Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Building a 3D Object Scanner

Bachelor Thesis

Ulla Aeschbacher

ullaa@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger, Gino Brunner
Prof. Dr. Roger Wattenhofer

September 8, 2017

Acknowledgements

I would like to thank my supervisors Manuel Eichelberger and Gino Brunner for their input in our meetings and their support throughout the project. I also thank my friends and family for proofreading and giving encouragement in times of need.

Abstract

This thesis explores the possibilities provided by Google Tango. We present an application for a smartphone that offers an easy way to scan objects and turn them into 3D models. We show how to acquire point clouds from an object and transform them so that multiple scans can be combined. Then, we describe an algorithm to transform the point clouds into surfaces. We also try to remove the underlying plane of objects with mixed results.

Contents

Chapter 1

Introduction

1.1 Motivation

3D scanning is the process of transforming an object of the real world into a digital surface model. Until recently, this required specialized and relatively expensive equipment. But with Google Tango [1] enabled smartphones soon everybody could have such a device in their pocket at no additional costs. Content creators could quickly scan an object they want to include in their animation short or game. Another application is the use of 3D printers to reproduce the scanned object. Both would be a great time saver because people do not have to hand-draw those models anymore. People could share models they created in a open library to maximize reuse.

The goal of this project is to create an app for a Google Tango device which allows one to create 3D models from arbitrary objects standing on a plane. Ideally, we cut out the plane afterwards, so the object can be handled on its own in a versatile manner.

We are working with the Lenovo Phab 2 Pro, which has the required components and computing capability, as shown in Figure 1.1. We mainly use three of the phone's sensors. For depth perception we need the IR projector which sends out infrared light and the time-of-flight camera, which resolves distance based on the known speed of light. It measures the time-of-flight of a light signal between the camera and the subject for each point in the image and thus gives us a depth for each of those points. For motion tracking the fisheye camera is needed. If we wanted the point clouds to have color, we would also have to use the RGB camera.

1.2 Related Work

There have been some applications for the Tango phone to scan your surroundings, but they focus on scanning rooms or appartments, like CHISEL [2] which allows scanning of large, connected areas. The process of turning point clouds

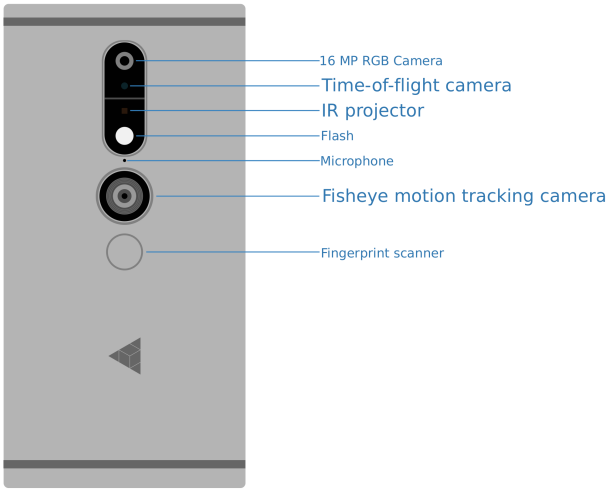


Figure 1.1: The Lenovo Phab 2 Pro Sensors

into surface models has been done many times, for example in [3], where the focus lies on large and noisy datasets. Segmentation of point clouds is also well researched, like in [4], which presents an approach to plane segmentation for tabletop objects. Rusu et al. [5] solve a similar task but additionally to planes they fit spheres, cylinders and cones. This helps patching up missing data and create smooth shapes. Dimitrov et al. [6] take a similar approach but put the focus on industry applications.

Chapter 2

Background

2.1 Motion Tracking and Area Learning

Motion tracking is the ability of a device to track its translations and rotations and to know where it is in space relative to a base point. We need this information to be able to relate mutliple scans to each other. Area learning gives the device the ability to remember what it has scanned before and to recognize learned areas again. This technique gives the following advantages: First it corrects drift that accumulates when scanning for longer times, as shown in Figure 2.1, second the device can localize itself within a learned area.

