SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Intelligent and Automated Path-Planning with Mixed Reality for Mobile Systems within Hospital Environments

Lars Christiansen

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Intelligent and Automated Path-Planning with Mixed Reality for Mobile Systems within Hospital Environments

Intelligente und automatisierte Pfadplanung mit Mixed Reality für mobile Systeme in Krankenhausumgebungen

Author:	Lars Christiansen
Examiner:	Prof. Dr. Nassir Navab
Supervisor:	Kevin Yu
Submission Date:	15.11.2024

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.11.2024

Abstract

This thesis investigates an automated path-planning solution supported by Mixed Reality to improve navigation in hospital environments. The primary objective is to enable autonomous path generation for Brainlab's Loop-X, a mobile imaging robot in combination with the Microsoft HoloLens 2. The HoloLens acts as the eyes of the Loop-X to accommodate the missing on-board sensors necessary for room reconstruction. Spatial data gets converted to a voxel octree, enabling a 3-dimensional representation. A 2D distance field is subsequently generated to facilitate fast collision detection and navigation during path-planning. Path generation utilizes a customized Rapidly Exploring Random Tree (RRT*) algorithm, with different sampling strategies tailored for hospital settings. A Mixed Reality Interface is introduced, allowing operators to specify waypoints, generate a collision-free path, and direct the Loop-X to autonomously follow the designated route. This method aims to provide a robust and safe path-planning solution while addressing the current challenges in manual navigation.

Kurzfassung

In dieser Arbeit wird eine automatische Pfadplanungslösung mithilfe von Mixed Reality in Krankenhausumgebungen untersucht. Das primäre Ziel ist es, autonome Navigation für Brainlabs Loop-X, ein mobiles CT, in Kombination mit der Microsoft HoloLens 2 zu ermöglichen. Die HoloLens wird zur Erfassung räumlicher Umgebungsdaten verwendet, die dann in einen Voxel-Octree umgewandelt werden. Für effiziente Kollisionschecks und den Pfadplanungsprozess wird daraus dann ein 2D Distance Field erstellt. Zur Pfad-Erzeugung wird der Rapidly Exploring Random Tree (RRT*)-Algorithmus verwendet, wobei verschiedene Sampling-Strategien untersucht werden. Es wird eine Mixed-Reality-Schnittstelle eingeführt, die es dem Benutzer ermöglicht, Wegpunkte festzulegen, einen kollisionsfreien Pfad zu generieren und den Loop-X diesem autonom folgen zu lassen. Diese Methode zielt darauf ab, eine robuste und sichere Lösung für die Pfadgenerierung zu bieten und gleichzeitig die aktuellen Herausforderungen bei der manuellen Navigation zu bewältigen.

Contents

1. Introduction

1.1. Motivation and Context

Integrating robot motion planning within hospital environments holds significant potential to enhance the efficiency of routine tasks for medical staff while reducing cognitive load. However, deploying such systems in these complex and dynamic environments requires intelligent path-planning solutions to ensure safe and efficient navigation.

Mobile imaging robots, such as Brainlab's Loop-X, are designed to improve flexibility by enabling use in various hospital locations. Long-distance navigation is manually controlled via a joystick, requiring a staff member to accompany the robot. Moreover, changing the robot's driving direction necessitates readjusting its wheels, which adds to the operational complexity and increases the time needed for even minor adjustments. This makes manual steering mentally demanding for hospital personnel, especially when navigating long distances. Although the Loop-X can autonomously move to preprogrammed positions, its current automation is limited by hospital environments' dynamic and unpredictable nature, such as the changing placement of equipment.

MR devices, like Microsoft's HoloLens 2, offer a promising approach to address these challenges. Equipped with accurate indoor mapping capabilities, they can provide real-time data for motion planning, enabling robots to better understand and adapt to their surroundings. Additionally, the HoloLens facilitates augmented visualization, allowing operators to view and validate the planned path before the robot moves. When combined with an intelligent path-planning algorithm, MR technologies can significantly enhance both usability and safety, reducing the cognitive effort required from staff and minimizing operational delays due to manual adjustments.

