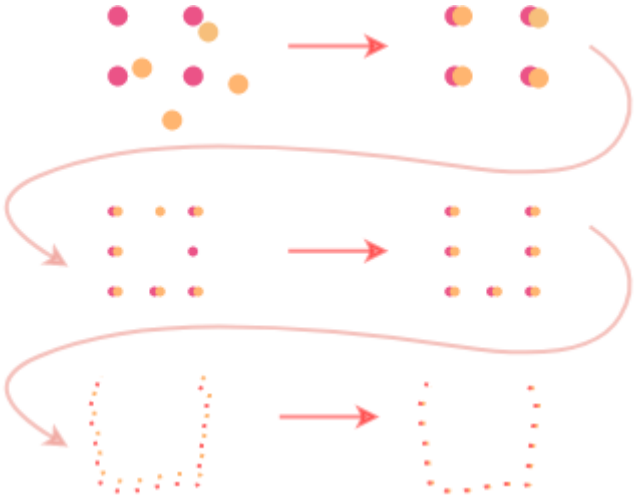


Figure 6.2: First row displays the point cloud with a high degree of downsampling and its aligned result. The second row displays further alignment that occurs with less downsampling. The last row has no downsampling and alignment according to the surface details of the object.

Multi-Resolution ICP is performed by applying ICP in stages. In earlier stages, we downsample our point clouds to a point where important geometrical features such as edges and corners are retained. By reducing the number of pixels in the earlier stages, we can look for correspondences much further in the scene. If the initial placement of the two point clouds is mutually exclusive, having a larger area to search for correspondences can provide the needed initial alignment required for the success of the algorithm.

Using a larger area for correspondence searching comes at the price of more time-consuming processing but this increase in complexity can be countered by downsampling our point clouds. With the reduction of details in the downsampled point clouds there is a decreased necessity for accuracy in the alignment and as a result, convergence can be reached sooner.

After completing this initial stage of ICP, our point clouds should be coarsely aligned. We then repeat this process by decreasing the degree of downsampling that we apply. During our last stage of multi-resolution ICP, we apply the algorithm to our original point clouds that have been closely aligned. This last stage allows us to align our point clouds to the highest degree possible with the information we have available. This process of iteratively decreasing the degree of downsampling and aligning according to the available information is shown more clearly in figure 6.2. With our point clouds in

their current alignment, we can also expect that the correspondence search radius and the number of steps of ICP to be performed will be low.

Through multi-resolution ICP we can reduce the number of overall steps required by the process while retaining the accuracy we would achieve by comparing the original point clouds. Due to the manner in which multi-resolution ICP is applied, we can use it in tandem with our other variants of ICP.

6.2.2 Point-to-Plane

The Vanilla ICP algorithm makes use of point-to-point comparisons to compute the error between the placement of two point clouds. Chen and Medioni [1992] shows that by making a small change to this metric that we aim to minimize we can decrease the number of steps it takes to reach convergence. Including this adjustment allows us to view the error based on the surface of the object as opposed to simply the location of the point.

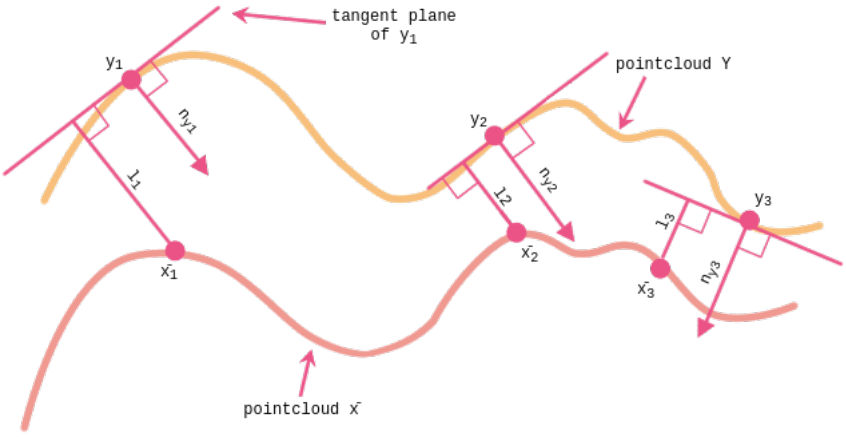


Figure 6.3: Diagram showcasing the effect of using the Point-to-Plane ICP approach. The normal vector included in our new objective function for point-to-plane equation is shown by $\mathbf{n}_{y1}$, $\mathbf{n}_{y2}$ and $\mathbf{n}_{y3}$. The distance we consider is displayed by $\mathbf{l}_1$, $\mathbf{l}_2$ and $\mathbf{l}_3$.

To include this change in our ICP algorithm, we simply apply the dot product of the normals from our source point cloud to our error function. Through this, our recorded distance instead becomes the distance between the target point and its corresponding intersection when projected onto the normal of the closest point in the source point cloud. If we consider figure 6.3, instead of calculating the direct distance between $\mathbf{x}_1$ and $\mathbf{y}_1$, we take the distance $\mathbf{l}_1$. As described by Rusinkiewicz and Levoy [2001], our point clouds converge much sooner when we consider this metric.

6.2.3 Coloured ICP

The inclusion of surface information allows us to efficiently align our point clouds. To further incorporate more information from the data we have available we can make use of coloured ICP [Park *et al.* 2017]. Coloured ICP further alters our objective function to include both the geometric term from our previous approaches as well as a photometric error that contains information about the colours of the corresponding points, denoted as $\mathbf{E}_G$ and $\mathbf{E}_C$ respectively as shown in equation 6.2. We also include a weighting term σ that helps us determine which of our error terms is more important.

$$\min_{E} \mathbf{E}(\mathbf{y}_i, \overline{\mathbf{x}}_i) = (1-\sigma)\mathbf{E}_C(\mathbf{y}_i, \overline{\mathbf{x}}_i) + \sigma\mathbf{E}_G(\mathbf{y}_i, \overline{\mathbf{x}}_i) \tag{6.2}$$

Our $\mathbf{E}_G$ term is the same as the error function we use in our point-to-plane approach but we also include an extra photometric error to compare the colour between two point clouds. Including the colour information further allows us to accurately align the features of our two point clouds when there is a lack of surface detail. The geometric term contains information from our point-to-plane approach while our photometric term contains a numerical value that represents the difference between the colours of the two closest points that we compared in our geometric term. The inclusion of this variant increased the alignment success when working with objects that had multiple colours across their surface. This form of ICP can help in cases where our reconstructed object is symmetrical and plain but has visual information on its surface.

6.2.4 Applied Approach

The ICP method that reaches convergence the fastest often differs based on the object that is scanned. We are able to test these various approaches through the use of the Open3D library from Zhou *et al.* [2018]. If the object has irregular surfaces or lots of varying colours on its surface we can assume that coloured ICP will work best. In our case, due to the lighting conditions that our objects were being scanned in, we had to make use of different variants of ICP in tandem.

We split our alignment process into two phases. During the first phase, we perform a coarse alignment by employing multi-resolution ICP with a large radius for our correspondence search. We apply a large degree of downsampling during the first stage of ICP and move down to a lower degree of downsampling in the last stage. We also make use of coloured ICP for our objective function as we can expect reliable colour matching even in our downsampled point clouds.

Our next phase of alignment also applies multi-resolution ICP but the degree of downsampling applied here is lower. As we expect our first phase to already have performed some degree of alignment, the radius that we search for correspondences in decreases. Our last stage of this phase's multi-resolution ICP includes no downsampling as we are attempting to perform accurate alignment with as much information as possible. This phase of ICP only implements point-to-plane comparison in its objective function.

Using this approach gave us more accurate alignments and can be attributed to the changing of colour on the surface of the object. As the lighting is static and exists in a specific area of the scene, the colours on the object's surface can vary depending on the direction the object is facing. Due to this possibly affecting the accuracy of our fine alignment, we had to focus on the reliable surface information that we had.

Since there is a chance that our first phase still performs better (possibly due to overfitting our models to an incorrect local minima), we compare the degree of alignment of both phases and apply the transformation that overlaps the most. There is also a risk of the degree of overlap being dependent on which object is bigger, we compare the overlap from the perspective of both point clouds and take the higher value. This allows us to measure the overlap in a way that is more fairly indicative of the algorithm's success.