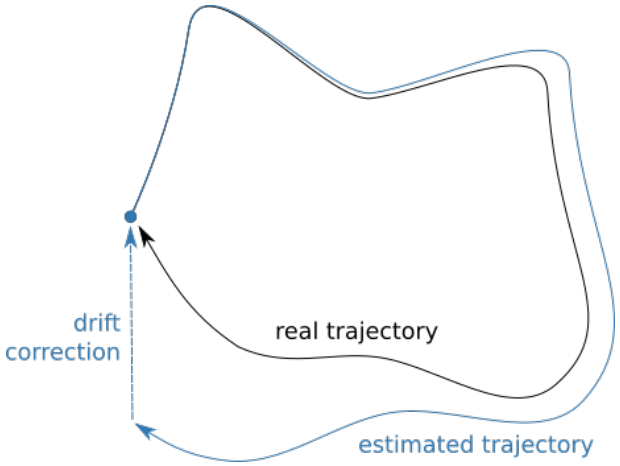


Figure 2.1: Drift Correction

For this thesis, we are using the Google Tango platform for both motion tracking and area learning. When scanning an area, the device gives us a point cloud of the scanned area. A point cloud is a set of (x, y, z)-formatted points, each representing a measured point in the room. It can be manipulated over the APIs [7] provided by Google Tango. In this project we are working with the Java API. Furthermore the phone can save `AreaDescriptionFile`s (ADFs), which

contain a description of the area it has seen while area learning was enabled. The user is required to scan the environment first. Afterwards, they can scan the object with the ADF loaded. As soon as the phone recognizes the area ('localizes') the ADF is automatically used for drift-correction.

2.2 3D Reconstruction and Image Segmentation

The technique to turn point clouds back into 3D surfaces is known as 3D reconstruction. It is often achieved via point set triangulation [8], where all or a subset of the points are connected by edges so that the resulting triangles form a surface. The quality of such a mesh can be evaluated by different means. One way is by the minimum weight, that is the minimum sum of edges, another by penalizing triangles with small angles. Both methods try to make the likeness of the surface to the original object to be as good as possible, while simultaneously trying to keep the mesh simple.

We use the free Point Cloud Library (PCL) [9] to achieve a triangulation [3].

Chapter 3

Implementation

3.1 Overview

We achieve the goal of getting from point clouds to 3D models by going through the following procedure, see also Figure 3.1:

- Acquire point clouds from the phone
- Combine several point clouds together into one by transforming them so they are all with respect to a common base coordinate system
- Reconstruct the surface from the combined point cloud
- Separate objects from the plane they sit on
- Display surface model on the computer

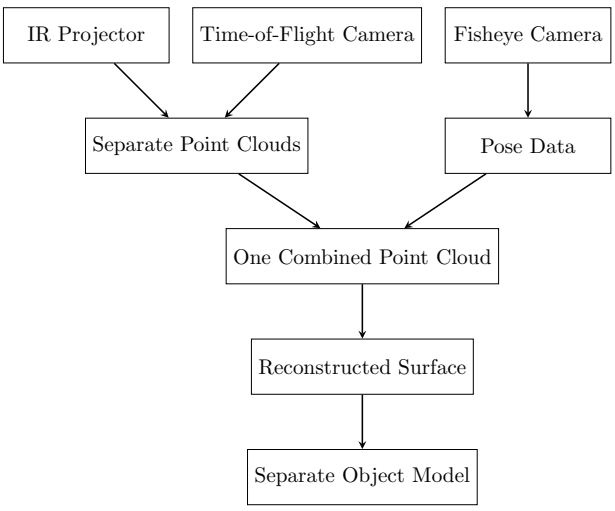


Figure 3.1: Flowchart of the process

3.2 Acquiring Data

When starting the Tango service we initialize a Tango object which is the interface used to connect to the Project Tango service. Then we connect a Listener to it. While scanning, we get four different kinds of events back to our listener `Tango UpdateCallback`.

Poses The first event gives us a `TangoPoseData` object, called a pose from now on. This object contains the rotation and translation of the phone at a timestamp with respect to a base pose. There are three different combinations of target-base-pairs:

- the Device frame with respect to the Start-of-Service frame, which is used if there are no ADFs available
- the Device frame with respect to the Area-Description frame, which is used if there is a ADF loaded and a normal point cloud comes in
- the Area-Description frame with respect to the Start-of-Service frame, which is used if the device localizes itself and updates the ADF

Every time a pose comes in, which is approximately every five milliseconds, it is automatically saved in the Tango object. We only have to update the frustrum on the screen and save the timestamp. The frustrum shows us the direction we are looking in, as seen in Figure 3.3 (b). We only update the information displayed on the screen periodically, so we wait until the timestamps are 100 ms apart before doing so.