1.2. Problem Statement

This work explores a solution that tries to alleviate the operational challenges associated with manually navigating the Loop-X through hospital environments and aims to provide an automated way for the operator to control the device's motion. The goal of this project is to implement an immersive MR application for the Microsoft HoloLens 2, which will enable the operator to track the robot, specify waypoints within the environment, plan the robot's motion, visualize the generated path and potential obstacle encounters, and execute the navigation process automatically. Once moving, the operator needs to oversee its correct execution and intervene only if necessary.

Various motion planning strategies will be explored to determine their effectiveness for

navigation tasks in the confined and dynamic spaces commonly found in hospitals. Ultimately, the solution aims to enhance both the safety and usability of the Loop-X, reducing the cognitive load on operators, increasing navigation efficiency, and improving the overall workflow within hospital settings.

1.3. Thesis Structure

This thesis begins with an overview of the fundamental hardware and technology utilized, followed by a review of related work in the field. Subsequently, the high-level system architecture is described, including the selection of algorithms and design principles guiding the interface. Next, the thesis explores specific implementation aspects, elaborating on the applied algorithms and distinct features alongside an in-depth presentation of the User Interface. In the discussion experimental results are critically analyzed and observed limitations are addressed. Finally, the thesis concludes with insights into potential directions for future research.

2. Background and Related Work

This chapter discusses the hardware used for this thesis, as well as related work on indoor path-planning interfaces in the context of Mixed Reality. It also covers existing path-planning algorithms and spatial mesh processing.

2.1. Background and Devices

2.1.1. Augmented Reality Devices

Augmented Reality (AR) glasses are wearable devices that overlay digital information onto the physical world, allowing users to interact with virtual objects. These devices typically use various sensors and cameras to create immersive experiences. Although AR glasses differ in design and functionality, they commonly feature depth sensing, hand-eye tracking, and spatial mapping. These capabilities support interactive simulations, industrial design, and remote collaboration applications. Examples of AR glasses include the Microsoft HoloLens 2, the Magic Leap 2, and the new Apple Vision Pro.

2.1.2. HoloLens 2

This work will restrict its implementation on Microsoft HoloLens 2, which brings its own feature toolset. It is equipped with advanced sensors, including depth cameras and several light cameras used for head, eye, and hand tracking, providing a good background for AR applications. This also allows for accessing a real-time environment mesh generated by the device and interacting with virtual objects using hand gestures. The HoloLens 2 is equipped with a custom GPU and a mobile CPU based on the Snapdragon architecture. However, it lacks active cooling mechanisms, such as cooling fans, which are typically used in other devices to manage heat dissipation. [1]

Figure 2.1.: The Microsoft HoloLens 2 [2].

2.1.3. Loop-X

The Loop-X is a mobile imaging robot developed by Brainlab, designed for intraoperative use and capable of providing patient-specific imaging solutions [3]. It is commonly used in procedures such as spinal incision planning, where it minimizes the patient's exposure to radiation by selectively scanning only necessary areas. Due to its mobility, the Loop-X is often stored in a dedicated location and may need to be relocated to different rooms throughout the day, depending on clinical requirements.

Figure 2.2.: Loop-X with Operation Table [4].

To control the robot's motion, operators are provided with a tablet interface (see Figure 2.3) to either set a target position or control the robot in real time using a thumbstick. This system is particularly well-suited for precise, near-patient positioning, allowing for flexible movement. Unlike traditional imaging devices, the Loop-X is not restricted to a fixed center of rotation, as it can adjust its wheelbase to pivot around any specified point. However, this flexibility results in additional delays whenever the robot's movement is adjusted, especially during multiple corrections.