6.3 Model Connectivity

Repeatedly performing alignment between our current frame and the previous frame allows us to build up an adequate surface that describes our object. But when our built surface reaches a part of the object that has been scanned before the shape does not correctly align with itself. Instead, the arc that the built surface follows is either too wide or bends in on itself as shown in figure 6.4.

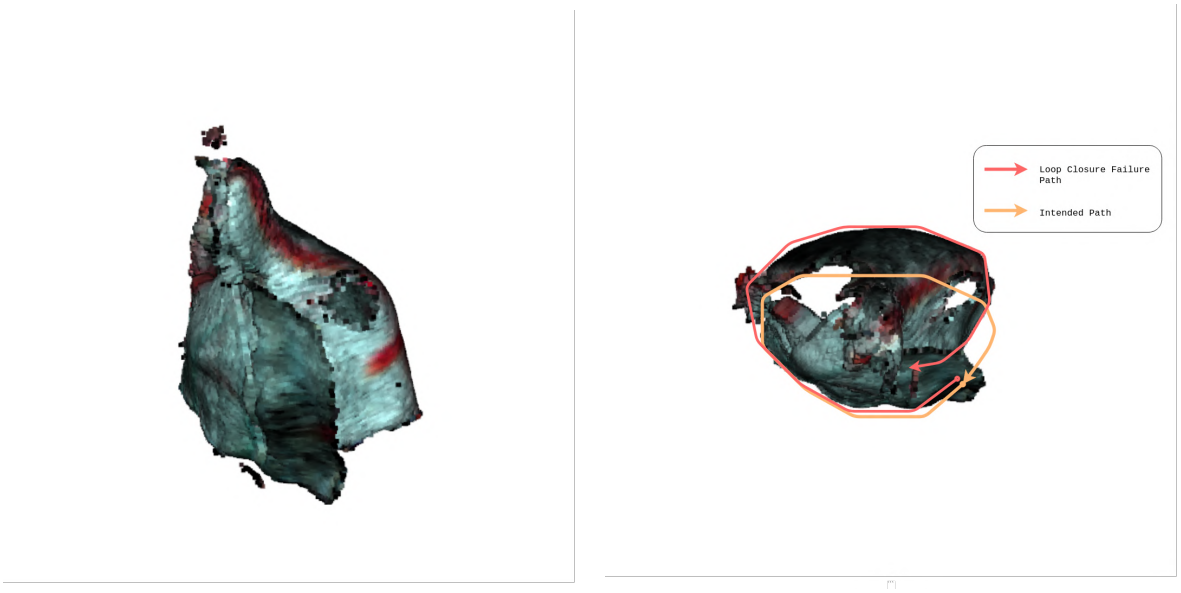


Figure 6.4: Example of loop closure failure on Batman model.

This is a result of an accumulated error in our alignment process. Due to the nature of using estimated correspondences between different frames, we often align our point clouds well enough as opposed to exactly correctly. These alignments are reliable when we consider a few frames of information but over time, the slight amount of error in our alignments build up. This causes the shape to curve at an incorrect angle and leaves us with a flawed result.

When mapping out a larger area through the use of SLAM, the loop closure issue is a common occurrence as described in Kim *et al.* [2022]. A graph-based solution is often implemented in these cases and can be applied to our problem too [Latif *et al.* 2013]. To do this, we need to view each of our frames of data as a node on our graph. The edges of our graph then describe the transformation required to overlap the two connected nodes. Since each of our nodes is an entire point cloud, we can denote a node as $\mathbf{X}_i$ and its corresponding pose as $\mathbf{x}_i$. If we have an edge from $\mathbf{X}_i$ and $\mathbf{X}_j$, we can also conceptually view this edge as the position of $\mathbf{X}_j$ as seen from $\mathbf{X}_i$ based on an observation $\mathbf{z}_{ij}$. In every edge, we store an observation $\mathbf{z}_{ij}$ and an information matrix $\mathbf{m}_{ij}$. $\mathbf{m}_{ij}$ contains our uncertainty of the observation $\mathbf{z}_{ij}$, essentially providing a weight for our edge.

We can assume that sequential nodes $\mathbf{X}_i$ and $\mathbf{X}_{i+1}$ are next to each other and will often result in successful alignment. As a result, we record edges with a high degree of certainty between them. To cater to loop closure cases, however, we have to make comparisons between our current node and every other node recorded as there is no real indication of when the loop will close. If there is a large degree of alignment between two non-contiguous nodes, we add an edge between the two and calculate our information matrix for the edge. In our approach, our certainty is based on our alignment of the two point clouds after ICP has been performed. The higher the degree of alignment, the more certain we are that this edge is important to the posegraph. Non-contiguous nodes with highly certain edges define the parts of the object where the shape needs to connect to itself.

Once we have established the overall shape of our graph along with all the different edges that occur between different nodes, there are often differences between the positions of nodes based on our observations and the original positions of the nodes according to our graph. The difference between these two positions is defined as our error for our posegraph. Our main goal through optimization is to minimize this error and generate a posegraph that is closer to our actual observations of the target [Grisetti *et al.* 2010].

Algorithm 2 We can represent a simplified version of the posegraph approach using the following algorithm:

```
initialize (posegraph)
for i ← 0 to n do
    posegraph ← AddNewNode (posegraph)
    for j ← 0 to n do
        transform ← AttemptICP (X_i, X_j)
        success ← EvaluateTransformation (X_i, X_j, transform)
        if success then
            posegraph ← AddNewEdge (posegraph, transform)
        end if
    end for
end for
posegraph ← OptimizePosegraph (posegraph)
```

6.4 Model Integration

Once we have our optimized graph, we need to merge the different point clouds efficiently to produce a model. As introduced in chapter 2, we can use a TSDF map to generate a reliable record of the shape of our data. The TSDF approach aims to convert an abundance of points in a space or the lack thereof into a description of how far a region is from a surface.

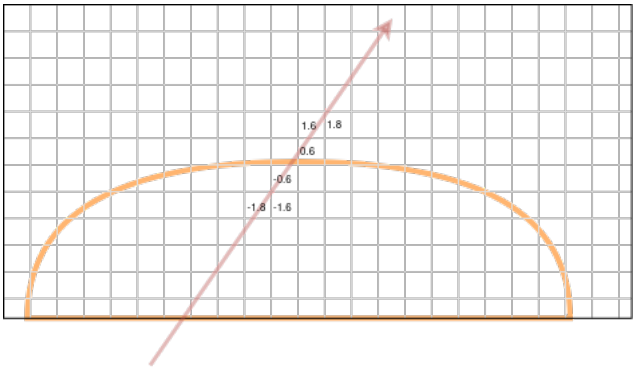


Figure 6.5: Diagram showcasing the method in which depth information is converted into SDF values. This example displays the effect a single ray has on the SDF value at certain voxel centers. When this is actually done, we have a ray for every pixel of our depth information.

We first establish the grid of blocks in which we will build up our model. Each block is defined as a voxel and the voxel size determines how large of an area of the point cloud is discretized into a single value. As a result of this, the voxel size naturally regulates the resolution of the output model. The smaller the voxel size, the higher the number of voxels that can fit into a grid of a certain size. And the more voxels in a grid, the higher the quality of surface details we can record in our output. However, due to the way Open3D generates a voxel block grid, keeping it in memory at all times is an expensive process. As a result, we need to choose a size that suits our purpose while also allowing our block grid to work with the amount of memory we have available.

Whenever we add a new point cloud to our voxel block grid, we have to convert the depth information into a signed distance value. The depth information is converted into an approximate distance from a surface. This is done by simulating the path of a ray from our camera position for the current frame as shown in figure 6.5. We follow the ray's path and estimate how far it is from a surface at every voxel center the ray goes through. This distance value is calculated using our depth information and stored in the voxel center along with a weight that defines how certain we are of the value. If the point is outside the surface, the sign of the value is negative and the opposite is true if the point is inside the surface.

Then when an overlapping point cloud is added, we take the weighted average of the two signed distance values and the sum of the weights. This information is then used to update the values in the corresponding voxel center as shown in figure 6.6. As we

add each of the point clouds in our posegraph with their assigned pose information, we build up a reliable grid of information. Once we have added all the point clouds we have available, we should be able to establish the shape of our final point cloud by looking at the zero-crossings in our grid. We can then retrieve an output model by applying Poisson surface reconstruction as done in Kazhdan *et al.* [2006]. This reconstruction has a surface more similar to the 3D model we use to create our 3D printed objects while also filling in holes in our generated models.