Point clouds The second event gives us a `TangoPointCloudData` object, which contains the point cloud aquired at a timestamp. Everytime such an object arrives, approximately every 200 ms, we also update the screen, then query the pose at the same timestamp. This gives us the pose that is closest to the given timestamp and in the specified target-base-pair. We are only interested in the point cloud, if the device has been sufficiently moved since the last point cloud was saved. After the transformation (discussed in section 3.3) and rounding, all points within a certain threshold are saved to a global list.

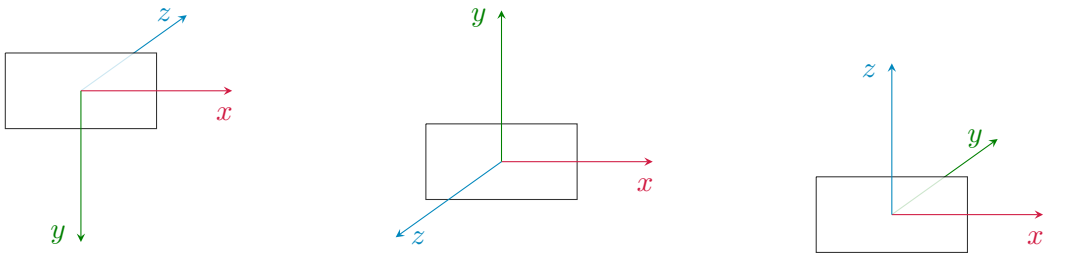
Events Then there are `TangoEvent`s, which are triggered whenever an event occurs that the developer needs to be aware of, like exceptions or notifications of sensors. When that happens, an error code is printed either as an alert on screen or a log entry visible in Android Studio, our programming environment.

Camera images And fourth we get a notification signaling that the RGB or fisheye camera has an image available. We are ignoring these notifications in this project to focus on getting correct depth data.

3.3 Transforming Data

Beeing able to correctly transform points is crucial as every point cloud is recorded with respect to the device. If we would merge them as they are, they would all end up stacking on top of each other. We first have to rotate and translate them according to where the phone was when the particular cloud was scanned. The first pose we get from the phone is used as the base. All point clouds have to be transformed so that they are with respect to the base before we are able to merge the clouds together.

We name some coordinate systems (CS) as follows, and will use them again throughout the thesis. The rectangle represents the phone pointing to the right and with the screen facing the reader:



(a) Phab CS (b) Device CS (c) Start of Service CS

Figure 3.2: Coordinate Systems

The points in `TangoPointCloudData` given by the phone are in the Phab CS, the Device frame is associated with the Device CS and both the Start-of-Service frame and the Area-Description frame are associated with the Start of Service CS. Our device reports its pose relative to its chosen frame of reference. The transformation matrices that allow to switch from one system into the other are as follows ($M_{\mathrm{X}}^{\mathrm{Y}}$ describes the transformation from system X to system Y) :

$$T_{\mathrm{Phab}}^{\mathrm{Device}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad A_{\mathrm{Device}}^{\mathrm{Start\ of\ Service}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

To make sure that all points are in a common coordinate system, we have to do a rotation and a translation. As the common base frame we use the Start-of-Service frame which is set when the Tango service is first started. We now

expect, that the following transformation would yield the desired result:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = A \cdot R \cdot T \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} + t \tag{3.1}$$

Here $A \cdot R$ is the rotation matrix equivalent to the quaternion (q_x, q_y, q_z, q_w) given by the `TangoPoseData`. While quaternions use less space, they are not as intuitive and easy to transform as rotation matrices. The matrix is thus acquired by doing the following:

$A \cdot R =$

$$\begin{pmatrix} 1 - 2 \cdot q_y^2 - 2 \cdot q_z^2 & 2 \cdot q_x \cdot q_y - 2 \cdot q_z \cdot q_w & 2 \cdot q_x \cdot q_z + 2 \cdot q_y \cdot q_w & 0 \\ 2 \cdot q_x \cdot q_y + 2 \cdot q_z \cdot q_w & 1 - 2 \cdot q_x^2 - 2 \cdot q_z^2 & 2 \cdot q_y \cdot q_z - 2 \cdot q_x \cdot q_w & 0 \\ 2 \cdot q_x \cdot q_z - 2 \cdot q_y \cdot q_w & 2 \cdot q_y \cdot q_z + 2 \cdot q_x \cdot q_w & 1 - 2 \cdot q_x^2 - 2 \cdot q_y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} q_x \\ q_y \\ q_z \\ q_w \end{pmatrix}$$

The translation vector t is also given by the `TangoPoseData`. The only part that changes with each new point cloud is R, as the phone is moved around and its pose changes.