Figure 2.3.: Control Panel with Movement Controls on the Right, Detector Position and Start/Stop for Preprogrammed Movements on the Left

As the Loop-X lacks an integrated environment-scanning technology, navigating longer distances through complex environments can be time-consuming, as the operator must manually control the robot's path. Nevertheless, the Loop-X provides an external (UDP) connection for sending custom target positions. However, the specifics of how the robot internally plans and executes these movements are unclear. The path-planning system developed by Brainlab is proprietary and not publicly disclosed, leaving the exact mechanisms of the internal MotionModel — how it transitions between two poses and translates a target position into a feasible path — unknown.

2.1.4. Development Platform

For the development of the application, Unity in combination with the Mixed Reality Toolkit 3 (MRTK) is used. Unity is a widely adopted game engine known for its flexibility and ease of use in creating 3D environments and interactive applications. Unity is particularly advantageous for this project due to its robust support for mixed reality development, including seamless integration with devices like the Microsoft HoloLens 2 and rapid prototype iteration.

2.2. Related Work

2.2.1. Mixed Reality in Medical Setting

Recently, MR has gained increasing attention in modern medical applications due to its potential to enhance various aspects of healthcare. MR applications blend the physical and digital work, offering new ways to visualize and interact with the environment. This is particularly valuable in medical education, surgical planning, and assisting in complex procedures [5][6]. Although significant advancements have been made, most of these innovations have not yet been integrated into daily use. The devices remain costly and frequently present further constraints in outdoor environments or scenarios with challenging lighting conditions [7].

2.2.2. Mixed Reality for Indoor Path Planning

There is significant work on integrating mixed reality (MR) for path-planning tasks, particularly in indoor navigation. Much of this research focuses on guiding users through unfamiliar environments [8]. Also, an interesting application of MR in indoor navigation is seen in the work of Yamashita, K. Sato, S. Sato, and Matsubayashi, which combines global route planning with local obstacle avoidance to assist visually impaired individuals in navigating multi-floor buildings [9]. They first plan a global route using Dijkstra's algorithm through the building and then use A* to refine the local path on the NavMesh generated with Unity.

Expanding the problem to 3 dimensions, it has also been shown that it's possible to plan an indoor flight path for drones using the HoloLens 2 and provide a User Interface for editing different waypoints [10, 11].

In the context of the limited computational capabilities inherent to the HoloLens 2, it has been suggested to delegate processing tasks to an external host machine via a remote connection, thereby enabling the full utilization of computational resources for systems with high processing demands [11, 12]. Furthermore, integration with the Robot Operating System (ROS), which also comes with path-planning functionalities, has been proposed to enhance operational efficiency [13].

2.2.3. Spatial Perception and Mesh Processing

Most path-planning algorithms require some form of mesh processing to generate navigable paths. The HoloLens provides valuable data for this purpose, including access to both its depth camera and spatial mesh [14].

These raw data streams need to be further processed to be suitable for navigation purposes. One common approach for 3D spatial analysis is to convert the spatial mesh into a voxel grid. This technique simplifies the mesh into a uniform grid of voxels, enabling efficient collision checks and path-planning. An example of this approach is employed by the Recast library, which generates navigation meshes in Unity by voxelizing 3D spaces [15].

A study by Hübner, Weinmann, and Wursthorn further explores this concept by converting HoloLens Time-of-Flight triangles, captured through the depth camera, into a voxel model. The voxelized data is then classified into different categories such as walls, ceilings, and floors, which significantly enhances the accuracy and applicability of the mesh for navigation purposes [16]. Fortunately, the MRTK developer set already provides a fully constructed mesh from all triangles.

Moreover, numerous motion planning implementations utilize 2D maps to reduce computational complexity relative to 3D data. Occupancy grids and distance fields are frequently employed within these frameworks to enable efficient collision detection and rapid identification of the nearest obstacles [17, 18].

2.2.4. Path-Planning

Path-planning represents a foundational area within robotics, instrumental to enabling autonomous navigation [19]. The primary objective in path-planning is to compute a collision-free trajectory that accommodates obstacles and various environmental constraints. For relatively simple, low-dimensional graph-based structures, established algorithms such as A* and Dijkstra's algorithm are frequently employed. In dynamic or evolving environments, the D* algorithm is commonly favored due to its extension of A*, allowing for efficient replanning as the map updates in response to environmental changes [18].