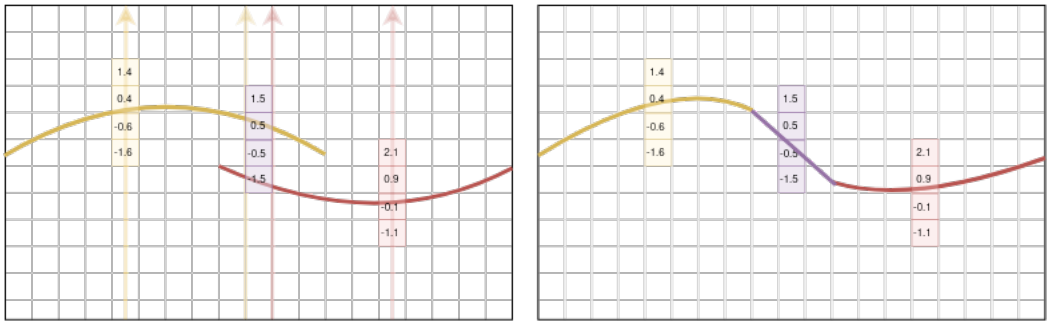


Figure 6.6: Diagram showcasing the method in which information from multiple point clouds is integrated. The diagram on the left shows the average SDF values as a result of the shown rays. The diagram on the right displays the effect of the SDF averaging on the resultant point cloud stored in the voxel block grid.

6.4.1 Evaluation Metrics

To evaluate the success of our reconstructions, we need to calculate the distance between the reference point cloud we used to 3D print our models and the point clouds generated through the application of our pipeline. We calculate our distance using two different methods that are readily available on the CloudCompare Software [Girardeau-Montaut 2023]. the first of which is the Chamfer distance. The Chamfer distance is an often-used metric to compare two point clouds. The process entails getting a rough estimate of our point cloud's accuracy by measuring the distance between every point in a point cloud and its nearest point in the other point cloud [Fan *et al.* 2017]. This distance is calculated from the perspective of both point clouds to get the total distance. If we consider a reference point cloud $\mathbf{Y}$ and a generated point cloud $\mathbf{X}$, we can calculate our Chamfer distance $\mathbf{d}_{chamfer}$ using equation 6.3. We then take the average and the standard deviation of the distance value stored in each point stored in our generated point cloud.

$$\mathbf{d}_{chamfer}(\mathbf{X},\mathbf{Y}) = \sum_{x\epsilon\mathbf{X}} \min_{y\epsilon\mathbf{Y}} \|x-y\|_2^2 + \sum_{y\epsilon Y} \min_{x\epsilon X} \|x-y\|_2^2 \tag{6.3}$$

Our second method of calculating the distance between our reference and generated point cloud is using the quadric height function. Since we currently have two point clouds, using the quadric height function allows us to generate a series of local planes

for the reference point cloud [Lague *et al.* 2013]. These planes act as an estimation of the surface and can better describe the surface of our reference object. If we compare the points on our generated point cloud to the closest point on the local height function as opposed to the closest point on the reference point cloud, we can get a clearer description of the success of our method. The method we use works by generating a local second-order polynomial function with the series of coefficients as shown in equation 6.4. We produce a local height function by fitting our equation to the local curves and calculating the coefficients of the equation. We also need to change the equation depending on the dimension that is the flattest for the current locale.

$$z = \mathbf{a} + \mathbf{b} \cdot x + \mathbf{c} \cdot y + \mathbf{d} \cdot x^2 + \mathbf{e} \cdot xy + \mathbf{f} \cdot y^2 \tag{6.4}$$

Once our quadric functions have been set, we calculate the distance between every point in our generated point cloud and the closest point on the nearest quadric height function. Once we have calculated the distance for each point, we take the average and the standard deviation of every point in our generated point cloud. These averages and standard deviation values are the results stored in table 6.1. The lower the recorded average distance, the higher the accuracy of our generated model. The standard deviation describes the variation in our distances. A higher standard deviation with a lower average can indicate the presence of a few outliers in the model.

The Hausdorff distance, from Mémoli and Sapiro [2004], is another often-used method of comparing our point clouds. However, since we have no noise removal applied to our final reconstruction, there may be cases with outliers. The Hausdorff distance is highly sensitive to these cases and as a result, we selected our earlier mentioned metrics.

6.5 Testing and Experimental Setup

At this point in our pipeline, we had access to a few initial datasets that were recorded with various different conditions in mind. By running these datasets through our reconstruction pipeline we could decipher what elements contributed to successful reconstruction. This knowledge would help us determine how to create datasets for objects moving forward. These tests included various object rotation speeds and orientations, a series of lighting conditions and a few objects that could not be evaluated due to our lack of access to the original model.

Testing various approaches to interacting with objects helped us determine how different hand grasps on the object can affect the quality of depth information and which grasps would still be successfully recognized by our tracking method. This method of testing also helps us determine that the orientation of rotation would not affect our methods success but the speed of the rotation would need to be within the bounds of the depth sensor's limits.

The different lighting conditions highlighted a further limitation of our current due to the skin segmentation networks inability to work under certain lighting conditions.

This would cause elements of the hands to stay in the data during the reconstruction process.

Through the testing of different objects, we can define a series of general features that help the pipeline to perform successfully. These tests also help us determine the method of augmentation that would have the largest affect on the end result and how these can be utilized to effectively increase the range of the pipeline.



Figure 6.7: Image of the Dino Model.

6.5.1 Optimal Conditions for Target

To highlight the features that help our permutation of the pipeline, we have to define what parts of our pipeline deal with the features of the object. For our tested method, this is the Frame-to-Frame ICP comparisons. During this part of our reconstruction, we need to actively be able to recognize important details when point clouds are being compared. Therefore, we can describe elements that support successful and correct point cloud comparisons as beneficial features.

During our testing, we have derived that objects with distinctive, non-repeating features give us the best results. These distinctive features can be a result of areas with unique colours as well as interesting surfaces that are unique to certain parts of the object. These features allow us to make comparisons between different frames much more easily. This means it becomes easier to create valuable edges between the different frames of our pose graph. From the evaluation models that were created, the Dino model as shown in figure 6.7 seemed to be the most reliable for successful reconstruction. The horns and details along its face and limbs contribute to our frame comparisons quite effectively.

6.5.2 Object Augmentation

The presence of unique features aids the creation of a successful posegraph after correct comparisons are generated. The opposite is true for non-ideal features. Repeating details and a lack of surface features can cause incorrect associations and edges to be

established in our posegraph approach. To circumvent these issues and increase the variety of models available we can introduce augmentations to our object.

We implement and test the following augmentations to add variation to our limited set of testable objects:

Color Augmentation Since these models had been 3D printed with single-coloured material, our resultant models were restricted to single colours across their surfaces. We apply our first set of augmentations to test with a different set of objects. By applying stickers onto the surface of our object in a specific pattern, we introduced colour variation to our surface.



Figure 6.8: Image of the Altered Batman Model.

As a result of this, we can even increment the weighting applied to our colour objective function in Coloured ICP to test the effectiveness of this method. We applied this augmentation to our Batman model, as shown in figure 6.8. The stickers were applied in areas that had the least amount of variation so that we could more accurately connect our posegraph. Having a plain edge with simple features contributed to the loop closure failure case shown in figure 6.4.

Surface Alteration Repeating features can often result in false-positive edges between different frames as when they are compared, they might seem to match. This was a problem in our Owl model as it had an abundance of repeating features in the form of feathers. While not all of the feathers have the exact same shape, they have a similar pattern. If different areas with feathers are overlapped during comparison, they might be considered a match by our algorithm.

To test this, we introduce our second augmentation. We augmented the Owl model by adding an object to the surface as shown in figure 6.9. By incorporating a new object with its own set of features, we can decrease the risk of incorrect edges forming in our posegraph. During testing, we noted that this augmentation is more successful when the augment can be viewed from several different angles. This means we can reliably compare and define when there is an incorrect association due to the presence of the augmentation. After the model of our has been generated, we can simply segment out the smaller object.



Figure 6.9: Image of the Altered Owl Model.

To evaluate the overarching success of our pipeline, we first scan all of our 3D printed models using our depth sensor. This segmented information is split into its colour and depth constituents. Our pipeline can accommodate the tracking and segmentation of the object in real-time so we can visually evaluate if our object is being tracked successfully during the scanning process. But once the scanning has been done, our reconstruction process needs to be run separately as we need to generate a posegraph. As our method needs to compare every frame with every other frame to check if edges need to be generated for our posegraph, we cannot run this task in real-time.

