TU WIEN Informatics

Efficient User Guidance to Next-Best-View Scan Positions

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Peter Fuchs
Registration Number 12125442

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass. Mag.rer.soc.oec. PhD Stefan Ohrhallinger

Vienna, January 2, 2025

Peter Fuchs

Michael Wimmer

Erklärung zur Verfassung der Arbeit

Peter Fuchs

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 2. Jänner 2025

Peter Fuchs

Kurzfassung

Das Einscannen von Objekten in der Digitalisierung ist eine Aufgabe, die in den nächsten Jahren vermutlich an Priorität gewinnen wird. Unterschiedliche wissenschaftliche Arbeiten zeigen bereits das Porenzial von Augmented Reality (AR) in der Nutzernavigation, daher können einige von den Erkenntnissen vermutlich auch für das Verbessern der Nutzererfahrung in Scan-Prozessen benutzt werden. Diese Arbeit versucht, diesen Prozess mithilfe von AR so effizient wie möglich zu machen. AR wird hierbei dafür benutzt, um den Nutzer so effizient wie möglich zur nächstbesten Scan-Position (der Next Best View (NBV)) zu führen. Diese Punkte werden von einem Algorithmus berechnet und, welcher die bereits gescannten Teile des Objektes als Eingabe bekommt und die berechnete NBV zurückgibt.

Weiters wurden einige Elemente für die AR-Nutzeroberfläche von vorherigen Studien bewertet. Aus ihnen wurde eine Reihe an Guidelines erfasst. So sind für einen möglichst Effizienten Scan-Prozess ein richtungsweisender Indikator, zum Beispiel ein Pfeil, und ein Feedback-Mechanismus, zum Beispiel durch Text-Labels, erforderlich. Weiters erleichtern im Raum positionierte Elemente, wie zum Beispiel kleine Sphären, das Finden der NBV.

Diese Guidelines wurden in einem Proof-of-concept-Projekt implementiert. Für das Projekt wurden das Visual Simultaneous Localization and Mapping (vSLAM)-basierende Real-Time Appearance-Based Mapping (RTAB-Map) gemeinsam mit NBVNet, einer Convolutional Neural Network (CNN)-Implementierung eines NBV-Algorithmus, als grundlegende Technologien verwendet. Für diese Implementierung wurde auch eine Liste an Best-Practices für die Verwendung entwickelt.

Basierend auf der Implementierung wurden wurden die Guidelines noch einmal erweitert. So wurde festgelegt, dass der richtungsweisende Indikator am besten im Raum selbst und nicht am Bildschirm gerendert werden sollte, der Feedback-Mechanismus hingegen nicht im Raum gerendert werden muss. Außerdem hat sich gezeigt, dass die an der NBV positionierte Sphäre zwar hilfreich sein kann, in den meisten Fällen aber nicht benötigt wurde.

Eines der Probleme der kombinierten Nutzung von vSLAM und NBV ist, dass vSLAM alle Punkte der gescannten Umgebung speichert, während NBV den Fokus nur auf das Objekt legen will. Daher wurde ein Viewpoint-intersection-Algorithmus implementiert, welcher dafür sorgt, dass nur das Objekt selbst für die Berechnung der NBV benutzt wird.

Abstract

Scanning objects for digitalization purposes is a task that, due to the ongoing digitalization process, may gain priority over the next few years. Different papers already show the potential of Augmented Reality (AR) within user navigation processes, thus some of these findings can also be used for user guidance within a scanning process. This paper aims to make this process more efficient, using AR as a guiding technology to reach the most efficient scanning points. These points are calculated using a Next Best View (NBV) algorithm, which receives parts of the scanned object as the input and provides the next best scanning position, the NBV, in this model as the output.

Different User Interface (UI) elements were evaluated from preexisting studies and, based on the research, a set of guidelines was developed. The outcome was that, for the most efficient user scanning, a directional indicator, like an arrow, is needed alongside some feedback mechanisms, like text labels. Additionally, positional elements, i.e. a marker implemented as a sphere, help to find the NBV.

These guidelines were implemented in a proof-of-concept application using the Visual Simultaneous Localization and Mapping (vSLAM)-based Real-Time Appearance-Based Mapping (RTAB-Map) and NBVNet, a CNN implementation of NBV computation, as the base technologies. For this application, a set of best practices for usage was developed.

Based on the implementation, the guidelines were further enhanced; the directional indicator should be rendered in the 3D world and not on the screen, while the text labels should be rendered on the screen. While the markers were not needed for finding the NBV most of the time, they were still helpful sometimes.

One of the problems of using vSLAM and NBV in combination is that vSLAM saves every point of the scanned environment, while NBV only works with the object itself. Thus, a viewpoint intersection algorithm was implemented to make the NBV implementation focus on the object and not the environment.

Contents

CHAPTER 1

Introduction

With digitalization gaining importance in the last few years, the digitalization of 3D objects has become an increasingly important problem. Thus different technologies to optimize this scanning process have been created. This includes Next Best View (NBV), which aims to provide the next best position for the user to scan as many new points of an object while still being able to map them to already scanned points. This is achieved by following algorithms considering the object's already scanned parts and returning the position for the most efficient scan. While NBV algorithms have existed for a while and have switched to AI-trained models in recent years, the display method of these next best positions has not been researched yet.

Augmented Reality (AR) has also been on the rise recently. Recent developments in Snapchat's AR-glasses and Apple's Vision Pro have also pushed this technology in the commercial direction, aiming to become a staple technology that may be used in people's daily lives. Therefore, multiple areas of development have opened to the usage of AR, including user navigation.

When breaking it down to the basics, guiding a user from their current position to the position of the Next Best View is just a navigational task within a small space. Therefore, some of the findings of related studies about user navigation using AR can be repurposed to generate both guidelines and an implementation of a User Interface (UI) for this task. The guidelines are necessary since AR is used in vastly differing environments thus having a set of rules that can easily be followed helps generate a consistent and working UI.

The end goal of this project is to build an environment in which all these technologies are joined together, enabling the user to scan objects as efficiently as possible. It will use a camera system to track the user's position and the already scanned points in the environment. These points are then run through the NBV model to generate the next scan location. This location is then sent to the AR system that guides the user from his current to this new location.

CHAPTER 2

Related Work

This section will discuss the theoretical background behind the different technologies. It shows different approaches to the problems of efficient scanning and user guidance.

2.1 Next Best View

The problem of searching for the most efficient point for the next best view of a partially scanned object has been around for many years with the first approaches being presented by Conolly in 1985 [6]. In his work, he proposed two algorithms which aimed to recreate 3D objects from depth images using an octree structure [6]. These algorithmic approaches further developed over time covering multiple algorithmic types and input datatypes.

In 2016, Isler et al. proposed the first learning-based Machine Learning (ML) approach to the problem of NBVs [7]. While Isler et al. based their work on robotic vision and their potential information gain for different candidate views, which makes the work not applicable to this problem, it was more effective and flexible than the algorithmic approaches [7].

Zeng et al. approached the problem of NBV by having a Deep Neural Network (DNN), named PC-NBV, propose a series of different next-best views and choosing the most effective series by its coverage [32]. Since the problem of evaluating the NBV is NP-complete, a utility function f_{util} is used to select the most effective one. This utility function is the DNN, PC-NBV [32]. As the name suggests, this model works with point clouds as the input data for training and NBV prediction.

In their work, Mendoza et al. use a 3D-Convolutional Neural Network (CNN) to select the best view from a list of candidate views [15]. In contrast to Zeng et al., however, these candidate views are based on a predefined list and not on the input data [15]. While this makes the work usable for most scanning problems, it might struggle with flexibility and multi-object scenes.

Model name	Input datatype	Model type
Isler et al. (2016) [7]	Volumetric	DNN
Zeng et al. (2020) [32]	Point Cloud	DNN
Mendoza et al. (2020) [15]	3D probabilistic grid	3D-CNN
Wang et al. (2024) [31]	Point Cloud	DQN

Table 2.1: Comparison of NBV models

Wang et al. base their work on Zeng et al.'s PC-NBV but use a reinforcement learning-based approach (Q-deep Network (DQN)) instead of a model-based one (DNN) [31]. This model improved accuracy and prediction speed [31], thus making it a more effective version of PC-NBV.

Since learning-based approaches proved more effective in generating the NBV, this paper focused only on them. Table 2.1 compares the different NBV models with their input datatype and model type. The models that were preferred for this project, are highlighted in bold. They were mainly chosen by model speed (according to [31, 15]), input type and ease of implementation, which is further discussed in 3.1.1.

2.2 Visual Simultaneous Localization and Mapping

Visual Simultaneous Localization and Mapping (vSLAM) describes the process of simultaneously localizing the user's position and generating a 3D map of the user's surroundings using visual input data, such as (depth) images [26]. There are multiple approaches to vSLAM algorithms: feature-based methods (where features are extracted from the images and inserted into the map), direct methods (transforming images directly to maps without extracting features beforehand) and RGB-D vSLAM (uses depth images to create more accurate maps) [26]. RGB-D vSLAM is the newest of these algorithms as it increased in popularity with the release of cheap and small RGB-D cameras like Microsoft's Azure Kinect [26]. Since they already provide 3D information in real-time, they are the most popular form of vSLAM algorithms nowadays.