However, these algorithms rely on discrete, graph-like representations of space, making them suboptimal for problems set in a continuous configuration space.

In continuous spaces, sampling-based planners have emerged as the preferred approach, especially for high-dimensional configuration spaces where they can incorporate the vehicle's motion model. A fundamental component of these algorithms is a robust collision-checking function, which determines if a given configuration intersects with obstacles. This feature

is particularly relevant to our problem since the Loop-X vehicle cannot be approximated as a point; instead, its rectangular dimensions and orientation require careful consideration, especially in constrained or narrow spaces. This scenario is analogous to the piano mover's problem, which is effectively addressed by these algorithms [18].

One widely used sampling-based algorithm is the Rapidly Exploring Random Tree (RRT). RRT has several notable variants, including the Informed RRT* algorithm, which incorporates incremental rewiring and ellipsoidal sampling techniques to increase efficiency. These enhancements enable the algorithm to converge asymptotically towards an optimal solution over time, providing a near-optimal path as the number of iterations grows [20]. In uncertain environments, Chance Constrained RRT provides safe paths within a given probability [21]. Additionally, research has explored improving sampling efficiency, by introducing heuristics or targeted strategies for handling narrow passages [22, 23, 24]. Triangulation has also proven effective for optimizing paths by guiding samples toward strategically placed "beacon nodes," which can significantly reduce path cost [25]. When dealing with complex cost spaces, the Transition-based RRT (T-RRT) further extends RRT's capabilities by biasing exploration towards low-cost regions, thus achieving more cost-effective paths [26]. However, a limitation of these algorithms is their incompleteness; they cannot reliably determine the absence of a feasible path in situations where one may not exist.

Another popular sampling-based approach is the Probabilistic Roadmap (PRM) method. While PRM is highly efficient for multi-query scenarios, it generally underperforms in single-query situations compared to RRT-based algorithms, due to the computational expense involved in constructing a Voronoi tessellation of the map. This limitation also renders PRM less suitable for dynamic environments and for the specific objectives of this work [18, 27, 28]. Another approach is the Artificial Potential Field (APF), which enables real-time path planning by generating repulsive force fields around obstacles and an attractive force toward the goal, guiding the agent within this potential landscape [29]. APF's simplicity and low computational cost make it effective for real-time applications, producing smooth, predictable paths. However, APF is prone to local minima, where opposing forces can trap the agent, and it doesn't inherently account for dynamic or kinematic constraints. These limitations mean APF often requires enhancements, such as escape behaviors or hybridization with other algorithms, to perform reliably in complex environments [30].

3. System Architecture and Design

This chapter provides a comprehensive overview of the proposed system and the design of the application. It emphasizes the rationale behind the selection of specific algorithms and systems, detailing how they contribute to achieving a reliable and efficient solution. Additionally, the chapter discusses the key design considerations necessary for ensuring the development of a robust and safe application.

3.1. Overview of the Proposed System

The proposed system primarily focuses on simplicity and a user-friendliness interface for operating the Loop-X robot. Therefore, the system architecture restricts itself to involve as few components as possible. Specifically, it consists of the Microsoft HoloLens 2 running a standalone application and the Loop-X mobile imaging robot. This approach ensures that the system remains self-contained and portable. However, this design choice also imposes limitations on computational resources, as all processing must be performed using the hardware capabilities of the HoloLens.

The HoloLens reconstructs the environment through its spatial mapping capabilities. Since the Loop-X's onboard cameras and sensors are dedicated solely to imaging purposes and do not provide environmental scanning data suitable for navigation, the HoloLens is the primary device for generating a spatial mesh of the surroundings. This spatial mesh is processed to create an intermediate occupancy grid, the foundation for the path-planning algorithms.