The segmented information was initially stored as two separate sequences of PNG images, each containing the colour and depth information respectively. As this could cause a loss of information due to the quantization of the format, we stored our depth information as NPY files to ensure we could store our information effectively. To measure the effectiveness of this change in storage medium, we retained different results for each.

This stored information is then run through our reconstruction section of the pipeline. Here we compare every frame with every other frame as described earlier. Through these comparisons, we build up our posegraph. Once our posegraph has been generated and optimized, we build up our TSDF representation and then convert these to meshes using Poisson reconstruction [Kazhdan *et al.* 2006].

For our experimentation, we run our reconstruction method on the multiple datasets we have recorded for each object. The datasets are split between the original PNG and new NPY formats that were recorded. We perform experimentation on both datasets to get a quantified difference between our two storage methods. We also perform further experimentation on the Owl model with surface augmentation to confirm the positive effect on an otherwise difficult case.

6.6 Results

Using our input models, as shown in figure 6.10, we generate multiple datasets of segmented information. This information is stored as sequential frames of data as dis-

played in figure 6.11. Our recorded tables have multiple models for each object as shown in Table 6.1. The first model recorded for every object (i.e. Batman_1, Dino_1 and Owl_1) is its model generated using depth information stored in PNG format while the second model for every object (i.e. Batman_2, Dino_2 and Owl_2) has its depth information stored using the NPY format. We also have an extra set of results for our Owl model (i.e. Owl_Matchbox) that records the difference with its surface augmentation.



Figure 6.10: Models used for evaluation. From left to right: Current Batman Model (Batman_1 and Batman_2), Dino Model(Dino_1 and Dino_2), Owl Model(Owl_1 and Owl_2) and Augmented Owl Model (Owl_Matchbox)

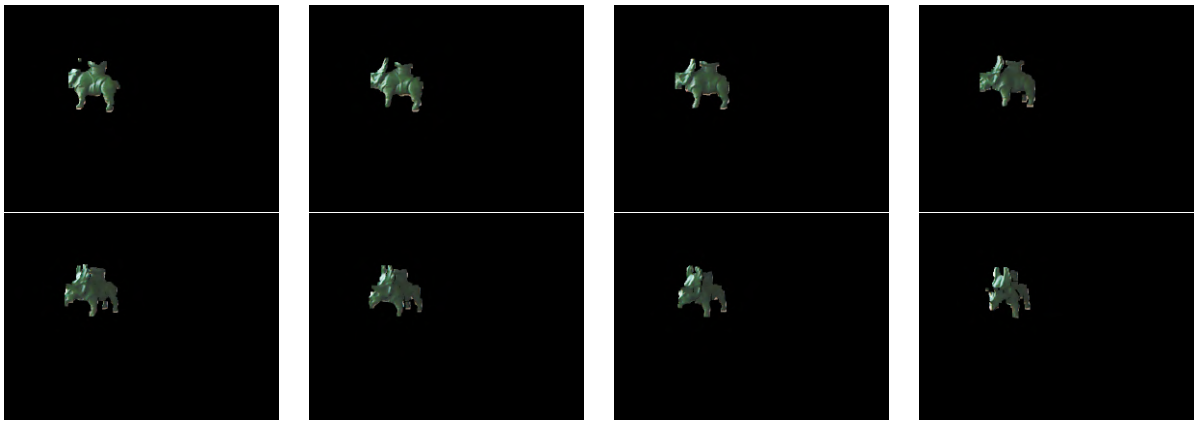


Figure 6.11: Example of sequential data stored in datasets. These images are a subset of the dataset used to generate the model for Dino_2

Table 6.1 confirms our assumptions about the augmentations and optimal conditions we described. We can see that the Dino model in figure 6.14 and figure 6.15 has the lowest average distances which entails that this is our most successful model. This information provides viable evidence of the Dino model being the model with optimal conditions for our pipeline.

We also note the pattern of increase in success when switching over to the second medium of storage for our depth information. This also confirms that converting to a format that decreases the loss of information improves the accuracy of our reconstruction. And lastly, we also note the large increase in the success of our Owl model with

the inclusion of the surface augmentation as shown in figure 6.18. The decrease in the average distance confirms the success of our experimentation with surface alteration.

Table 6.1: Table showing the results of comparing generated models with original reference models.

		Chamfer Distance		Quadric Height Function	
Reference	**Generated**	**Average Dist.**	**Std. Deviation**	**Average Dist.**	**Std. Deviation**
Batman	Batman_1	0.686871	1.11899	1.01515	0.95831
	Batman_2	1.23635	1.45698	1.56631	1.46162
Dino	Dino_1	0.905942	0.868055	1.14138	0.836603
	Dino_2	0.732396	0.801128	0.952422	0.748223
Owl	Owl_1	2.65158	3.15085	3.33507	3.15799
	Owl_2	2.15052	2.39274	2.7974	2.35256
	Owl_Matchbox	1.31549	1.84378	1.88662	1.71579

The figures below from figure 6.12 to figure 6.18 are the resultant meshes generated at the end of our pipeline. They are all displayed alongside an image of the generated model's distance calculated using the quadric height function. The bar in the distance images indicated the colours assigned according to the distance of the point. The bluer colour at the bottom of the bar indicates a small distance between the two models. The higher the indicated colour is on the graph, the further the distance of the point from the reference model. Red points in the model often indicate outliers for our images.



Figure 6.12: Batman_1 Pointcloud and its corresponding results with Quadric Height Function



Figure 6.13: Batman_2 Model and its corresponding results with Quadric Height Function



Figure 6.14: Dino_1 Model and its corresponding results with Quadric Height Function

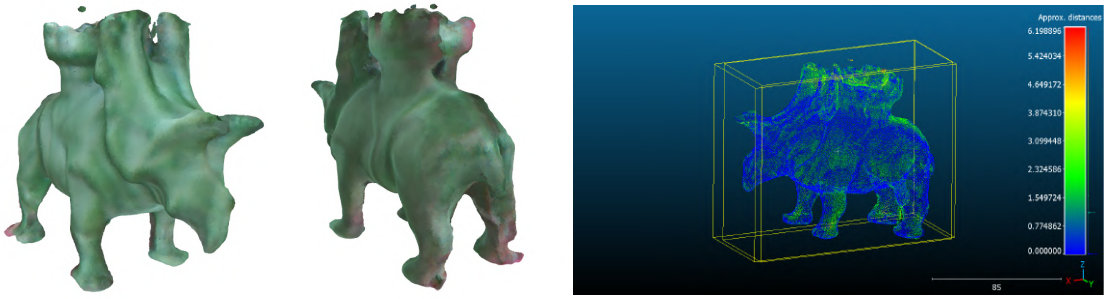


Figure 6.15: Dino_2 Model and its corresponding results with Quadric Height Function



Figure 6.16: Owl_1 Model and its corresponding results with Quadric Height Function

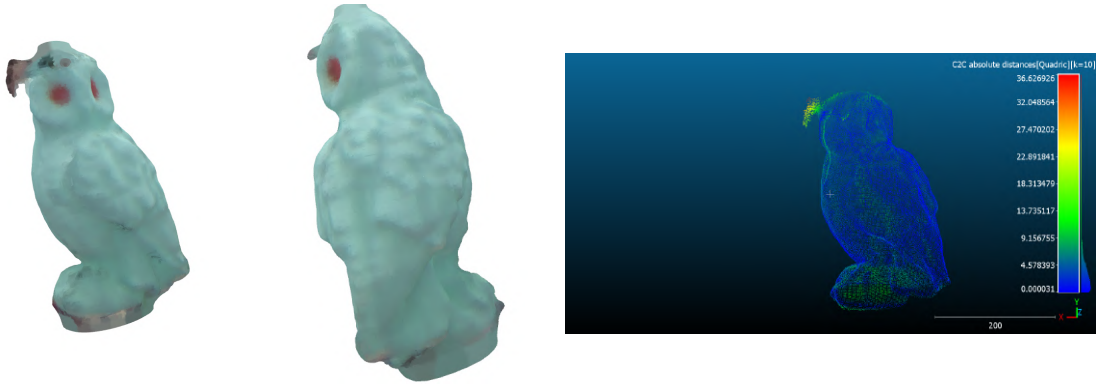


Figure 6.17: Owl_2 Model and its corresponding results with Quadric Height Function