Different RGB-D vSLAM algorithms include Newcombe et al.'s KinectFusion [17] that was also ported to mobile devices successfully [9]. It does, however, not support global optimization or loop closures making the algorithm more prone to errors than newer algorithms [26].

Salas-Moreno et al. proposed SLAM++, an algorithm that uses online resources to register 3D objects [24].

Labbé et al. proposed Real-Time Appearance-Based Mapping (RTAB-Map) [11], an algorithm that uses loop closures and depth images to build a 3D map [22].

2.3 Augmented Reality user interface

Multiple studies have related to UIs within AR. In the scope of this paper, they will be split into studies about conventional AR UIs and studies that propose UIs for navigation using AR.

2.3.1 Conventional AR-UIs

In their works, Cao et al. summarize the current state of the art for mobile AR user interfaces and practices, going into more detail about different requirements for AR UIs [5].

Arifin et al. propose metrics that allow the evaluation of user experience in AR UIs for its objective measurements, such as effectiveness or usability. While there are still metrics that cannot be measured objectively (e.g. user preference or attractiveness), the UX can be judged based on the following categories: performance (how well the user performs on the task), issues (how many problems while performing the task), user perceptions (ease of use, usefulness) and subjective criteria (i.e. user emotions, attractiveness) [1].

Sandor et al. compare different state-of-the-art AR interfaces over multiple fields. While they mainly focus on general UIs, they also look at navigation interfaces such as CAR, a navigation interface for driving [25].

2.3.2 AR navigation UIs

Lee et al. studied user preferences for UI elements in navigation spaces. In their study, they differentiated between multiple environments, namely outdoor spaces and large and small indoor spaces and provided results based on the surroundings [12]. The proposed elements can be seen in Figure 2.1.

Figure 2.1: Proposed UI elements for user navigation in AR. [12]

Ventä-Olkkonen et al. looked at the differences between navigation in real cities versus augmentations of that city. They studied user preferences regarding watching a videotape of the navigation instructions, the instructions being overlaid over the live camera feed, and the whole environment being approximated using digitalization [29].

In his master thesis, Mang proposes a system for vision-based indoor navigation. He further conducted two studies; first, he proposed three instruction modes: fully automatic, decision-point, and manual. These modes indicate how often users want to receive updates on the AR UI. The second study focuses on different UI elements and their effectiveness in the navigation process [13].

Joshi et al. and Martin et al. propose a AR UI for indoor navigation. While Joshi et al. focus more on the technical implementation of the application, proposing different technologies for creating a working application [8], Martin et al.'s work implements a navigation application within an indoor environment [14].

2.4 Guildelines for AR scanning UIs

Guiding a user to a specific scanning location around an object can generally be described as a navigation process in a small space. In the context of scanning, this process requires that the destination point is found as accurately as possible, and, since the AR interface is the only interaction point with the user, two preconditions must be met to make the User Experience (UX) as smooth as possible: The UI must follow some guidelines to be consistent and efficient and the user must know his goal is to scan an object as efficiently as possible. The second precondition is due to the limitations of RTAB-Map, as it cannot distinguish between the 3D object and the environment, thus pointing the camera away from the object may result in less performant scanning. This also generates the main goal of the interface: The user should be able to move from his current position to the current NBV without pointing the camera away from the 3D object. This section will summarise the solutions to this problem described by the different navigation user interfaces described in section 2.3 and the most relevant findings. The implementation within the RTAB-Map-framework is described in section 3.6.

2.4.1 Requirements for AR UIs

When it comes to general requirements for AR UIs, Cao et al. propose a set of requirements that must be fulfilled to display the information efficiently: Information Filtering and Clustering (filter and group information so only the important information is shown not to overstimulate the user), Occlusion Representation and Depth Cues (giving depth information to UI elements to help the user understand the correct three-dimensional order of the elements), Illumination Estimation (estimating the illumination of UI elements so they do not feel out of place), Registration Error Adaptation (the AR system should be able to adapt to slightly misplaced graphical components) and Adaptive Content Placement (informative UI elements, like text labels, must be drawn on the visible parts of the corresponding components) [5].

2.4.2 Navigation UIs in AR

While navigation in a closed space differs from navigation in a wider open space, there can still be some findings from comparing and finding similarities between the different navigation UIs. For example, as Lee et al. mention, arrows are some of the most important elements of navigational UIs [12]. This is also established in Mang's proposed UI, where directional navigation plays an important role, as well as in Joshi et al. and Martin et al.'s UIs, which both also use arrows as the main indicator [8, 14], which can also be seen in Figure 2.2.

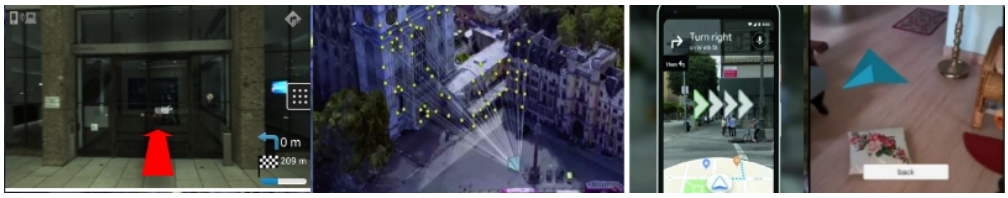

Figure 2.2: Arrow being used as the main indicator by different UIs [13, 8, 14]

According to Lee et al., other popular UI components for navigation include callouts [12], which were also the preferred navigation component in Ventä-Olkkonen et al.'s work [29], as well as desaturation [12], which was not directly implemented in any of the mentioned works. However, text labels, which are 2D implementations of callouts, are mentioned in multiple studies, like Martin et al.'s navigation UI [14]. They can either be used as feedback elements when successfully finishing a task but also as general information elements [14] as they can also be used in callouts [12, 29] (see Figure 2.3).

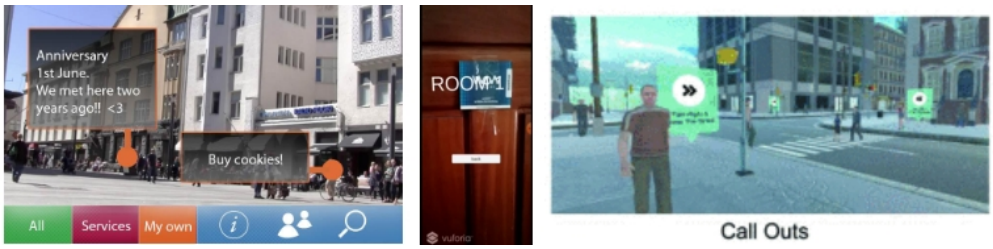

Figure 2.3: Text labels being used in AR UIs [29, 14, 12]

Finally, when reaching the destination, multiple ways of displaying the final point of the navigation were used. Mang et al. marked the destination with a specific finishing UI element [13], while Martin et al. displayed a success message when the destination was reached [14] and Lee et al. proposed other visualization techniques like desaturating every part of the image except for the destination point [12].

2.4.3 AR navigation in small spaces

Since most papers mainly focused on general navigation, these findings cannot directly be translated into small spaces. However, some of the proposed UI elements are more

important for navigating small spaces than others. For example, having a marker at the final destination (at all times) is more important for navigation in small spaces than within large spaces, like streets or large indoor spaces, as the destination is visible at most times. This marker can and should also indicate how close the user is to reaching the destination. This can be done by using text labels to display the absolute distance from the destination or by adapting the marker's size with the absolute distance (see Figure 3.13).

Figure 2.4: Different marker sizes with different distances

Additionally, some elements, like arrows, serve a different purpose in small spaces than in large ones. For large spaces, the main goal of UI elements is to guide the user to different checkpoints and instruct them to further navigate to the desired endpoints [12]. This is not only unnecessary in small spaces, as the destination point is visible at most times, but might also be counterintuitive, as, for example, arrows are difficult to track in small environments [12]. Therefore, UI elements should aim to accomplish a directional input to the user, as presented by Martin et al. [14], since overusing them might lead to a cluttered environment [5]. Looking at the UI elements proposed by Lee et al. (see Figure 2.1), only a number of these elements can be used as indicators for directional change. These can be put in two categories: instructive indicators that propose directional change (i.e. arrows, callouts, glowing paths) and informative indicators that propose some need for change (i.e. desaturated image when the destination is not visible). While informative indicators might be valid for certain environments, for example when UI elements can not be used properly, they might lead to confusion, since they do not indicate the necessary change (i.e. in which direction to turn).

In contrast to the final destination, instructive indicators may be placed in both 2D and 3D space. They do not necessarily need to give depth cues since the destination marker already gives the necessary depth cues for the destination. Thus, the indicator's main job is to provide feedback about needed changes in rotation and, in contrast to navigation in bigger spaces, not necessarily movement.

Additionally, the indicators must be able to give information to the user all the time. The user should never be in a situation where he does not know how they can reach the destination or how he should proceed with the navigation process. This means that at any moment, an indicator, ideally instructive, or the destination marker should be visible

on the screen. This should, however, not lead to an overly cluttered screen. Having too much information here has a similar effect as having no information [1]. The directional indicator may also be removed when approaching the destination since this might lead to clutter and overlapping elements.