3.2. Environmental Scanning

The HoloLens 2 supports two methods for handling spatial data. First, the spatial awareness feature generates a collection of meshes that can be used, for example, for occlusion of virtual objects, collision testing, or physical simulation (Figure 3.1). Secondly, the Scene Understanding SDK provides a more refined collection of labeled meshes. This SDK aims to eliminate many issues in the spatial mesh, such as hallucinations, holes, and unclean faces. However, it may not recognize every small obstacle present on the floor.

Since the depth camera can only detect meshes within a maximum radius of 3.1 m from the user, distant meshes can only be captured by physically visiting those locations. Fortunately, the generated mesh can be cached and may persist across application boundaries, which enhances usability in known environments. The update interval and resolution of spatial data on the HoloLens 2 cannot be directly configured via MRTK3; instead, they depend on device-specific factors such as battery level and processing load. Under typical circumstances,

update intervals range from 0.1 to 5 seconds. In our experiments, the resolution allowed for detecting obstacles as small as 10 cm, though lower lighting can impact accuracy. [31]

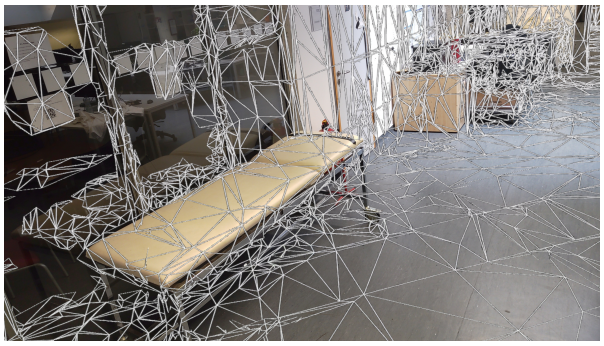


Figure 3.1.: Spatial Mesh generated by the HoloLens 2.

3.3. Path-Planning

3.3.1. Selection of Algorithm

As discussed in related works, there are numerous options for path-planning algorithms. For our problem, two key requirements must be satisfied:

1. Rapid convergence to a solution with minimal computational overhead, particularly in high-dimensional spaces.
2. The capability to solve the piano mover's problem, i.e., considering the vehicle's motion model and geometry in constrained environments.

Given that the system is expected to operate in confined spaces such as long, narrow corridors, it is crucial to select efficient algorithms in such scenarios. In this work, we assume a simple continuous configuration space suitable for a car-like vehicle: $q = (x, y, \theta) \in \mathcal{C}$ where $\mathcal{C} \in \mathbb{R}^2 \times S^1$ [18]. Based on these requirements, we focus on RRT*, an improved variant of the well-known Rapidly-exploring Random Tree (RRT) algorithm, which, as a sampling-based algorithm, is particularly suitable for problems where the configuration space is high-dimensional and complex [19].

Advantages of RRT*

The choice of RRT* stems from several key advantages:

Asymptotic Optimality:
Unlike the basic RRT algorithm, which focuses on finding a feasible solution quickly, RRT* guarantees asymptotic optimality [20]. RRT* converges towards the optimal path as the number of iterations increases, making it ideal for tasks where both feasibility and optimality are crucial, such as autonomous navigation in cluttered environments like narrow hallways.

Scalability in High-Dimensional Spaces:
RRT* is effective in high-dimensional configuration spaces [19]. For a problem with three degrees of freedom, such as our car-like vehicle's configuration space, RRT* ensures that computational costs remain manageable even in complex environments. Additionally, this provides the ability to change the configuration space later on to include additional freedoms.

Disadvantages of RRT*

Despite being asymptotically optimal often, many iterations are involved in finding a suitable path [24]. This is particularly true when working in cluttered environments with many obstacles and narrow passages. As our environment in the hospital often implies such constraints, improved versions of the RRT* are considered in this work. One key component that leads to a slow convergence rate is the sampling strategy used in the algorithm, as random sampling may fail near obstacles. To improve the performance in such spaces, combining the A* heuristic with the node sampling like the Hybrid RRT-A* drastically reduces exploration time [22]. Furthermore, Fast-RRT also promises to improve sampling by disallowing sampling in well-explored areas and using certain rewiring strategies [24]. The detailed implementation of our solution will be discussed in chapter 4.