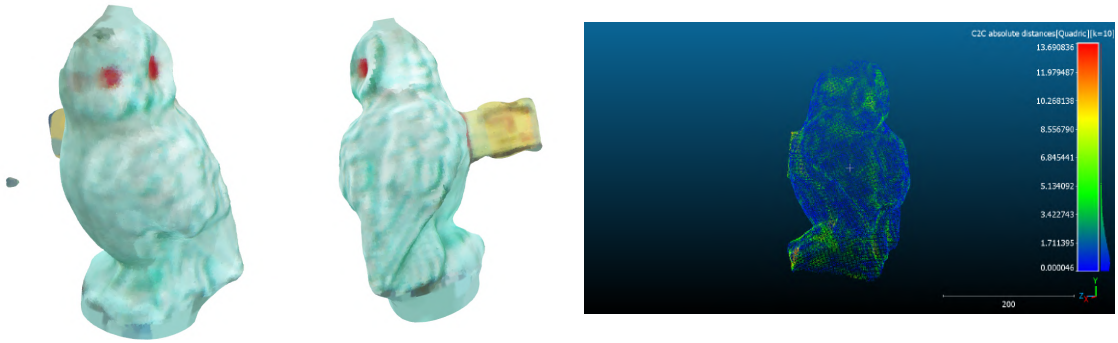


Figure 6.18: Owl_Matchbox Model and its corresponding results with Quadric Height Function after the matchbox has been segmented out.

Chapter 7

Depth Estimation

7.1 Introduction

In this section, we attempt to change the input of our pipeline to incorporate depth estimation. The change to depth estimation allows us to pursue a pipeline that moves away from its reliance on the presence of a depth sensor. The research done in depth estimation has been growing rapidly with the emergence of popular use cases. The need for research in tasks such as self-driving cars has expedited the need for depth estimation techniques. Commodity depth sensors such as the Kinect V1 and V2 are not effective at larger distances and can have trouble while they are in motion. Incorporating more expensive technology to have effective depth information adds to the cost of an already expensive machine. Depth estimation is a more efficient method of retrieving depth information and provides a cost-effective alternative to be applied. We attempt to apply depth estimation in our pipeline to test its effectiveness in being able to recreate the shape of our 3D-printed models.

7.2 Monocular Depth Estimation

The main aim of monocular depth estimation is to generate depth information given only RGB data. This is often done by utilizing several learning-based approaches. We applied monocular depth estimation when generating depth information for our segmentation networks in chapter 5. While the network we used there was effective for getting depth information from our datasets, the output generated when applied to real-world data was not as accurate as we needed. To get clearer results on real-world data we made use of a depth estimation network known as MiDaS. This network, introduced by Lasinger *et al.* [2019], is both well-known and often used as a benchmark for other depth estimation networks.

As mentioned in Lasinger *et al.* [2019], one of the main capabilities of a successful depth estimation network is its robustness to new scenes. There are a wide variety of different environments that can be captured with just RGB information. A network needs to be able to estimate the depth with just this information by making sense of

contextual clues that we can derive from colour data. To do this, this approach needs to make use of a variety of enhanced learning approaches that help incorporate our desired traits.

Currently available datasets for monocular depth estimation such as Kim *et al.* [2018] and Schops *et al.* [2017] can be limited in the variation they provide. To circumvent the inherent biases that can arise in a network trained on a limited dataset, the MiDaS network is trained on a unique mix of multiple large datasets. To augment and enhance the amount of data available, the method mixes different parts of each dataset together to ensure that there is a multitude of visual cues being learned by the network.

When applied to our camera input, the results of depth estimation are visually plausible. The network makes valid suggestions based on the visual cues received in our RGB data. The estimated depth has an excellent understanding of the disparity and 3D positions of objects in comparison to each other. As a result, our mask for segmenting out the object is quite clear and forms an accurate and clean edge as shown in figure 7.1. However, when we convert this estimated depth into a model, it becomes clear that we will not be able to effectively perform reconstruction with this information as shown in figure 7.2.

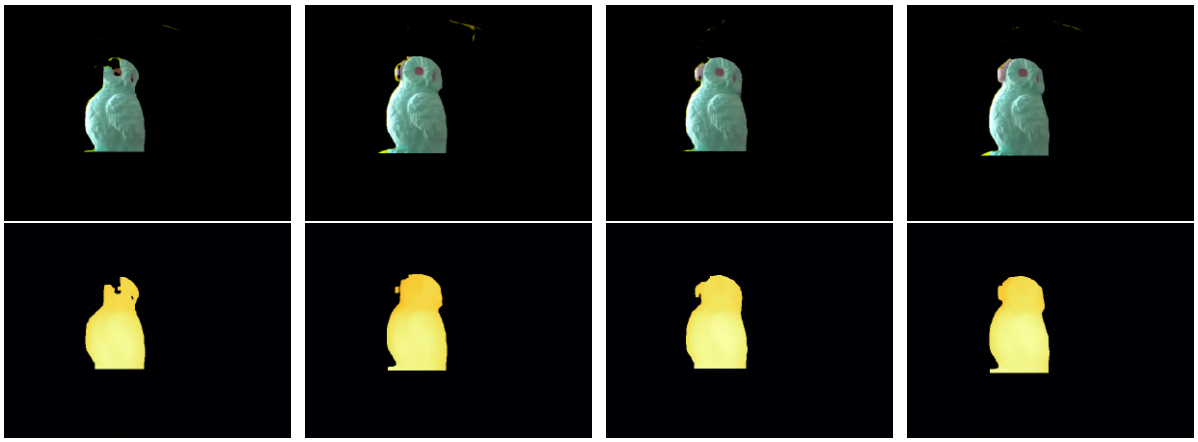


Figure 7.1: Example frames of our segmented RGB and depth estimation data

The point cloud retains a large degree of inaccurate surface information in regards to the object. With our ground-truth depth data, we can understand the details of our object's surface and this largely contributes to the success of our ICP algorithm. When we move to our estimated depth information, along with a dulling of the surface features, the edges of the objects also stretch in 3D space as opposed to cleanly cutting off. When we try to perform ICP on this data, the flattened surfaces align on top of one another to a certain degree. Unfortunately, due to the amount of inaccurate data, we cannot recreate the overall shape of the structure. The depth information has some semblance of the object's shape as we can see from the sunken eyes of the owl model as well as the elevated region of the wing. This information is still not reliable enough to be able to perform an accurate reconstruction of our object.

As shown in figure 7.3, when we attempt to perform the reconstruction section of the pipeline on these estimated models, our output overlaps incorrectly. There is some degree of overlap between the different frames and it moves in a generally correct



Figure 7.2: Depth estimation when viewed as a point cloud

direction but there is no way for each of the frames to fit into each other as each of the frames has its own degree of information loss, resulting in an overarching failure to achieve reconstruction.

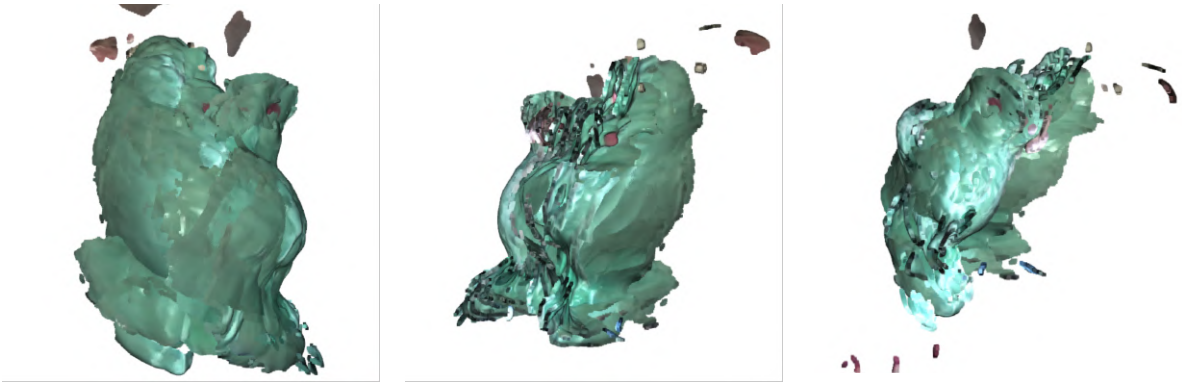


Figure 7.3: Multiple views of our reconstruction attempt through the use of depth estimation.