When reaching the destination, a success message should be given to the user, giving them feedback about the task. This message can either be shown as a text label, stating that the destination was reached (see Figure 2.5), by providing a visual cue, such as lighting up the screen in green colour, or a combination of them.

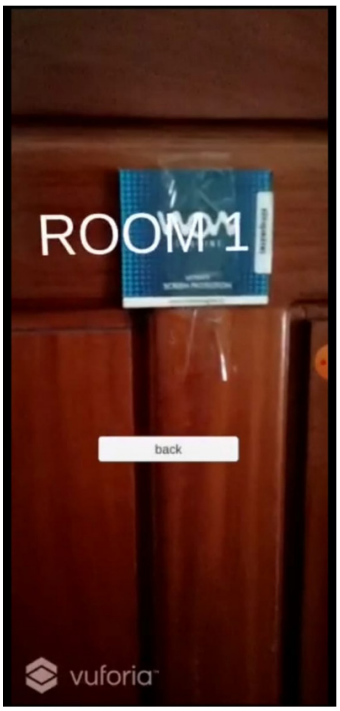

Figure 2.5: Text label that shows the final destination [14]

Finally, some of the mentioned requirements for AR UIs must be given extra priority. Being in a small room carries the potential for an easily cluttered environment [5] thus filtering out unnecessary information and grouping overlaying information becomes exceedingly important. Furthermore, as the directional input is likely to overlap with the marker of the final destination, depth cues for both elements are also of high importance. Since the directional indicator will always point to the marker, it should always be placed on top of it, as partial overlap between the elements might lead to confusion.

2.4.4 AR navigation while scanning

However, AR navigation has some properties that may inflict further problems when it comes to the specific task of scanning an object using vSLAM in combination with NBV. The main issue is the constant scanning of the environment, which, when focusing

on the marker for reaching the next destination, leads to the surroundings gaining in importance (i.e. being scanned more than the object) and thus potentially confusing the NBV algorithm. This means that the goal of the UI is to lead the user to a specific destination while trying to keep the focus on a different object.

While there are multiple solutions to this problem, the one implemented in this project aims to use visual indicators, like bounding boxes, to give the user feedback about the current scanning status, specifically the currently scanned object. A detailed implementation description can be found in section 3.6.5.

CHAPTER 3

Method

The following chapter describes the different parts of the scanning process - in particular, the necessary components: RTAB-Map as the scanning and visualization tool and AR provider, the AR UI for the user guidance and NBVNet as the NBV computation tool. Figure 3.1 visualizes the data flow between these components.

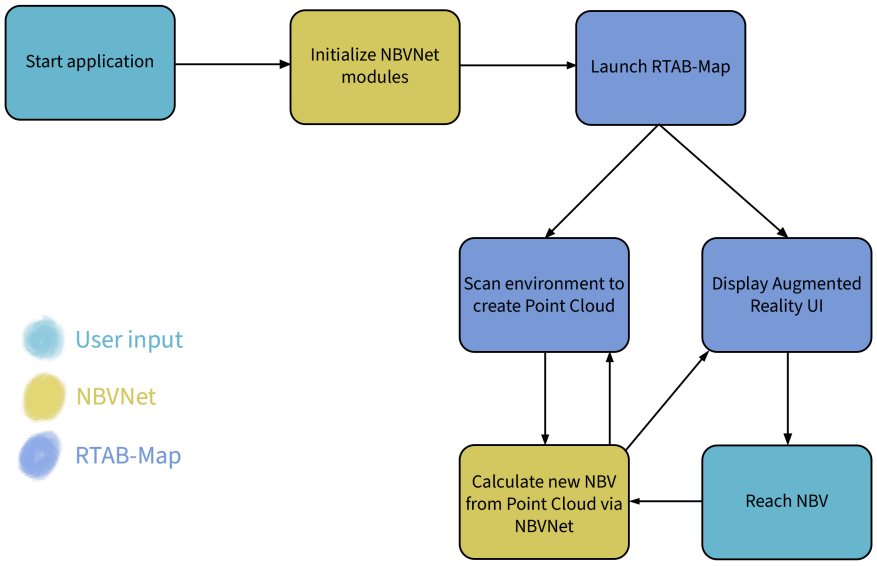

Figure 3.1: Overview of the different application components and the information flow between them

3.1 Prerequisites

This project was developed on Ubuntu 24.04 using the Azure Kinect DK as the camera input and RTAB-Map [22] on the release of November 2023.

This section describes all the prerequisites that need to be met for the code to compile successfully. This includes the necessary SDKs for the camera system, the NBV model and the AR system. The source code can be found in [27].

3.1.1 Next Best View

In this project, the first instinct was to work with either PC-NBV or RL-NBV due to point clouds being the input data and their faster processing speed compared to the other models [31]. They did, however, require additional training of their model as they do not provide a pre-trained AI model. Additionally, PC-NBV and RL-NBV base their work on CUDA meaning that the code must be compiled before it can be called from the project.

For these reasons, Mendoza et al.'s NBV-Net was used since they provide an already pre-trained model which can easily be loaded using torch [28]. The NBV is also calculated using only Python, meaning that no additional CUDA-files must be compiled. NBV-Net does, however, also provide CUDA compatibility, so it performs better on a GPU with CUDA enabled.

3.1.2 Azure Kinect DK

The Azure Kinect DK needs the Azure Kinect Sensor Software Development Kit (SDK) [2] to work. This can be installed using the installation guide provided by the developers [4]. For this project, the SDK was additionally built on the system itself as it was necessary to debug some of the Kinect's internal data. This process was done using the building guide [3]. It was, however, necessary to switch to OpenSSL v1 as building the SDK only works in this version when using Ubuntu 24. Additional changes include including standard C++ libraries in the code (<limits> in *k4adepthpixelcolorizer.h*, <string> in *k4aaudiochanneldatagraph.h*, <cstring> in *k4amicrophonelistener.h* and <string> in *perfcounter.h*) as well as deleting some of the non-working unit tests (*depth_ut.cpp* in the DepthTests-folder, *depthmcu_ut.cpp* of the general unit tests folder as well as the test for object usage after being freed in *handle_ut.cpp* : $105 - 118$). It was also important that the *CMakeLists.txt*-files were adjusted accordingly.

3.1.3 RTAB-Map

All the corresponding packages must be installed to work with RTAB-Map. This includes OpenCV, g2o and GTSAM [23]. This project was developed using RTAB-Map's November 2023 release. It should, however, also work with the newer December 2024 release as this mainly added new demos and examples [22].

3.2 Scanning process

The implemented scanning process can be split into two parts:

1. Initialization and object detection
2. Scanning and NBV calculation

The main goal of the initialization and object detection process is to initialize the application and identify the object the user wants to scan. First, the packages necessary for NBV computation are loaded into the program and the Python script is prepared for execution (as described in section 3.3). After this step, a seven-second cooldown is started, during which the user can position the camera correctly. This duration was chosen through testing and was the most efficient, providing enough time to position the camera and not making the user wait too long to start the scanning process. Afterwards, the object detection algorithm (see section 3.5) is started. Here, the user is guided with a simple text and progress bar UI that shows the current status of the object centre detection process.

After the object centre is computed, the first NBV is calculated based on the area scanned in the detection process. When the result is received in the application, a marker is placed on the NBV position and a 3D arrow is rendered in the 3D space which points to the destination. Even though the arrow is placed in the 3D world, one end is locked on the camera, which allows the placement of a 2D label displaying the distance to the NBV position in meters.

Once the user gets within 5cm of the calculated NBV, the newly scanned point cloud is run through the NBV algorithm computing a new NBV. This process is repeated indefinitely or until the user terminates the program. The point cloud last used for NBV calculation can then be accessed in a *.ply* file to make the result of the scanning process available.

3.3 Next Best View implementation

This section describes the steps to implement the Next Best View (NBV) computation. This includes a C++ wrapper for calling the necessary Python function and a more detailed explanation of the algorithm for the NBV calculation.

3.3.1 NBV interface

After ensuring the correct functionality of the NBV model, the interface between RTAB-Map and the AI was defined. Since the RTAB-Map code runs on C++ and the NBV model uses Python's torch, the interface needed to call the Python code and retrieve the result afterwards. For this, Python's embedding API [20] was used from which the

corresponding NBV methods were called. One of the important aspects here was that the Python modules needed to be loaded on program startup as loading the modules every time the NBV is calculated would add extra execution time to the calculation. For this, the C++ object $NBVExecutor$ loads the prediction module (*predict.py*) in the constructor and stores the function *predict*(), which runs the NBV prediction algorithm, as an attribute. This function can be called anytime the NBV needs to be calculated. One important aspect here is that the environment variable $PYTHONPATH$ includes the project folder to find the *predict*-module and also points to a Python environment where the necessary modules (i.e. torch, numpy and open3d) are installed (e.g. venv).