3.3.2. Assumptions Regarding the Motion Model

An essential consideration for path planners is the motion model or the transition behavior between two configurations a and b. The Loop-X utilizes an internal model to execute a given pose change. Based on practical observation of the movement, the internal process appears to rely on linear interpolation for both rotation and translation. This approach is assumed in this work due to its straightforward implementation and practicality.

3.4. Communication between HoloLens and Loop-X

The Loop-X system provides an internal WiFi network that enables the HoloLens to establish a connection. Target poses can be adjusted through a UDP interface, and movement execution commands can be transmitted. Once a path has been generated, the application iterates through each waypoint, calculating the relative offset to be applied to the current position. Given the non-transactional nature of the UDP interface, the application lacks the capability to detect packet loss. To minimize the risk of skipping waypoints due to undetected packet loss, the next waypoint is transmitted only when the UDP interface position matches the intended target position. If alignment is not achieved, the user might reinitiate movement as needed.

3.5. MR Application Interface

Within the MR interface provided by the HoloLens application, the user follows a structured sequence of steps to operate the system effectively. All steps are within the consideration of the Human Computer Interaction (HCI) principles [32]. For the initial setup, initiating the movement of the Loop-X robot requires completing seven consecutive steps. However, after initial setup, mesh processing operates passively, and device connection details are saved, thus reducing the procedure to five essential steps required to accomplish the intended operation:

1. **Environment Scanning**: (Full) Scan of the environment to generate a spatial mesh.
2. **Start Pose Tracking**: Set or track the initial position of the Loop-X.
3. **Target Pose Specification**: Define the target pose and optional waypoints using AR.
4. **Path Generation**: Compute the optimal path while avoiding obstacles.
5. **Path Visualization**: Display the path for user review.
6. **Device Connection**: Connect the HoloLens 2 to the Loop-X robot.
7. **Execution of Motion**: Execute the path upon user approval.

3.5.1. Environment Scanning

In the first step, the user has to scan the environment with the HoloLens 2, capturing spatial geometry to create the necessary spatial mesh for path planning. This mesh is cached afterward; therefore, this step must only be performed when the environment changes. Notably, this step does not impose a significant workload on the user by design, as the scan can be done simultaneously while adding waypoints. Typically, the operator starts near the Loop-X and moves to the target position, scanning the environment on the way. This eliminates the need for additional scanning imposed on the operator. Nevertheless, it seems important to have some visual feedback where the unscanned area is. For this purpose, the user can always enable the spatial mesh overlay, displaying scanned areas.

3.5.2. Start Pose Tracking

The first waypoint is fixed to the current position of the Loop-X. An integrated tracking system automatically finds the robot and places an initial waypoint to make it easier for the user. As this might not be sufficient if tracking fails, additionally, it is also possible to set the first waypoint manually at the current position of the Loop-X.

3.5.3. Target Pose Specification

Next, the user defines the target pose and optional intermediate waypoints by placing a ghost model of the Loop-X in the AR environment. The user can easily adjust the target pose to ensure that it fits the specific spatial constraints of the environment. This model can give first hints if intersections with the spatial mesh occur.

3.5.4. Path Generation

Once waypoints are set, the user can trigger path generation, calculating an optimal path from the starting position to the target pose. The algorithm accounts for obstacles and non-drivable areas using the spatial mesh and occupancy grid generated during environment scanning. The goal is to ensure a smooth, collision-free path, considering the robot's movement constraints and the environment's geometry.

3.5.5. Path Visualization

In this step, the system visualizes the planned path within the AR interface. This allows the user to review the path's feasibility and safety before execution. Adjustments can be made to the waypoints or target pose to ensure the robot's movement is safe and efficient.

3.5.6. Device Connection

Following environment scanning a communication link between the HoloLens 2 and the Loop-X robot is established. This enables the exchange of data necessary for navigation and robot control, ensuring that both systems remain synchronized during operation.