But, for a reason that is not clear to me, we have to switch the x and y angles of the R rotation matrix.. We do this by first extracting the angles

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{pmatrix}$$

$$\theta_x = \arctan2\ (r_{32}, r_{33}) \tag{3.2}$$

$$\theta_y = \arctan2\ (-r_{31}, \sqrt{{r_{32}}^2 + {r_{33}}^2}) \tag{3.3}$$

$$\theta_z = \arctan2\ (r_{21}, r_{11}) \tag{3.4}$$

and then creating three rotation matrices as follows

$$X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_y) & -\sin(\theta_y) & 0 \\ 0 & \sin(\theta_y) & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Y = \begin{pmatrix} \cos(-\theta_x) & 0 & \sin(-\theta_x) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\theta_x) & 0 & \cos(-\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Z = \begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We combine those into the matrix $R' = Z \cdot Y \cdot X$ and change Formula 3.1 by including R' instead of R. With this, we are now able to scan while rotating the phone around the x and y axis.

In a similar way we define t'

$$t = \begin{pmatrix} t_x \\ t_y \\ t_z \\ 0 \end{pmatrix} \quad \rightarrow \quad t' = \begin{pmatrix} t'_x \\ t'_y \\ t'_z \\ 0 \end{pmatrix} = \begin{pmatrix} t_y \\ -t_x \\ t_z \\ 0 \end{pmatrix}$$

and replace t in Formula 3.1 by t'. Now all three translation axis work as expected.

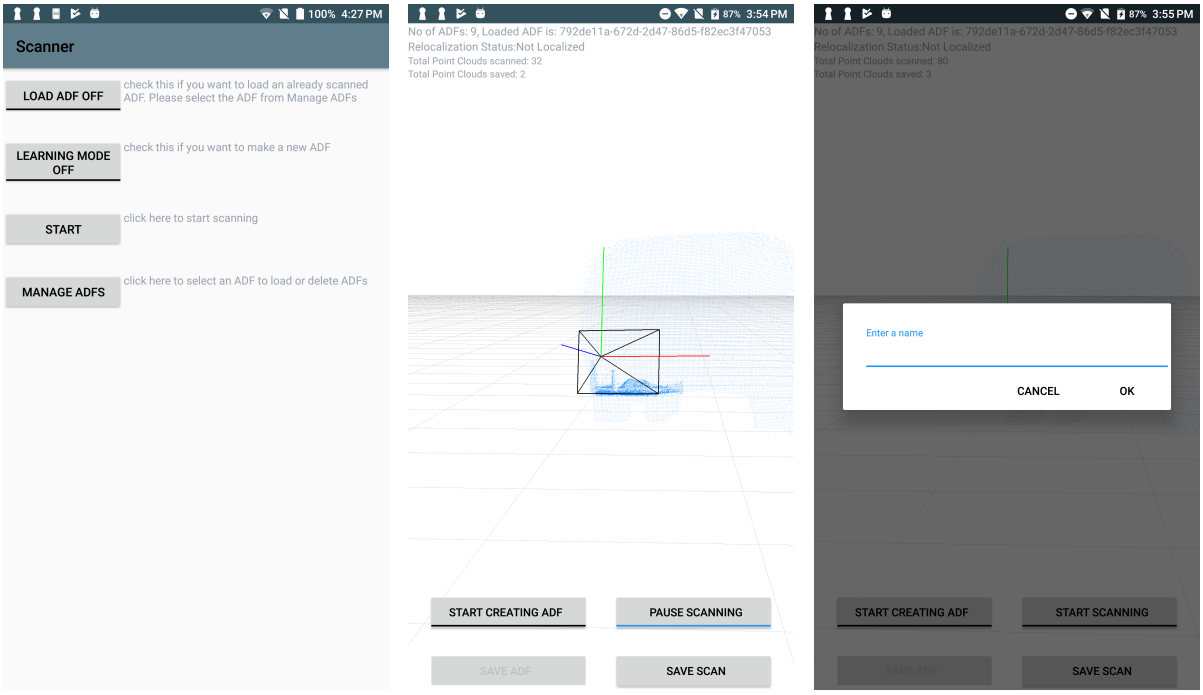
3.4 User Interface

The starting screen, seen in Figure 3.3(a), presents the user with the scanning options. The first button is for switching between loading an ADF or not. When activated, the user has to select the ADF by tapping Manage ADFs. On this list they can also delete and rename them. Area learning mode has to be on for the scanner to be able to create ADFs. When tapping *Start*, the scanning screen appears, Figure 3.3(b). Here the user can create ADFs or scan objects. The buttons to save the ADF or the scan trigger a dialog, Figure 3.3(c), where the user has to enter a file name. The ADF is then saved. When a scan is saved, a file is created in the directory Documents/ScannerFiles and the points are written one by one to the file.