3.3.2 NBV calculation

The NBV calculation is based on NBV-Net's $nbv\_inference$ Jupyter-workbook [16]. First, the point cloud is transformed into a 3D probabilistic grid following algorithm 3.1. This algorithm maps the point cloud into a 32x32x32 grid, where each cell contains the number of points. This number is then normalized between 0.3 and 0.7 (meaning that the cell with the most points contains 0.7 and the cell with the least points contains 0.3). These values are taken from the examples provided by Mendoza et al. [16].

Algorithm 3.1: Point cloud to 3D probabilistic grid

Input: Point cloud pc and scalar $grid\_size = 32$
Output: 3D probabilistic $grid$ with size $32x32x32$

$grid\_spacing = [(max(pc) - min(pc))/grid\_size]$;
for $point$ **in** pc **do**
    $idx = [(point - min(pc))/grid\_spacing]$;
    $grid[idx] += 1$;
end
$grid = grid/sum(grid)$;
$grid = (grid - min(grid)/max(grid)$;
$grid = grid * 0.4 + 0.3$;
return *grid;*

This grid is then run through the AI model which returns a list of possible NBVs each containing a value of how well it fits as the Next Best View. The algorithm selects the best of these views, calculates its position in the point cloud coordinate system and returns this NBV to the program (algorithm 3.2). The final value is multiplied by 64 since it first needs to be readjusted from the centre of the grid size (times 16) and the voxelmap has an approximate size of 0.25, which the output needs to be divided by $(16/0.25 = 64)$ [15].

Algorithm 3.2: Calculate NBV from grid

Input: 3D probabilistic $grid$ with size $32x32x32$ and point cloud pc
Output: Point nbv in point cloud coordinate system

```
grid_spacing = [(max(pc) − min(pc))/32];
net = torch.load(weights);
output = net.forward(grid);
best_nbv = output[val = max(output)];
return best_nbv ∗ 64 ∗ grid_spacing + min(pc);
```

3.4 Point cloud acquisition

RTAB-Map implements a vSLAM algorithm that tracks the current camera position while simultaneously enabling AR. This works because RTAB-Map uses the camera feed to build a 3D map, where digital elements can be inserted using the PCLVisualizer [18]. This map is then stored as a point cloud view, which can be displayed as a 2D image. The project was built on the RGB-D usage example [21].

This project runs two point cloud viewers simultaneously: one scans the environment continually, always keeping the 3D map up to date, and one displays the latest point cloud used in the NBV algorithm to show the already scanned parts of the object. The first viewer is not shown to the user as it runs in the background to collect data. The second is put on top of the camera input using a QStackedLayout to create the illusion that the 3D map is built on top of the camera input (see Figures 3.2 and 3.3). The AR elements are then inserted into the displayed viewer, being shown on top of the 3D map (see Figure 3.3).

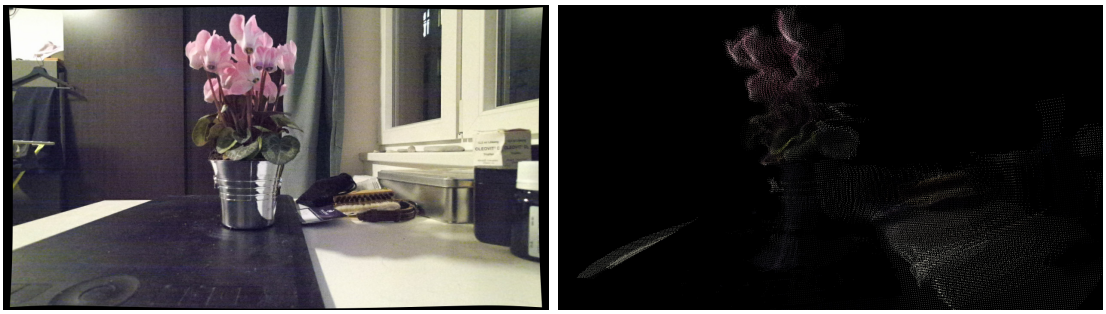

Figure 3.2: Camera input without point cloud and corresponding point cloud

Since RTAB-Map already saves the map in the form of point clouds, they can be extracted from RTAB-Map using its *util3d*-package and exported to a *ply*-file using PointCloud-Library (PCL)'s $savePLYFile$ [19]. This procedure is executed every time the user gets within 5cm of the next best view to generate a new point cloud for the next prediction of the NBV algorithm. This value was chosen through testing.

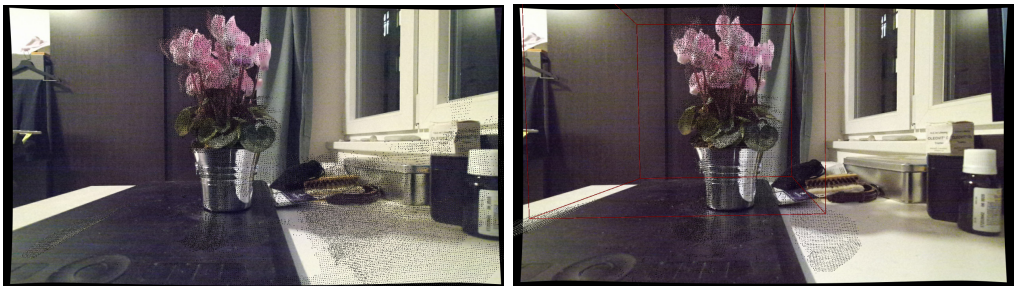

Figure 3.3: Camera input with point cloud and AR elements projected on top of it

3.5 Object centre computation

Since proper functionality of the NBV algorithm is necessary for the scanning process, there has to be some distinction between the scanned object and the background, as the NBV algorithm would otherwise value features from both equally. Thus, an algorithm was implemented that computes the centre point of the scanned object by intersecting the camera viewpoints. This task is done in the initialization part of the scanning process before the NBV positions are calculated. The one criterion for this algorithm is that the camera must be moved around the object on a sphere while always pointing at the object. It then stores the camera angles and positions and calculates the intersection point following algorithm 3.3. The algorithm first stores the camera positions P and rotations (the viewpoint directions D) in a set of 3D vectors (*set P* and *set D*), where the rotation vectors are normalized. The best intersection point x of this set of vectors can be calculated by taking the sum of their squared distances and calculating the zero point of its derivative:

$\Delta(\sum \vec{u_i} \cdot \vec{u_i}) = \vec{0}$, where $\vec{u_i} = \vec{D_i} \times (\vec{P_i} - \vec{x})$, $\vec{D_i} \in D, \vec{P_i} \in P$

According to [30], this function can be expanded and yielded into a 3x3 matrix multiplication:

$$M * \vec{x} = \vec{b}$$

with $\vec{b} = \sum(\vec{D_i}(\vec{P_i} \cdot \vec{D_i}) - \vec{P_i}(\vec{D_i} \cdot \vec{D_i})\ ,\ \vec{D_i} \in D, \vec{P_i} \in P$
and $M_k = \sum(D_{ik}\vec{D_i} - (\vec{D_i} \cdot \vec{D_i})\vec{e_k})^T\ ,\ \vec{D_i} \in D, \vec{P_i} \in P$, where M_k is the kth row of the matrix and $\vec{e_k}$ is the respective unit basis vector of the kth row [30]. This equation is solved with a linear equation solver provided by [30].

This is done multiple times throughout the spherical camera movement around the object. After enough camera viewpoints are considered (for this project it was 10 points with a minimum angular difference of 6.75°) the calculated point is interpreted as the centre point of the scanned object. This means that all points within a certain distance from the computed centre (i.e. 3/4ths of the distance from the centre point to the original camera position) are considered for the NBV calculation. Figure 3.4 visualizes this process.

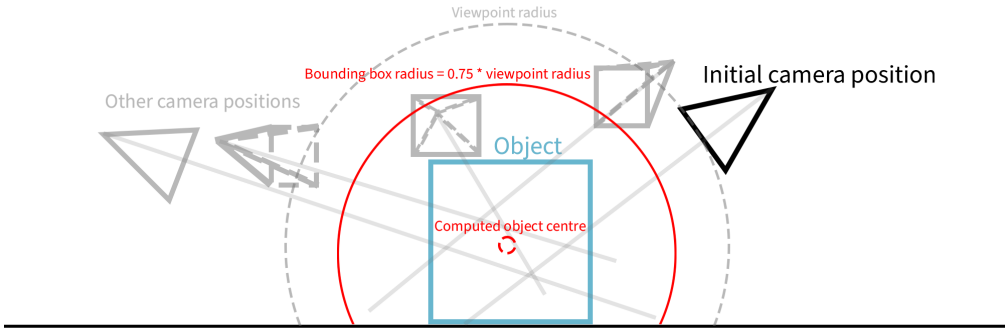

Figure 3.4: Process visualization of bounding box calculation

3.6 Augmented Reality interface implementation

The UI elements described in the section above were implemented using the PCLVisualizer-and, if an UI element has to be built a custom, the Visualization Toolkit (VTK)-libraries. Since section 2.4 already proposed some guidelines and UI elements, this section only covers their graphical implementation. Section 4.2 further compares the different graphical approaches regarding their strengths and weaknesses within the navigational task.