7.3 Estimation Noise Simulation

After the evaluation of our pipeline using depth estimation and recording the effect of the artefacts that occur in the 3D shape information, we attempt to summarize the main reasons for the failure of reconstruction using estimated depth information. We propose that the flattening of details and unreliable edge information introduce a degree of information loss that can be quantified. By simulating these conditions, we aim to evaluate the robustness of our pipeline when considering depth inaccuracies. Through this, we can also analyse what degree of improvement needs to be introduced into depth estimation to be able to perform successful reconstruction with our pipeline.

Using this, we can ignore edge cases by using a static parameter on the outlier removal algorithm as we are more interested in the degree of flattening that can be applied to our surface while still working with our pipeline. These noise simulation algorithms allow us to emulate the effects of using depth information. Once we apply

these algorithms to our datasets with ground-truth depth data, we can reattempt our reconstruction pipeline.



Figure 7.4: Degradation of information showcased through the increasing kernel size in bilateral filtering



Figure 7.5: Example showcasing the application of the radial outlier removal algorithm. The points highlighted in red are considered outliers and removed when moving to the next step.

7.4 Evaluation and Comparison

Reattempting our reconstruction pipeline with these altered datasets allows us to determine the level of information loss that our current system can handle effectively. We evaluate our reconstructions, as done previously in chapter 6, by applying our two distance metrics to calculate the distance between the model used to 3D print our object and the model generated by our pipeline. We hope to quantitatively display the correlation in the loss of detail with a larger distance between our two point clouds.

We were able to perform two successful reconstructions before the pipeline stopped working due to the amount of information loss. Figure 7.6 is flattened using a bilateral

Table 7.1: Table showing the results of comparing noise affected models with original reference models.

		Chamfer Distance		Quadric Height Function	
Reference	**Generated**	**Average Dist.**	**Std. Deviation**	**Average Dist.**	**Std. Deviation**
Owl	Owl_1	2.65158	3.15085	3.33507	3.15799
	Owl_2	2.15052	2.39274	2.7974	2.35256
	Owl_Matchbox	1.31549	1.84378	1.88662	1.71579
	Owl_k10	2.87668	3.15413	3.57496	3.22032
	Owl_k20	4.91159	4.98098	5.70774	5.12278

filter with its kernel size set to 10. This results in a small degree of deformation and our resultant reconstruction seems quite similar to our reconstruction using ground-truth depth data.

Figure 7.7 has been flattened using a bilateral filter with its kernel size set to 20. There is larger degree of deformation here and although we were able to perform loop closure, this was not done successfully. As shown in figure 7.7, we can clearly see how information loss affects our resultant point cloud. Our posegraph starts making incorrect associations between different frames as the bilateral filtering reduces the prominence of unique details on the surface of our object.



Figure 7.6: Owl_k10 Model and its corresponding results with Quadric Height Function

Lastly, we need to quantify the degree of information loss that our pipeline can handle. We do this by creating a graph that highlights the correlation between the degree of information loss and the average distance between our generated point clouds. We mark the graph at 40% on the x-axis as our cut-off degree of information loss. We estimate that if the information loss of the depth data moves past this point, the chance of a successful reconstruction occurring decreases drastically. If our depth estimation can limit itself to this level of information loss, our pipeline might be able to successfully reconstruct the object. Once we pass this point of information loss, the reconstruction algorithm fails to wrap around in a loop as a complete model and we consider this degree infinitesimally high. Due to this, As shown by figure 7.8, we define an exponential relationship for the loss in data and the distance.

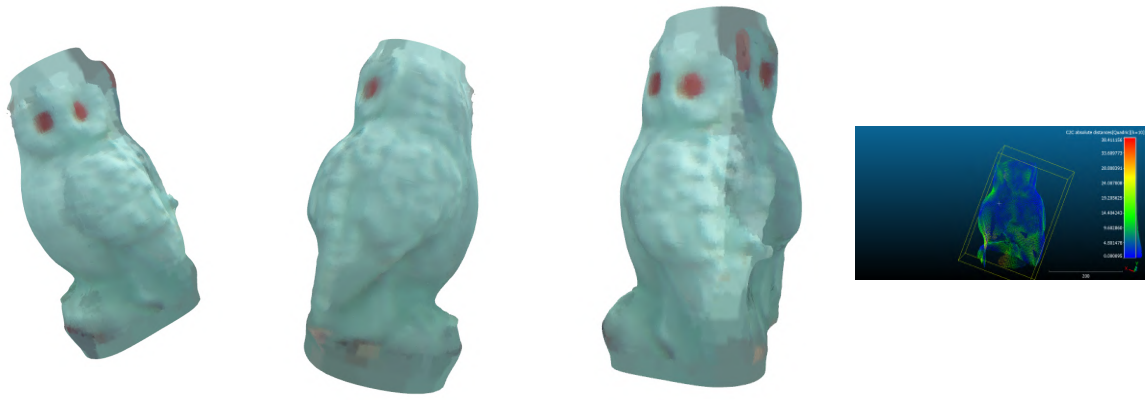
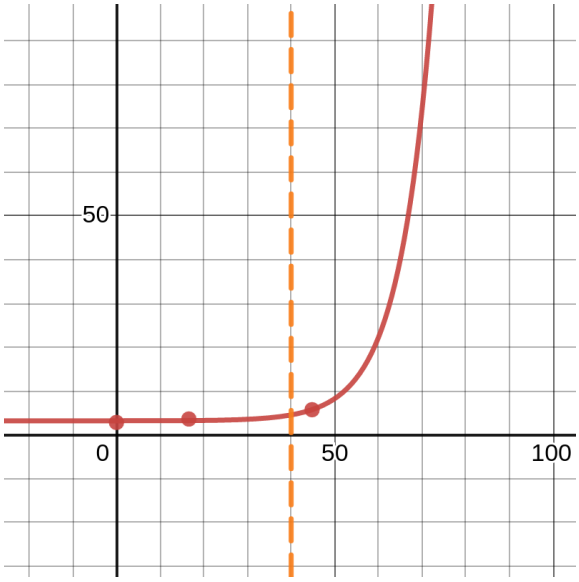


Figure 7.7: Owl_k20 Model, artifact at loop closure area and its corresponding results with Quadric Height Function



	X	Y
	Information Loss	Average Distance
Owl_Matchbox	0%	1.88
Owl_k10	16.58%	3.57
Owl_k20	44.78%	5.71

Figure 7.8: Graph displaying the correlation between information loss and average distance of our generated point clouds using the quadric height function

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Through our testing of the different approaches in each section of the pipeline, we display the effectiveness of our suggested pipeline. By determining the conditions of our specific scenario and iteratively working towards the setup that applies to our case, we perform the reconstruction of handheld objects. While we need to consider the overall effect that each change has on the other phases of the pipeline, this often falls to what new type of information will be available.

Our handheld object reconstruction method is able to generate models of the targets while also being robust to a certain degree of noise. The robustness of our implemented system also extends to situations such as the failure of hand tracking. Furthermore, our general pipeline provides labels for the larger problems that need to be approached. By considering the specific details surrounding experimentation such as what kind of equipment is available and what kind of target we will be recording, we can effectively recreate our own unique pipeline for the successful reconstruction of our handheld objects.

Providing the structure for the problem to be broken down into helps to pave forward the path for future research. By focusing on researching horizontally, we can create a catalogue of different permutations that are the most effective for a variety of different situations.

Through our research, we introduce a variety of different attempted approaches that can replace each section of our pipeline. This leaves us with a highly variable pipeline that can be customized to many different scenarios. We can improve each section separately without having to alter our other phases of the pipeline. As long as we keep track of what information is being received from the previous section and effectively make use of this data, an improvement in any section acts as an improvement for our pipeline as a whole.

Being able to pinpoint which part of the pipeline needs to be changed to make this process successful for specific scenarios improves how quickly we can locate our problem

space. For example, if we need to reconstruct a handheld object from the outdoors, we need to augment our input phase to provide depth information outdoors. We cannot do this using a structured light depth sensor so it may be more effective to incorporate a different method of depth input.