The rest of the process is done on the computer. To get a file from the phone, we can call

```
cd Library/Android/sdk/platform-tools/
./adb pull storage/emulated/0/Documents/ScannerFiles/Name
      ~/Target/Directory/Name.txt
```

We use Polygon File Format (ply) to display point clouds in blender. It is a file format designed to store data from 3D scanners. To transform them into surfaces, we need Point Cloud Data (pcd) files, an extension to ply-files specifically for PCL. They are both pretty similar in storing one point per line, its components separated by spaces. The headers they use differ though. We execute a shell script to prepend headers to the file, depending on what type of point cloud file we want. Examples of the two file formats can be seen in Figure 3.4.



(a) The Start Activity (b) Scanning (c) Saving a scan

Figure 3.3: Screenshots

```
ply
format ascii 1.0
element vertex 33444
property float x
property float y
property float z
end_header
-0.009965 -3.097502 -0.520190
1.998883 0.388584 -0.472801
2.074370 -0.948711 -0.705841
2.048257 -0.807378 -0.255036
...
```

(a) Example .ply

```
VERSION .7
FIELDS x y z
SIZE 4 4 4
TYPE F F F
COUNT 1 1 1
WIDTH 33444
HEIGHT 1
POINTS 33444
DATA ascii
-0.009965 -3.097502 -0.520190
1.998883 0.388584 -0.472801
2.074370 -0.948711 -0.705841
2.048257 -0.807378 -0.255036
...
```

(b) Example .pcd

Figure 3.4: The two file formats used

3.5 Surface Reconstruction

Surface reconstruction is done on the computer. We use PCL [9], which is a library offering lots of algorithms for point cloud and mesh processing. To get a triangulation, we first need the surface normals. They are computed with a `NormalEstimation` object from the PCL. Next we input our point cloud and normals into a `Greedy ProjectionTriangulation`, which returns us a Visualization Toolkit (vtk) file with the polygon mesh. This file format can represent various datasets such as points, grids or polygonal data.

The algorithm we are using is based on the incremental surface growing principle. It selects a starting point, searches through the neighbors to find points that are a best match for forming a triangle, then branches out from that triangle until no more valid triangles can be connected. Then it starts with an unconnected point again. Additionally we use the library for separating the point cloud into planes and non-planes [10]. This is done such that objects standing on a plane like a table can be detached and used as a standalone object.

There are three main variables for this algorithm: the maximum edge length, the maximum number of nearest neighbors that are searched and the distance threshold for plane segmentation. For our application on medium sized objects, we settled on a small edge length of 2.5 cm, trying to exclude outliers from creating singular triangles. Our maximum number of neighbors is a standard 100 and we chose a threshold of 10 cm for the plane segmentation.

Chapter 4

Evaluation

Transformation The process of combining several point clouds into one is the most challenging part of this thesis, mostly because the point clouds do not fit together without an additional translation. To illustrate that, let us take a quadratic room as a recurring example. We are holding the phone in portrait orientation and scanning from the middle of the room to one side, then turning 90 degrees clockwise and scanning again, then doing the same for the third wall. We assume the Phab CS of Figure 3.2(a). We are thus scanning around the x axis.

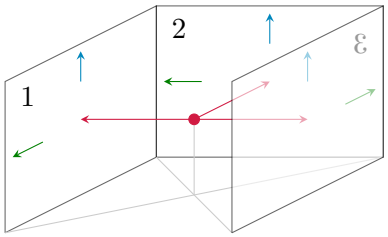


Figure 4.1: Rotation around x axis

It shows as follows when not doing anything to the point clouds we receive from the phone.

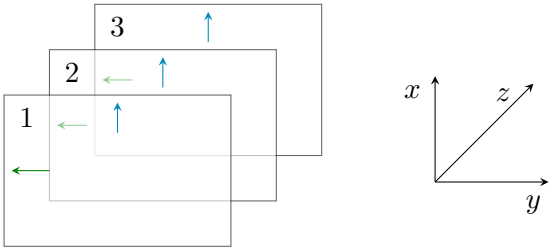


Figure 4.2: x axis, no transformation

This is to be expected because we ignore the movement the phone did while scanning. There is no translation, but there is rotation around the vertical axis of

the phone. Now if we take into account the matrix T from the previous chapter, we get the following result:

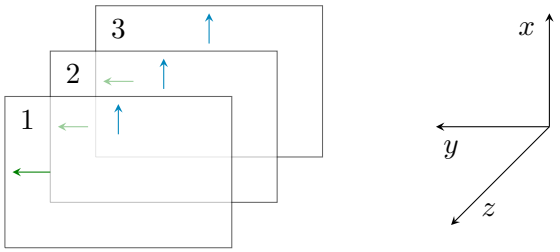


Figure 4.3: x axis, transformation with T

It works as expected and just rotates the points into the Device CS. All we have to do now is to rotate the points with the corresponding rotation matrix $A \cdot R$ given by the API. It indicates the transformation from the Device CS into the Start of Service CS. The scans give the following image:

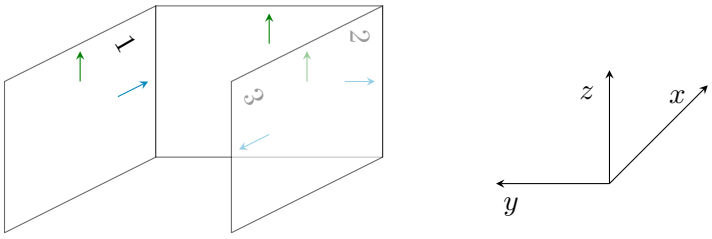


Figure 4.4: x axis, transformation with T and $A \cdot R$

We can see, that this is not what we wanted. We notice though that the points rotate around the y axis instead of the x. So we switch those two as mentioned in Section 3.3 and get:

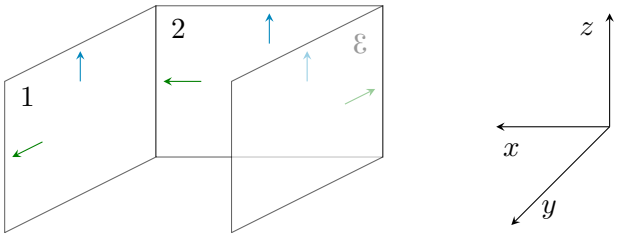


Figure 4.5: x axis, transformation with T and $A \cdot R'$

When scanning around the horizontal axis, the y axis in Figure 3.2(a), we get a similar result.

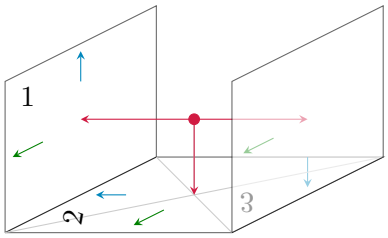


Figure 4.6: Rotation around y axis

Here again, only when doing the described translation we get a satisfactory result.

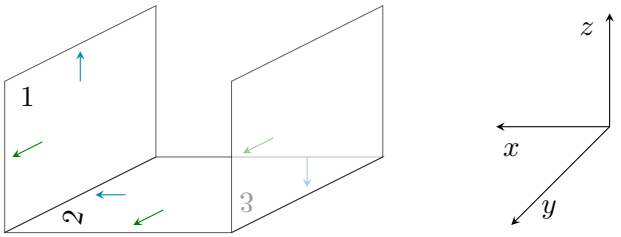


Figure 4.7: y axis, transformation with T and $A \cdot R'$

When rotating around the z axis in Figure 3.2 (a), we do not receive what we expected.

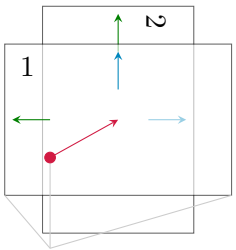


Figure 4.8: Rotation around z axis

When applying the described transformation, we get the situation in Figure 4.9.

We see that it rotates with the right angle, but around the wrong axis. If we wanted to fix this, then the previous two rotations would not work again. We actually found that not being able to rotate the phone around this axis was not too hindering when scanning.