3.6.1 Arrows

Since arrows were found to be the most effective tool in user navigation [12] they were also the primary focus of this study. The following approaches were chosen: 2D arrow in 2D space, 2D arrow in 3D space and 3D arrow in 3D space (see Figure 3.5).

The 2D arrows are built from a 2D rectangle combined with a 2D rectangular triangle, where both are rotated in the direction of the destination. The rectangle and the triangle are assumed to be centred around the origin, then rotated and finally translated to the correct position. This resolves to the following formula for each point:

$$(x * cos\theta - y * sin\theta + t_x, x * sin\theta + y * cos\theta + t_y)$$

where x and y are the coordinates of the point, θ is the angle of rotation and t_x and t_y correspond to the needed translations within the space. Figures 3.6, 3.7 and 3.8 present this process visually.

The 3D arrow on the other hand is a combination of a cylinder and a cone (Figure 3.9). The element generally works in a way where both the starting point of the arrow and the direction of the arrow are provided by 3D vectors. The length of the cylinder is equal to 5/8ths of the length of the directional vector while the cone takes up the remaining 3/8ths. This value was chosen by testing different arrowhead lengths, where the desired outcome was showing direction and distortion change with arrow length. Since the radius

Algorithm 3.3: Viewpoint intersection calculation for object centre detection[30]

Input: Viewport positions vps which contain x, y, z-coordinates and $roll, pitch, yaw$-angles
Output: Intersection point p as $Vector3$

$n = size(vps)$;
$points = Vector3[n]$;
$directions = Vector3[n]$;
for $cameraPosition$ **in** vps **do**
  $points[i] = cameraPosition[x, y, z]$;
  $pitch, yaw = cameraPostion[pitch, yaw]$;
  $directions[i] = [cos(yaw) * cos(pitch), sin(yaw) * cos(pitch), sin(pitch)]$;
  $directions[i] = normalize(directions[i])$;
end
$m = Matrix[3, 3],\ b = Vector3(),\ p = Vector3()$;
for $i = 0; i < n; i++$ **do**
  $d^2 = dot(directions[i], directions[i]), dp = dot(directions[i], points[i])$;
  for $k = 0; k < 3; k++$ **do**
    $m[k] += directions[i][k] * directions[i]$;
    $m[k][k] -= d^2$;
    $b[k] += directions[i][k] * dp - points[i][k] * d^2$
  end
end
return $solve(m * p = b)$ **for** p;

of the cone's base is set by the angle from the tip [18], this angle is calculated by the *arcus tangens* of the radius divided by the cone's length.

While the direction of the arrow is always provided by the camera's current position and the position of the final destination, its length is limited by the camera's field of view because the user must always know in which direction the arrow is pointing. Thus, having parts of the arrow outside the field of view may lead to confusion (see Figure 3.11).

The 2D arrow in 3D space uses the implementation of the *PCLVisualizer*-library.

3.6.2 Labels

The main purpose of a label is to provide the user with more information, thus an important property is the label's positioning. For this project, one of the main use cases for labels was displaying the distance to the NBV. The different versions tested in this study were: The label positioned within the arrow, the label placed at the destination and the label positioned on the screen to make it always visible (Figure 3.10). While the VTK already provides small borders for the elements, some extra background elements,

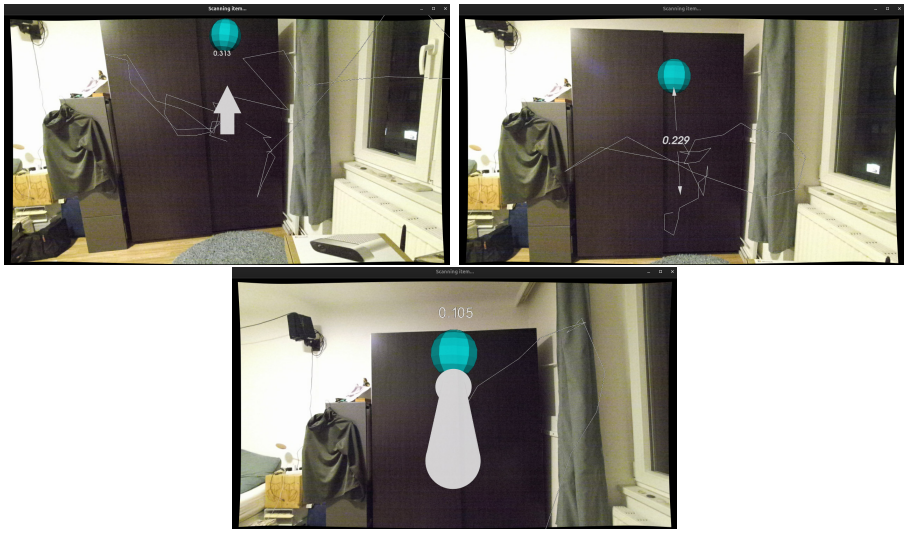

Figure 3.5: UI design implementation for the 2D/3D arrow in 2D/3D space respectively

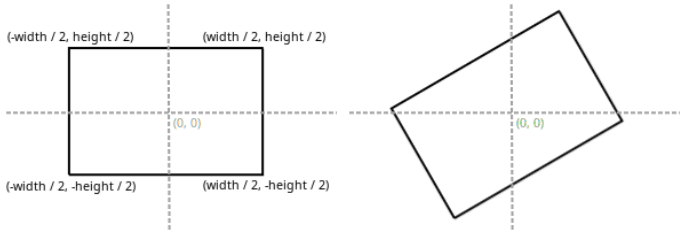

Figure 3.6: Rotation of the rectangle

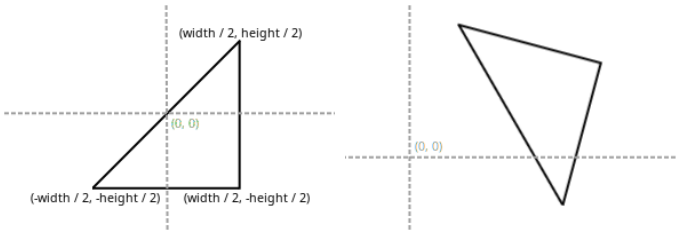

Figure 3.7: Rotation and translation of the triangle

like shadow texts or rectangles, should be added for increased visibility (Figure 3.11).

3.6.3 Markers

Markers are essential for pointing out specific positions in the environment. Since the destination of the NBV is not known to the user but calculated within the system, displaying this point to the user is necessary for the system's functionality. It is, however, also important that this marker is visible from all distances, thus this study mainly focused on marker size changing with distance.

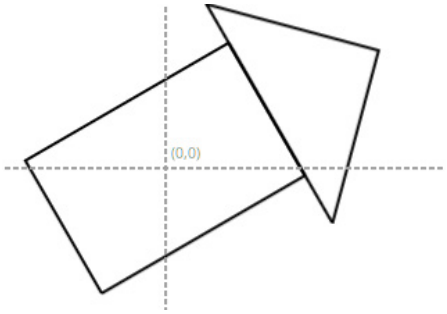

Figure 3.8: Combination of rectangle and triangle

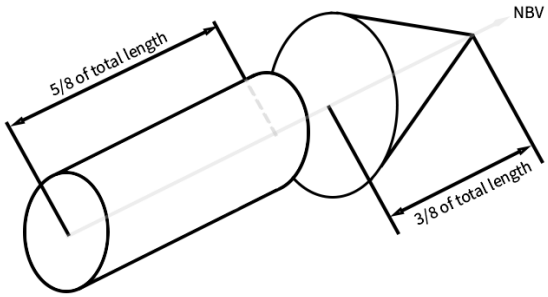

Figure 3.9: Combination of cylinder and cone to create the 3D arrow

Figure 3.12 shows the marker placed into the 3D world with a constant size, while Figure 3.13 shows the marker when its size is calculated using the following formula:

$$size = (distance * scaling) + size_{init}/2$$

with $size_{init} < 0.1$ (in this case 0.02), $0 < distance < 10$ (mostly between 0.01 and 2) and $scaling = 0.03$.

3.6.4 Scanned area

Specifically when scanning an object, telling the user which parts of the object have already been scanned helps to understand the current completion status. This was achieved with RTAB-Map's point cloud implementation, a 3D representation of the current map. This was laid on top of the images received by the camera to display what has already been scanned and what has not (see Figure 3.14).