Horizontally researching the different permutations possible for different scenarios introduces a large degree of experimentation that is possible in the field of handheld object reconstruction. And with further research in this manner, we can reduce the complexity of recreating the task, providing this method of reconstruction as a more publicly available solution when an object needs to be reconstructed. With further research, this could reduce the need for specific equipment such as depth sensors and this task could be implemented in important real-world tasks such as the cataloguing of evidence in police investigations or the scanning of fossils at archaeological dig sites.

Creating a permanent record of evidence by simply showing the target to a camera could prove to be a revolutionary change in important investigations. This digitized information would reduce the risk of mishandling evidence during high-profile cases. On the other hand, creating an immediate scan of a fossil at each step of its discovery can allow us to have an undamaged record of each stage. As a result, if the fossil is damaged during excavation, we still have access to the latest unaffected surface details. There are several different use cases available for handheld object reconstruction and our pipeline's adaptability increases the number of possibilities.

8.2 Future Work

There are many possible permutations of our tested pipeline that can be implemented due to the variability of our conditions. Our assigned target may have some inherent features that prove harder to track, the target may be flexible or we may not have access to ground-truth information. Due to this wide variety of permutations, there is a large amount of possible experimentation to test.

The above-mentioned cases are some of the setups that could be tested in the future. Using hand landmarks to help with the overall placement of frames in the posegraph when the object has no surface features and incorporating a deformation map to test this pipeline's effectiveness with flexible objects are obvious choices for further experimentation.

While testing our depth estimation, we expected some degree of deformation on the depth maps as the frames are not temporally consistent. We would like to test the effectiveness of incorporating methods such as consistent video depth from Luo *et al.* [2020] when considering our targets. Another series of tests that may prove useful would be the inclusion of high-quality images when performing depth information as opposed to the low-resolution data we used.

Furthermore, we also took steps during our experimentation to attempt the inclusion of stereo depth estimation. We wanted to display this on the list of depth estimation methods we tested for success on our pipeline but due to time constraints, we were unable to produce an effective example to test. If possible, we would like to re-approach

stereo-based depth estimation with a commercially available component to compare its success against our currently used depth sensor.

While our noise simulation also introduced a degree of deformation onto the object surfaces, we would like to test if we could record the degree of deformation required at the posegraph phase for a viable complete model. If we could apply this deformation and reduce the flattening introduced by depth estimation, we could possibly update this pipeline to work with depth estimation as its input.

While we attempted to implement a network for hand segmentation, the training done for this section was quite flawed due to both the data we had available as well as the network structures we attempted to implement. While the U-Net architecture is excellent at the segmentation task, there are better networks to implement. Our main goal was to segment out an unknown structure by generating intermediate results for the placement of hands in the scene. We could have incorporated a different network structure that treated the hand segmentation as another series of nodes in the overall network architecture. Since our method often works with videos we can also introduce temporal consistency to our network.

Along with this, we would also like to try testing the effectiveness of several different approaches to setting up the formation of the posegraph. Setting up the posegraph by comparing every frame to every other frame is robust but also takes a large amount of time. We attempted some naive approaches to decreasing the number of frames that need to be checked but we would like to approach these solutions in a more involved way.

Introducing a step value that records how many frames we skip before checking left us with a sparse amount of connections in our posegraph. We have noted during the process that when a correct edge is generated, the frames surrounding the current frame are highly likely to also generate an edge as the surrounding frames are often observing a similar scene.

To improve on this and reintroduce a dense amount of connections when correct edges are generated, we would like to test the effectiveness of creating a list of frames that need to be checked based on our number of frames and a high step value. Then if an edge is detected, we update our list to test the frames surrounding the successful edge found. To make the process more robust, we can iteratively check if the number of edges is too low by the end of the process and update our list to check a few more frames by decreasing our step value.

We would also like to update the process to test the effectiveness of online loop closure. While we worked with the posegraph-based approach as it provided us with successful models, the amount of time the offline approach takes is a strong disadvantage with our tested permutation of the pipeline. We can improve on this by introducing online loop closure to further increase the effectiveness of the pipeline as recreating the object in real-time is more intuitive. Recording the object in real-time is more intuitive as you can understand what part of the object still needs to be recorded and rotate the object accordingly.

This might likely go hand-in-hand with the usage of a surfel representation for our object. Along this line of change in format, replacing depth estimation techniques through the use of a neural SDF representation is a possible interesting implementation to test.

References

[Amos 2011] Evan Amos. *The Kinect sensor for the Xbox 360*, 2011. Accessed on March 23, 2023.

[Anderson *et al.* 2015] Sean Anderson, Kirk MacTavish, and Timothy D Barfoot. Relative continuous-time slam. *The International Journal of Robotics Research*, 34(12):1453–1479, 2015.

[Andrews 2011] David L Andrews. *Structured light and its applications: An introduction to phase-structured beams and nanoscale optical forces*. Academic press, 2011.

[Angeli *et al.* 2008] Adrien Angeli, Stéphane Doncieux, Jean-Arcady Meyer, and David Filliat. Real-time visual loop-closure detection. In *2008 IEEE international conference on robotics and automation*, pages 1842–1847. IEEE, 2008.

[Bambach *et al.* 2015] Sven Bambach, Stefan Lee, David J. Crandall, and Chen Yu. Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[Besl and McKay 1992] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

[Bouguet and others 2001] Jean-Yves Bouguet et al. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel corporation*, 5(1-10):4, 2001.

[Carrozzino and Bergamasco 2010] Marcello Carrozzino and Massimo Bergamasco. Beyond virtual museums: Experiencing immersive virtual reality in real museums. *Journal of Cultural Heritage*, 11(4):452–458, 2010.

[Chen and Medioni 1992] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[Curless and Levoy 1996] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[Du *et al.* 2020] Getao Du, Xu Cao, Jimin Liang, Xueli Chen, and Yonghua Zhan. Medical image segmentation based on u-net: A review. *Journal of Imaging Science and Technology*, 64:1–12, 2020.

[Durrant-Whyte and Bailey 2006] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[Elfes and Matthies 1987] Alberto Elfes and Larry Matthies. Sensor integration for robot navigation: combining sonar and stereo range data in a grid-based representataion. In *26th IEEE conference on decision and control*, volume 26, pages 1802–1807. IEEE, 1987.

[Fan *et al.* 2017] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[Fremont and Chellali 2004] Vincent Fremont and Ryad Chellali. Turntable-based 3d object reconstruction. In *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, volume 2, pages 1277–1282. IEEE, 2004.

[Garcia *et al.* 2009] Pablo Garcia, Jacob Rosen, Chetan Kapoor, Mark Noakes, Greg Elbert, Michael Treat, Tim Ganous, Matt Hanson, Joe Manak, Chris Hasser, et al. Trauma pod: a semi-automated telerobotic surgical system. *The International Journal of Medical Robotics and Computer Assisted Surgery*, 5(2):136–146, 2009.

[Girardeau-Montaut 2023] Daniel Girardeau-Montaut. *CloudCompare*, 2023.

[Gonzalez-Barbosa *et al.* 2015] José-Joel Gonzalez-Barbosa, José-Guadalupe Rico-Espino, Roberto-Augusto Gómez-Loenzo, Hugo Jiménez-Hernández, Miguel Razo, and Ricardo Gonzalez-Barbosa. Accurate 3d reconstruction using a turntable-based and telecentric vision. *Automatika: časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije*, 56(4):508–521, 2015.

[Graves and Graves 2012] Alex Graves and Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[Grisetti *et al.* 2010] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[Hall-Holt and Rusinkiewicz 2001] Olaf Hall-Holt and Szymon Rusinkiewicz. Stripe boundary codes for real-time structured-light range scanning of moving objects. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 359–366. IEEE, 2001.

[Hartley and Zisserman 2003] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[Huang *et al.* 2022] Di Huang, Xiaopeng Ji, Xingyi He, Jiaming Sun, Tong He, Qing Shuai, Wanli Ouyang, and Xiaowei Zhou. Reconstructing hand-held objects from monocular video. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022.

[Innmann *et al.* 2016] Matthias Innmann, Michael Zollhöfer, Matthias Nießner, Christian Theobalt, and Marc Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *European conference on computer vision*, pages 362–379. Springer, 2016.

[Jared *et al.* 2015] Heinly Jared, Johannes L Schonberger, Enrique Dunn, and Jan-Michael Frahm. Reconstructing the world in six days. In *CVPR*, 2015.

[Jones and Rehg 2002] Michael J Jones and James M Rehg. Statistical color models with application to skin detection. *International journal of computer vision*, 46(1):81–96, 2002.