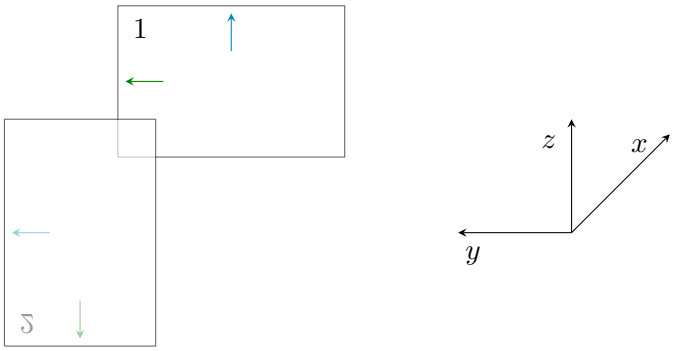


Figure 4.9: z axis, transformation with T and $A \cdot R'$

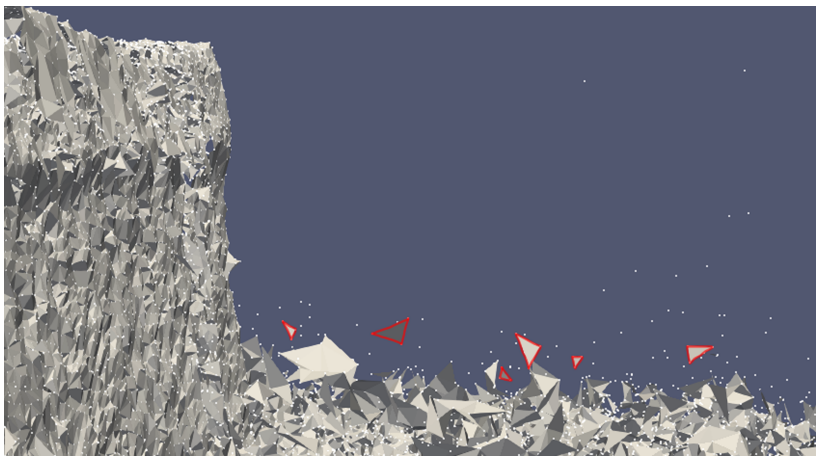
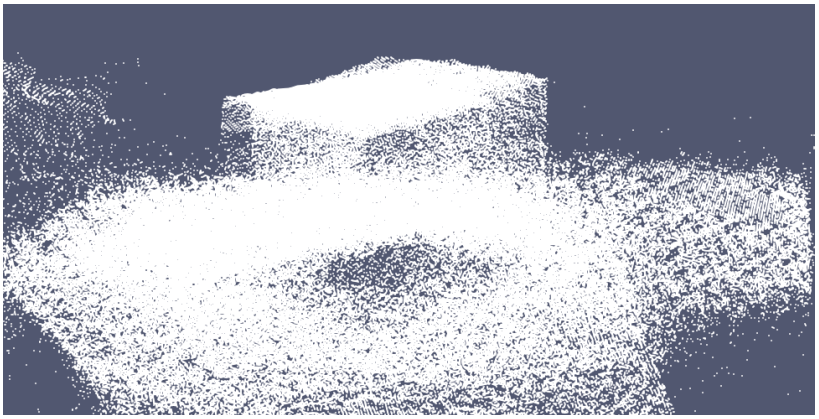


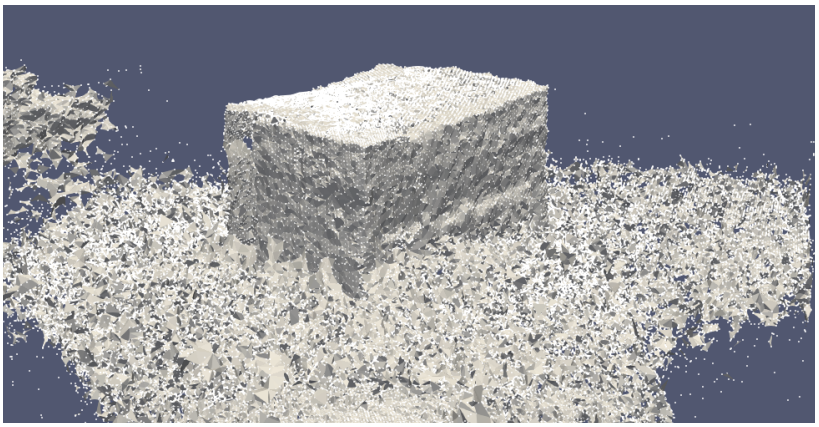
Figure 4.10: Outlier triangles

Surface Reconstruction A problem with surface reconstruction is the existence of outliers in the point cloud. Ignoring the outliers by setting a small edge length worked pretty well, but was not completely successfull. There were still some separate floating triangles, as marked in red in Figure 4.10. Also the sheer amount of points leads to the creation of quite uneven surfaces.