3.6.5 Bounding Boxes

Similar to the scanned areas, the bounding box is a feedback mechanism that aims to provide the user with information about the current status of the scanning process. In

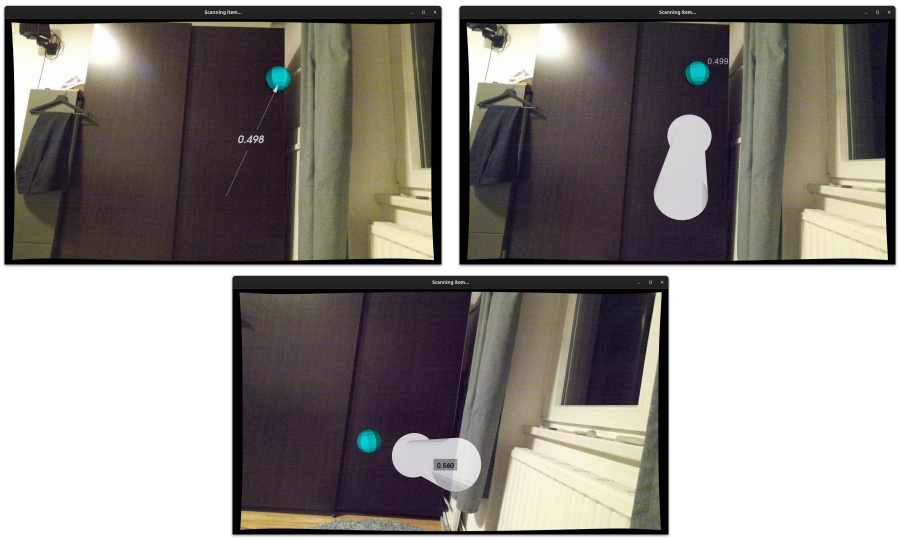

Figure 3.10: The different implemented labels in direct comparison

Figure 3.11: Label with background compared to no background

this case, the bounding box tries to show which element is seen as the object to scan, thus allowing the user to adapt or abort the scanning process if wrong information is detected.

The size of the bounding boxes is calculated based on algorithm 3.3, which specifies a centre point of the box. As already described in section 3.5 the distance from this centre point to the original camera position defines the radius of a sphere around this centre point. All the points, whose distance to the centre point is less than this sphere radius are interpreted as part of the scanned object. This radius is displayed as the bounding box's width, height and depth.

Figure 3.12: The marker without scaling close versus further away

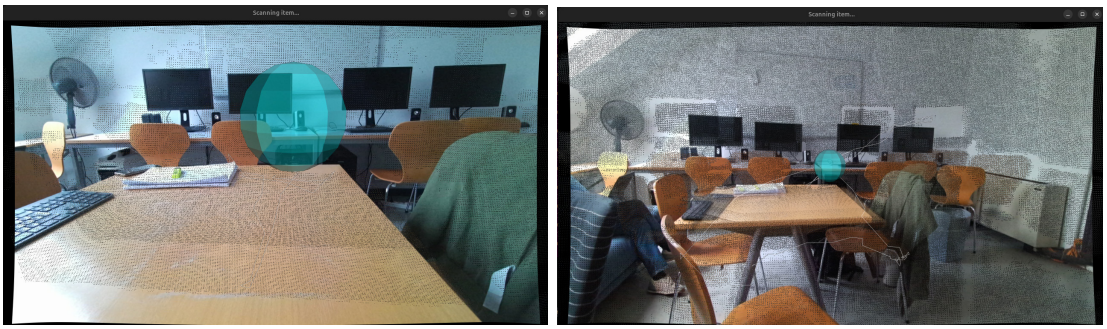

Figure 3.13: The marker with scaling close compared to further away

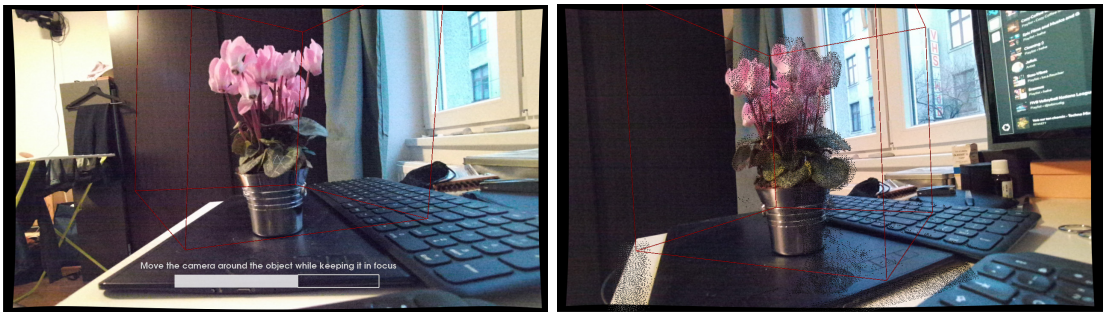

Figure 3.14: The scanned area with and without the indicator

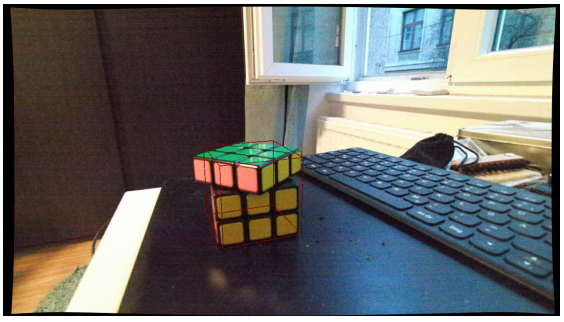

Figure 3.15: Bounding box around the object

CHAPTER 4

Results

The following section shows the findings of the paper. In particular, this includes some of the best practices for working with the application, the reasonings for the chosen AR elements, the implemented solution of combined NBV and vSLAM usage and an evaluation of the efficiency of the implemented application.

4.1 Best practices

This section describes some of the best practices for object centre detection and the scanning process.

4.1.1 Object centre detection

One of the most important aspects of the initialization process is that the camera always faces the object. Thus, the seven-second countdown before the start of the process must be used to position the camera accordingly. Here, it is important to move the camera slowly, because the vSLAM algorithm is already computing the camera position. Since the sphere radius around the object centre is set relative to the camera distance, the best practice is to position the camera close enough to the object to make it fully visible within the frame, but not further away. Otherwise, the radius will be too big and parts of the surroundings may be seen as part of the object. If the camera's position in the beginning is not set correctly, the scanning process should be restarted.

As soon as the camera frame is shown, you can use the triangle placed in the centre of the screen to keep the object in the centre frame, which enhances the bounding box calculation. Additionally, while only the horizontal rotation (i.e. the camera's rotation around the z-axis) is measured for the scanning process, the camera should also be moved vertically on the imagined scanning sphere. This gives the NBV algorithm

more information on the object's shape thus making the computed NBV positions more accurate.

4.1.2 Scanning process

While the scanning process is very straightforward, some recurring problems exist. They can be attributed to problems with the NBV algorithm since it sometimes might get stuck on positions and only move by a little bit. One fix to this problem is to move the camera to a position where new parts of the object are scanned and then again move the camera to the NBV position. The algorithm should then compute a new, better position.

Another best practice for finding the NBV position as quickly as possible is to follow the arrow and ignore the marker. The marker is most helpful if following the arrow leads to missing the NBV position and positioning the camera behind it. Then, small parts of the marker should be visible on the side parts, so the NBV position can be found quickly. The marker also helps identify the next camera position if it is on the other side of the scanning sphere. In the performed tests, this did not happen very often. The reasoning for this may be that the algorithm aims to connect the newly scanned points to already existing ones, thus positioning the NBV on the other side of the object might fail in that regard.

4.2 Augmented Reality User Interface

One of the main tasks of this paper was to Figure out which AR-based UI elements were the most useful when guiding the user to the next best scanning position. As mentioned in section 2.4, there are several necessary elements, like arrows, text labels and markers, but most elements have different possible implementations. This section shows which UI elements were implemented and why they were chosen over the other implementations.

4.2.1 Arrows

As shown in section 3.6.1 arrows can be displayed on the 2D screen or in the 3D world while rendered as a 2D or a 3D element. The arrow on the 2D screen and the 3D arrow rendered in the 3D world were implemented within this project while the 2D arrow rendered in the 3D world was already provided by RTAB-Map.

There are different pros and cons for choosing the arrow. Arrows rendered in the 3D world, for example, can direct in six directions (forward, backwards, left, right, up and down) while arrows in the 2D world are limited to only four directions (forward, backwards, left and right). When the scanning position is straight above or below you, arrows rendered on the 2D screen cannot direct the user to the position, as opposed to arrows rendered in the 3D world (see Figure 4.1).

The most important aspect of arrows is that they easily provide directional information. The element of the arrow responsible for this information is the arrowhead. Both

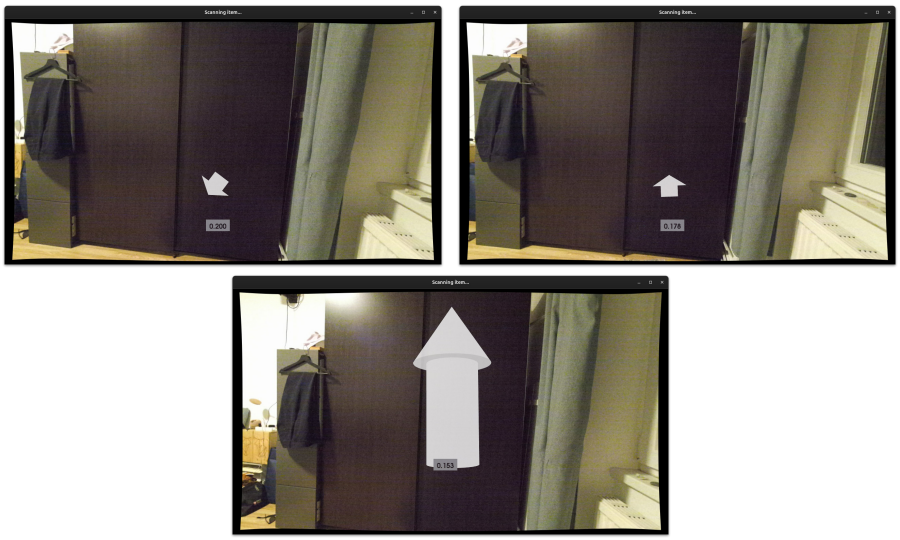

Figure 4.1: 2D and 3D arrow behaviour comparison with the destination right above the current position

implemented arrows thus put extra focus on the arrowhead, where the arrowhead's size is 1/2 (2D arrow) and 3/8 (3D arrow), respectively, of the total arrow length. The native arrow on the other hand does not focus on the arrowhead that much as it is always a fixed size no matter the arrow length. This especially makes distorting the arrow impossible, thus using the length of the arrow to display elements that are closer or further away is less useful for the native arrow. A comparison between the differently scaled arrows can also be seen in Figure 4.2.