[Jost and Hugli 2003] Timothée Jost and Heinz Hugli. A multi-resolution icp with heuristic closest point search for fast and robust 3d registration of range images. In *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 427–433. IEEE, 2003.

[Kabzan *et al.* 2020] Juraj Kabzan, Miguel I Valls, Victor JF Reijgwart, Hubertus FC Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, et al. Amz driverless: The full autonomous racing system. *Journal of Field Robotics*, 37(7):1267–1294, 2020.

[Kazhdan *et al.* 2006] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[Kim *et al.* 2018] Youngjung Kim, Hyungjoo Jung, Dongbo Min, and Kwanghoon Sohn. Deep monocular depth estimation via integration of global and local predictions. *IEEE transactions on Image Processing*, 27(8):4131–4144, 2018.

[Kim *et al.* 2022] Taehun Kim, Kunhee Kim, Joonyeong Lee, Dongmin Cha, Jiho Lee, and Daijin Kim. Revisiting image pyramid structure for high resolution salient object detection. In *Proceedings of the Asian Conference on Computer Vision*, pages 108–124, 2022.

[Lague *et al.* 2013] Dimitri Lague, Nicolas Brodu, and Jérôme Leroux. Accurate 3d comparison of complex topography with terrestrial laser scanner: Application to the rangitikei canyon (nz). *ISPRS journal of photogrammetry and remote sensing*, 82:10–26, 2013.

[Lasinger *et al.* 2019] Katrin Lasinger, René Ranftl, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *CoRR*, abs/1907.01341, 2019.

[Latif *et al.* 2013] Yasir Latif, César Cadena, and José Neira. Robust loop closing over time for pose graph slam. *The International Journal of Robotics Research*, 32(14):1611–1626, 2013.

[Lee *et al.* 2022] Min Seok Lee, WooSeok Shin, and Sung Won Han. Tracer: Extreme attention guided salient object tracing network (student abstract). In *Proceedings*

of the AAAI Conference on Artificial Intelligence, volume 36, pages 12993–12994, 2022.

[Lin *et al.* 2014] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[Longuet-Higgins 1981] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135, 1981.

[Lowe 1999] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[Luo *et al.* 2020] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf. Consistent video depth estimation. *ACM Transactions on Graphics (ToG)*, 39(4):71–1, 2020.

[Mémoli and Sapiro 2004] Facundo Mémoli and Guillermo Sapiro. Comparing point clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 32–40, 2004.

[Mescheder *et al.* 2019] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4460–4470, 2019.

[Mester *et al.* 2001] Rudolf Mester, Til Aach, and Lutz Dümbgen. Illumination-invariant change detection using a statistical colinearity criterion. In *Joint Pattern Recognition Symposium*, pages 170–177. Springer, 2001.

[Newcombe *et al.* 2011] Richard Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *IEEE International Symposium on Mixed and Augmented Reality*, volume 201, pages 127–136, 2011.

[Newcombe *et al.* 2015] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, 2015.

[Niklaus *et al.* 2019] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3d ken burns effect from a single image. *ACM Transactions on Graphics*, 38(6):184:1–184:15, 2019.

[Nikoohemat *et al.* 2020] Shayan Nikoohemat, Abdoulaye A Diakité, Sisi Zlatanova, and George Vosselman. Indoor 3d reconstruction from point clouds for optimal routing in complex buildings to support disaster management. *Automation in construction*, 113:103109, 2020.

[Orest 2018] Orest. *Battle-Dinosaur-1*, 2018. Accessed on March 23, 2023.

[Park *et al.* 2017] Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Colored point cloud registration revisited. In *Proceedings of the IEEE international conference on computer vision*, pages 143–152, 2017.

[Patel 2020] Jay Patel. *Garden-Owl-1*, 2020. Accessed on March 23, 2023.

[Pfister *et al.* 2000] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000.

[Presl 2022] Jan Presl. *Batman-86*, 2022. Accessed on March 23, 2023.

[Raneri 2018] Domenic Raneri. Enhancing forensic investigation through the use of modern three-dimensional (3d) imaging technologies for crime scene reconstruction. *Australian journal of forensic sciences*, 50(6):697–707, 2018.

[Remondino and El-Hakim 2006] Fabio Remondino and Sabry El-Hakim. Image-based 3d modelling: a review. *The photogrammetric record*, 21(115):269–291, 2006.

[Remondino 2003] Fabio Remondino. From point cloud to surface: the modeling and visualization problem. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34, 2003.

[Ronneberger *et al.* 2015] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[Runz *et al.* 2018] Martin Runz, Maud Buffier, and Lourdes Agapito. Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 10–20. IEEE, 2018.

[Rusinkiewicz and Levoy 2001] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001.

[Rusinkiewicz *et al.* 2002] Szymon Rusinkiewicz, Olaf Hall-Holt, and Marc Levoy. Real-time 3d model acquisition. *ACM Transactions on Graphics (TOG)*, 21(3):438–446, 2002.

[Schops *et al.* 2017] Thomas Schops, Johannes L Schonberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3260–3269, 2017.

[Sener and Koltun 2018] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

[Taketomi *et al.* 2017] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):1–11, 2017.

[Tjaden *et al.* 2016] Henning Tjaden, Ulrich Schwanecke, and Elmar Schömer. Real-time monocular segmentation and pose tracking of multiple objects. In *European conference on computer vision*, pages 423–438. Springer, 2016.

[Tzionas and Gall 2015] Dimitrios Tzionas and Juergen Gall. 3d object reconstruction from hand-object interactions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 729–737, 2015.

[Tzionas *et al.* 2014] Dimitrios Tzionas, Abhilash Srikantha, Pablo Aponte, and Juergen Gall. Capturing hand motion with an rgb-d sensor, fusing a generative model with salient points. In *Pattern Recognition: 36th German Conference, GCPR 2014, Münster, Germany, September 2-5, 2014, Proceedings 36*, pages 277–289. Springer, 2014.

[Wang and Hauser 2019] Fan Wang and Kris Hauser. In-hand object scanning via rgb-d video segmentation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3296–3302. IEEE, 2019.

[Wang *et al.* 2019] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3343–3352, 2019.

[Wang *et al.* 2021] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689*, 2021.

[Wang *et al.* 2022] Tengfei Wang, Qingdong Wang, Haibin Ai, and Li Zhang. Semantics-and-primitives-guided indoor 3d reconstruction from point clouds. *Remote Sensing*, 14(19):4820, 2022.

[Weise *et al.* 2008] Thibaut Weise, Bastian Leibe, and Luc Van Gool. Accurate and robust registration for in-hand modeling. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[Weise *et al.* 2011] Thibaut Weise, Thomas Wismer, Bastian Leibe, and Luc Van Gool. Online loop closure for real-time interactive 3d scanning. *Computer Vision and Image Understanding*, 115(5):635–648, 2011.

[Wen and Bekris 2021] Bowen Wen and Kostas Bekris. Bundletrack: 6d pose tracking for novel objects without instance or category-level 3d models. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8067–8074. IEEE, 2021.

[Xiang *et al.* 2017] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[Xu *et al.* 2017] Jie Xu, Lieyun Ding, and Peter ED Love. Digital reproduction of historical building ornamental components: From 3d scanning to 3d printing. *Automation in Construction*, 76:85–96, 2017.

[Ye *et al.* 2022] Yufei Ye, Abhinav Gupta, and Shubham Tulsiani. What's in your hands? 3d reconstruction of generic objects in hands. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3895–3905, 2022.

[Zach *et al.* 2007] Christopher Zach, Thomas Pock, and Horst Bischof. A globally optimal algorithm for robust tv-l1 range image integration. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, 2007.

[Zhang *et al.* 2019] Hao Zhang, Zi-Hao Bo, Jun-Hai Yong, and Feng Xu. Interactionfusion: real-time reconstruction of hand poses and deformable objects in hand-object interactions. *ACM Transactions on Graphics (TOG)*, 38(4):1–11, 2019.

[Zhang *et al.* 2020] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.

[Zhang *et al.* 2021] Hao Zhang, Yuxiao Zhou, Yifei Tian, Jun-Hai Yong, and Feng Xu. Single depth view based real-time reconstruction of hand-object interactions. *ACM Transactions on Graphics (TOG)*, 40(3):1–12, 2021.

[Zhou *et al.* 2018] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *CoRR*, abs/1801.09847, 2018.