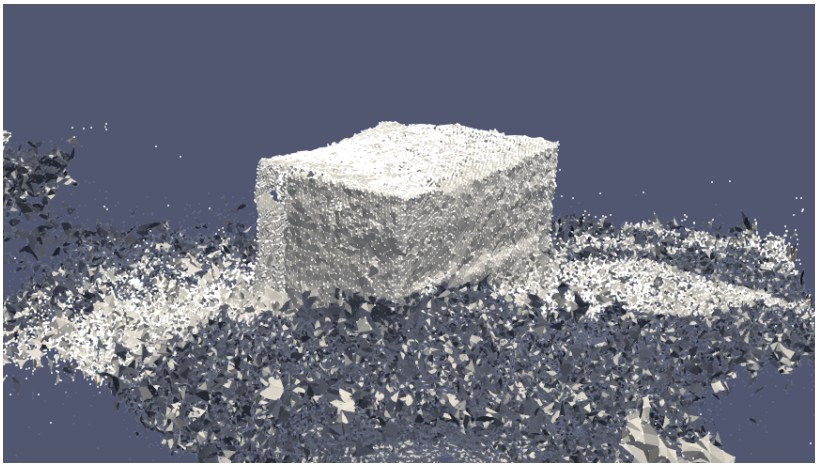
In Figure 4.11 we can see a box scanned with our app. Interestingly, the box itself is triangulated better than the plane that its standing on. Our first thought was that this happened because it was a dark floor and Tango does not work well with dark objects. But when we tried the same object on a lighter floor, we got a similar result. Because of the uneven triangles of the floor, plane segmentation gets difficult. We have to cut away a broad strip to get all points associated to the floor. With that we also cut away some of the object. In Figure 4.11 (c) we tried to find a middleground, using the 10 cm threshold mentionied in section 3.5.



(a) Box Point Cloud



(b) Box Surface



(c) Box Surface with plane cut away

Figure 4.11: Surface Reconstruction, scanned with our app

Memory Leak At first, the screen only shows the currently visible points. We wanted to expand this by showing all points that have been accumulated so far in the current scan. A buffer contains the points that are to be displayed. By adding the current points to the buffer and increasing the buffer size, instead of replacing the points in the buffer with the new ones we should be able to see all points. This process works, but brings an important problem: the memory used for the app is suddenly increasing constantly by about 1MB per ten seconds, even if we are not scanning any new points. There has to be a thread that has a memory leak, but we can not figure out which one or how to stop it. Thus, we only show the current points on the screen.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we created an app that can scan an object, and transform it into a surface model on a computer. We are successfully taking into account ADFs, so that the phone corrects its position as soon as it recognizes where it is. This does not lead to a great change in point cloud accuracy though. The point clouds are loaded onto the computer and can be triangulated. The goal of separating objects from the plane they are standing on was only partially accomplished, small objects often get accidentally cut away as well.

5.2 Future work

There are several ways in which this project could be improved or extended:

- Filter out outliers and simplify the point clouds in general so that plane separation would be more accurate
- Include the information from the color camera to assign a color to every point
- Separate multiple objects that are scanned at the same time
- Fit shapes like spheres and cones to the objects to get surfaces without holes
- Find and fix the bug so that one can rotate the phone in every direction while scanning
- Do the surface reconstruction on the phone instead of on a computer
- Let people move objects in front of the camera instead of moving the camera around the object

Bibliography

[1] Tango — Google Developers: https://developers.google.com/tango/

[2] Klingensmith, M., Dryanovski, I., Srinivasa, S., Xiao, J.: Chisel: Real time large scale 3d reconstruction onboard a mobile device. In: Robotics Science and Systems 2015, Pittsburgh, PA (July 2015)

[3] Marton, Z.C., Rusu, R.B., Beetz, M.: On fast surface reconstruction methods for large and noisy point clouds. In: IEEE. (2009)

[4] Trevor, A., Gedikli, S., Rusu, R., Christensen, H.: Efficient organized point cloud segmentation with connected components. (01 2013) 1–6

[5] Rusu, R.B., Blodow, N., Marton, Z.C., Beetz, M.: Close-range scene segmentation and reconstruction of 3d point cloud maps for mobile manipulation in domestic environments. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. (Oct 2009) 1–6

[6] Dimitrov, A., Golparvar-Fard, M.: Segmentation of building point cloud models including detailed architectural/structural features and mep systems. Automation in Construction **51** (2015) 32 – 45

[7] Tango Developer Overview — Tango — Google Developers: https://developers.google.com/tango/developer-overview

[8] Point set triangulation - Wikipedia: https://en.wikipedia.org/wiki/point_set_triangulation

[9] Rusu, R.B., Cousins, S.: 3D is here: Point Cloud Library (PCL). In: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China (May 9-13 2011)

[10] Holzer, S.: Pcl: Segmentation. In: ICCV. (2011)