Thus, when guiding the user to the next best scanning position the 3D arrow in the 3D world is the preferred option.

4.2.2 Labels

When scanning an object the user should constantly receive feedback from the scanning application. This includes information about the distance to the next scanning point, the current status of the scanning process, and positive feedback on when one of the subtasks, like reaching one of the NBVs, was achieved successfully. Some of this feedback is best displayed with text labels, which can be either shown on the screen or rendered in the world [10]. Each option has different advantages and problems for deciding on the rendering type.

One of the biggest advantages of labels rendered within the 3D world is that they can easily be placed next to or on top of the corresponding objects. This project includes, for example, arrows rendered in the 3D world, the destination marker or the bounding box around the object's centre. Thus, information like the distance to the marker or the current status of the initialization process may be placed as (text) labels in the 3D world

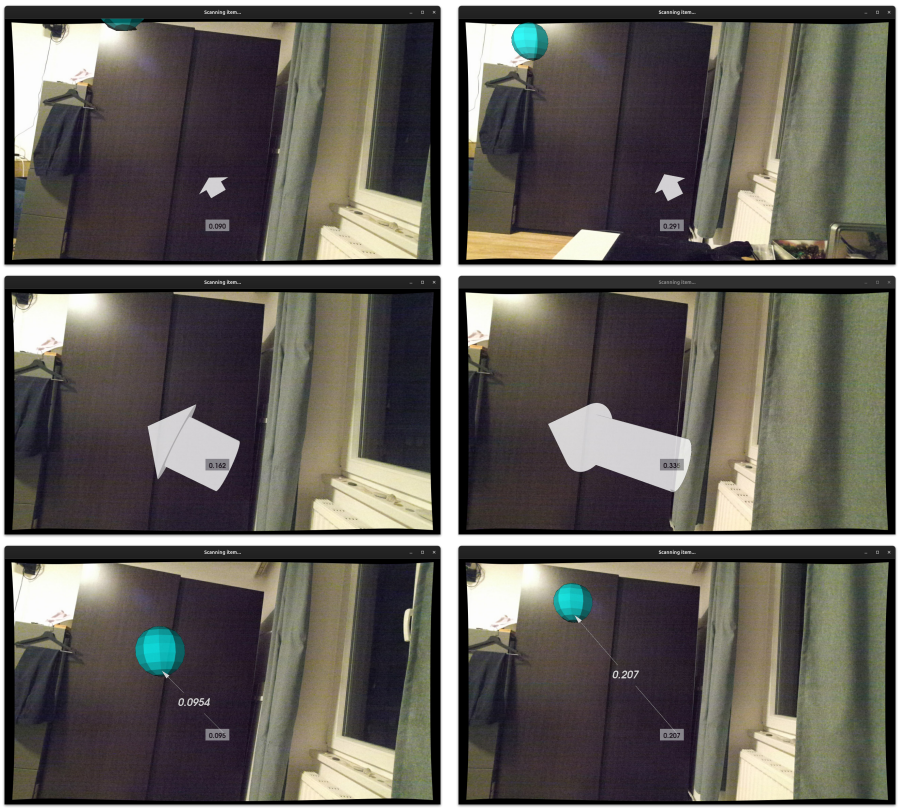

Figure 4.2: Arrowhead scaling comparison between the different designs

(see Figure 4.3). However, one of the problems with these labels is that rendering them with a shadow for increased visibility is more difficult since the text is rendered so it always points towards the camera.

Figure 4.3: Example use cases of labels rendered in the 3D world

On the other hand, labels that are placed on the 2D screen have the big advantage of being positioned more consistently. This means that no matter where the user looks, the label will always be placed in the same position and can thus be found easily. Therefore, labels unrelated to the 3D world, such as success messages, or that need to be placed consistently, like progress bars, should be rendered on the 2D screen (see Figure 4.4).

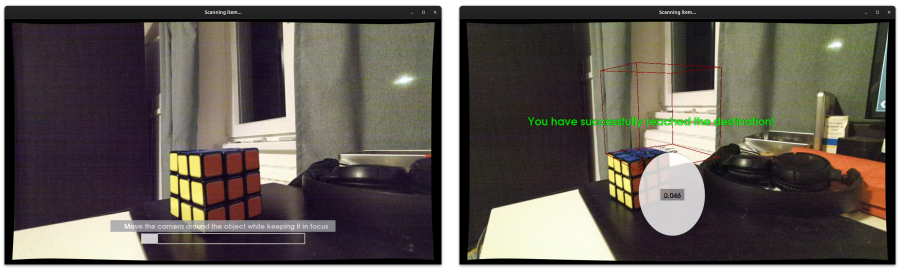

Figure 4.4: Example use cases of labels rendered on the 2D screen

For this project, labels were only rendered on the 2D screen. The only label that could be considered to be rendered in the 3D world is the distance to the marker. However, the beginning of the 3D arrow is relative to the camera's position, so it can also be rendered on the screen, circumventing the potential problem of label incoherence [10].

4.2.3 Marker

Section 2.4 gave a short overview of why markers are an important element regarding AR navigation in small spaces. This, however, does not fully apply to the scanning process. As discussed in section 4.1, the marker is often not required for navigating to the NBV position. However, it is still necessary for visualizing the point within the world, which is the implementation reason for the scanning process.

4.3 NBV and vSLAM

In theory, NBV and vSLAM are technologies that accompany each other well. While vSLAM tries to scan and create a map of the environment, NBV aims to find the best position to achieve that. However, one of the biggest issues with specifically NBV is that it is only trained on scans from objects without their surrounding environment. In contrast, vSLAM wants to focus on this information since positioning the camera in the 3D map is better with more feature points.

The implemented fix for this problem was an algorithm which aims to compute the object centre as accurately as possible. Afterwards, a spherical boundary was introduced, where all the points within the sphere were considered part of the object. All the other points were not further considered in the point cloud. This enhanced the NBV calculation and sped up the point cloud saving process. However, this process might not work for all objects. For example, objects that are very large on one axis (e.g. a lamp or a pencil case) could be given a sphere radius that is not big enough for the whole object or too big so features of the environment are considered part of the object (see Figure 4.5).

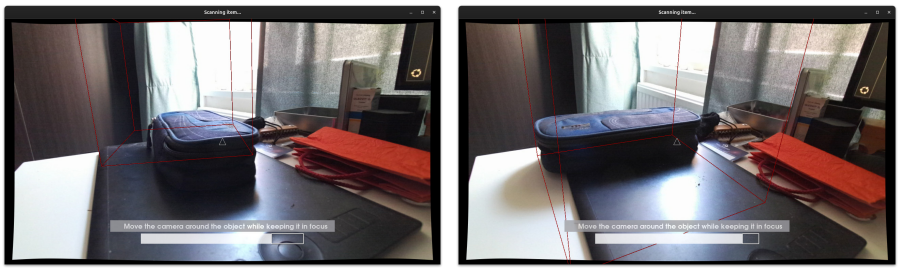

Figure 4.5: Proposed sphere sizes for different objects and their problems

4.4 Scanning process efficiency evaluation

This section provides a short overview of the factual evaluations done by this paper. This includes the processing speed of the point cloud, the computation speed of the NBV and the mean time it takes to reach the NBV.

First, the scanning speed of the point cloud relies a lot on the actual size of the already scanned data. In the tests, the minimum time for processing the point cloud was 0.2 seconds for around 7,000 points in the point cloud, while the maximum processing time was 1.3 seconds. The scan result of this test can be seen in Figure 4.6. In contrast, the computation speed for the NBV was consistently around 0.3 seconds, no matter the point cloud size. This can probably be attributed to the small variance of point clouds within the bounding box, as this value was mostly set between 700 and 1,000 points. With this small number of points, translating the point cloud into a probabilistic grid does not require much computation, thus it does not change the computation time. Finally, the mean time to reach the NBV was about 9 seconds. However, this number was also highly dependent on the position of the NBV compared to the current camera position. Points close to the current camera position were reached within 5 seconds, while it took up to 15 seconds to reach positions opposite the current camera position since the camera still needed to be moved on the scan sphere and point to the scanned object. Points close to the camera were more frequent than points opposite it, which can probably be attributed to NBVNet, which skewed the mean toward lower values.

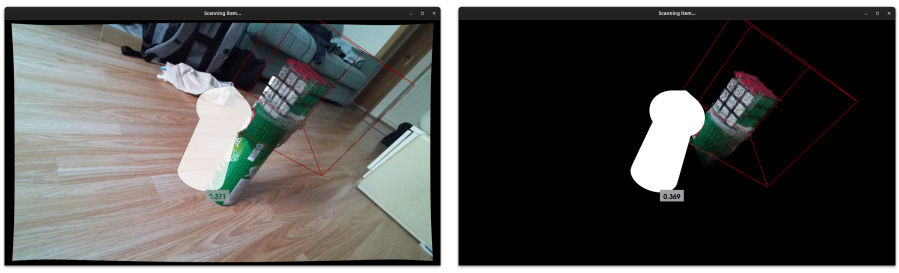

Figure 4.6: Output point cloud after one complete scan, with and without the camera data.

CHAPTER 5

Conclusion and Future Work

This project proposes guidelines for and implements the User Interface of an Augmented Reality based scanning application. This application is based on Visual Simultaneous Localization and Mapping (more precisely RTAB-Map for the scanning process, the Artificial Intelligence-model NBVNet [15] for the Next Best View calculation and 3D rendered arrows and markers (which in this case is implemented as a sphere) and 2D rendered labels as the navigation interface. While the reasoning for most of the implemented elements is based on previous works and argumentative reasons, some of the future work may include user studies for design evaluation and UI performance.

Other future work may also be done on the scanning completion process, i.e. figuring out how much of the object is scanned and when the scanning process is completed. This requires the implementation of object detection algorithms, which allow distinguishing between the scanned object and background elements [33]. This fixes a big problem of the combined usage of NBV and vSLAM, which is described in section 4.3 and may also remove the initialization part of the scanning process. However, the bounding box around the scanned object may still be kept for visual clarity and to provide user feedback on the scanning process. Other parts of future work may be on the UI for the scanning completion process and how its UX could be increased.

On the other hand, an algorithm or AI model which can compute the NBV while within a scene may be implemented. However, to the author's knowledge, this does not currently exist and may need to be developed beforehand. But a NBV implementation that is capable of computing the NBV of an object within a scene would also remove the necessity of using object detection algorithms while also solving the NBV and vSLAM problem described in section 4.3.

Overview of Generative AI Tools Used

Parts of the code referenced as [27] was developed using ChatGPT as a generative AI tool. The prompts used included the following topics:

- Generation of 2D triangles and rectangles using VTK
- Generation of filled and outlined 2D shapes using VTK
- Python script execution from C++
- 2D actor removal via VTK
- Cross and dot product calculation using Eigen's Vector3d/Vector3f

List of Figures

List of Tables

List of Algorithms

Acronyms

AI Artificial Intelligence. 1, 12–14, 29

AR Augmented Reality. v, vii, 1, 5, 6, 9, 11, 12, 15, 23, 24, 27, 29, 33

CNN Convolutional Neural Network. v, vii, 3, 4

CUDA Compute Unified Device Architecture. 12

DNN Deep Neural Network. 3, 4

DQN Q-deep Network. 4

ML Machine Learning. 3

NBV Next Best View. v, vii, 1, 3, 4, 6, 9–16, 18, 19, 23–25, 27–29

PCL PointCloud-Library. 15

RTAB-Map Real-Time Appearance-Based Mapping. v, vii, 4, 6, 11–13, 15, 20, 24, 29

SDK Software Development Kit. 12

UI User Interface. vii, 1, 5–8, 10, 11, 13, 17, 24, 29, 33

UX User Experience. 5, 6, 29

vSLAM Visual Simultaneous Localization and Mapping. v, vii, 4, 9, 15, 23, 27, 29

VTK Visualization Toolkit. 17, 18, 31

Bibliography

[1] Y. Arifin, T. Galih, S. Barlian, and E. Barlian. User experience metric for augmented reality application: A review. *Procedia Computer Science*, 135:648–656, 2018. The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018): Empowering Smart Technology in Digital Era for a Better Life.

[2] Azure Kinect Sensor SDK. `https://github.com/microsoft/Azure-Kinect-Sensor-SDK/tree/develop`. Last accessed: 2025-01-01.

[3] Azure Kinect Sensor SDK build guide. `https://github.com/microsoft/Azure-Kinect-Sensor-SDK/blob/develop/docs/building.md`. Last accessed: 2025-01-01.

[4] Azure Kinect Sensor SDK usage guide. `https://github.com/microsoft/Azure-Kinect-Sensor-SDK/blob/develop/docs/usage.md`. Last accessed: 2025-01-01.

[5] J. Cao, K.-Y. Lam, L.-H. Lee, X. Liu, P. Hui, and X. Su. Mobile augmented reality: User interfaces, frameworks, and intelligence. *ACM Comput. Surv.*, 55(9), Jan. 2023.

[6] C. Connolly. The determination of next best views. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 432–435, 1985.

[7] S. Isler, R. Sabzevari, J. Delmerico, and D. Scaramuzza. An information gain formulation for active volumetric 3d reconstruction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3477–3484, 2016.

[8] R. Joshi, A. Hiwale, S. Birajdar, and R. Gound. Indoor navigation with augmented reality. In A. Kumar and S. Mozar, editors, *ICCCE 2019*, pages 159–165, Singapore, 2020. Springer Singapore.

[9] O. Kähler, V. Adrian Prisacariu, C. Yuheng Ren, X. Sun, P. Torr, and D. Murray. Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1241–1250, 2015.

[10] T. Köppel, M. Eduard Gröller, and H.-Y. Wu. Context-responsive labeling in augmented reality. In *2021 IEEE 14th Pacific Visualization Symposium (PacificVis)*, pages 91–100, 2021.

[11] M. Labbé and F. Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[12] J. Lee, F. Jin, Y. Kim, and D. Lindlbauer. User preference for navigation instructions in mixed reality. In *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 802–811, 2022.

[13] S. Mang. Towards improving instruction presentation for indoor navigation. Master's thesis, Dept. Electr. Eng. Inform. Technol., Inst. Med. Technol., München, Germany, 2013.

[14] A. Martin, J. Cheriyan, J. Ganesh, J. Sebastian, and J. V. Indoor navigation using augmented reality. *EAI Endorsed Transactions on Creative Technologies*, 8(26), 2 2021.

[15] M. Mendoza, J. I. Vasquez-Gomez, H. Taud, L. E. Sucar, and C. Reta. Supervised learning of the next-best-view for 3d object reconstruction. *Pattern Recognition Letters*, 133:224–231, 2020.

[16] NBV-Net source code. `https://github.com/irvingvasquez/nbv-net`. Last accessed: 2025-01-26.

[17] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[18] PCLVisualizer documentation. `https://pointclouds.org/documentation/classpcl_1_1visualization_1_1_p_c_l_visualizer.html`. Last accessed: 2025-01-09.

[19] PCL documentation. `http://pointclouds.org/documentation/group__io.html`. Last accessed: 2025-01-09.

[20] Python embedding in C and C++. `https://docs.python.org/3/extending/embedding.html`. Last accessed: 2025-01-26.

[21] RTAB-Map example for RGB-D mapping. `https://github.com/introlab/rtabmap/wiki/Cplusplus-RGBD-Mapping`. Last accessed: 2025-01-02.

[22] RTAB-Map build guide. `http://introlab.github.io/rtabmap/`. Last accessed: 2025-01-02.

[23] RTAB-Map build guide. https://github.com/introlab/rtabmap/wiki/Installation#ubuntu. Last accessed: 2025-01-01.

[24] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, 2013.

[25] C. Sandor, G. Klinker, B. Thomas, M. Billinghurst, and M. Haller. Lessons learned in designing ubiquitous augmented reality user interfaces. In *Emerging Technologies of Augmented Reality*, pages 218–235. IGI Global, United States, 2006.

[26] T. Taketomi, H. Uchiyama, and S. Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, Jun 2017.

[27] Source code of the project with build guideline. https://gitlab.cg.tuwien.ac.at/stef/nbv-guidance. Last accessed: 2025-02-18.

[28] torch documentation. https://pypi.org/project/torch/. Last accessed: 2025-01-26.

[29] L. Ventä-Olkkonen, M. Posti, O. Koskenranta, and J. Häkkilä. Investigating the balance between virtuality and reality in mobile mixed reality ui design: user perception of an augmented city. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*, NordiCHI '14, page 137–146, New York, NY, USA, 2014. Association for Computing Machinery.

[30] Viewpoint intersection algorithm source. https://stackoverflow.com/questions/48154210/3d-point-closest-to-multiple-lines-in-3d-space. Last accessed: 2025-02-11.

[31] T. Wang, W. Xi, Y. Cheng, H. Han, and Y. Yang. Rl-nbv: A deep reinforcement learning based next-best-view method for unknown object reconstruction. *Pattern Recognition Letters*, 184:1–6, 2024.

[32] R. Zeng, W. Zhao, and Y.-J. Liu. Pc-nbv: A point cloud based deep network for efficient next best view planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7050–7057, 2020.

[33] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.