3.5.7. Execution of Motion

After the user reviews and approves the path, the system transmits it to the Loop-X, which then can autonomously follow the planned path to reach the target pose. The user can monitor the robot's progress in real-time through the AR interface, ensuring precise and safe navigation. If any issues occur during motion execution, the operator can abort it at any time.

4. Implementation

This chapter focuses on the development tools used in this work and provides a detailed explanation of the application's core features. Additionally, it discusses the design and development of the user interface and how users interact with virtual objects. All methods described refer to the actual implementation used in the provided repository (A.1).

4.1. Development Environment and Tools

The development of the application for the HoloLens 2 was carried out using the Unity3D game engine (version 2022.3) in conjunction with the Mixed Reality Toolkit (MRTK) from Microsoft. These tools were chosen for their robust support for MR development, ease of use, and the comprehensive feature set they provide, specifically tailored for creating interactive, immersive applications [33]. Unity supports both UnityScript and C# for coding. As a scripting backend, IL2CPP is utilized, which supports AOT code compilation for the Windows universal platform [34]. Performance critical sections were optimized with the Burst compiler, which can compile to "highly optimized native code" [35].

4.1.1. Mixed Reality Support

Unity has various XR features supporting virtual, mixed, and mobile augmented reality development tools. The HoloLens 2 is, amongst other devices, directly supported by Unity [36]. The key component used here is the Mixed Reality Toolkit 3.0 (MRTK), which, in combination with the OpenXR Plugin, provides "cross-platform input and building blocks for spatial interactions and UI" [37].

4.1.2. Environment Scanning

Unity supports spatial meshes within the ARFoundation using the cross-platform ARMeshManager interface [38]. When used on the device, the ARMeshManager yields multiple meshes generated by the HoloLens's sensors. Also, the Scene Understanding SDK is not supported in MRTK version 3.0 and is replaced by the less feature-rich ARPlaneManager. Although not quite as powerful as the Scene Understanding mechanism, it is cross-platform and can provide the application with horizontal or vertical planes.

4.1.3. Tracking

The Vuforia Engine is being used to track the Loop-X. The library adds advanced computer vision functionality to MR devices, like tracking images and 3D models [39]. In this work, the

Model Target feature automatically sets the starting waypoint on the Loop-X. As the Vuforia library provides a plugin for Unity, setup is straightforward and directly integrates into the application. With Vuforia, it is possible to directly listen to model targets tracking events and access the tracked pose. To optimize performance, the application includes a dedicated tracking mode that users can activate to enable tracking functionality. By default, tracking remains disabled to enhance the usability and responsiveness of the user interface.

4.2. Collision Checks

The main bottleneck in path planning with RRT* are environment queries like collision checks [40]. With that in mind, a fast way of handling many collisions needed to be found. In this work, collision checks were reimplemented due to limitations in Unity's standard approach, where attaching a collider to an object and performing collision computations on the main thread lacked support for parallelization. Additionally, collision check methods are not exposed, necessitating a customized implementation. Most of these algorithms are taken from the book *Real-Time Rendering 4th Edition* and are presented in the following. While this work does not primarily focus on collision checks, a brief overview of the most common checks is provided:

4.2.1. Mathematical Definitions

Ray

A ray is mathematically defined as:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}, \quad t \geq 0$$

where:

- $\mathbf{o}$ is the origin of the ray,
- $\mathbf{d}$ is the direction vector of the ray,
- t is a scalar that represents the distance along the ray.

AABB

An Axis Aligned Bounding Box (AABB) is defined by its minimum and maximum coordinates:

$$\mathbf{b}_{\text{min}} = (x_{\text{min}}, y_{\text{min}}, z_{\text{min}}), \quad \mathbf{b}_{\text{max}} = (x_{\text{max}}, y_{\text{max}}, z_{\text{max}})$$

OBB

An Oriented Bounding Box (OBB) is defined by its center $\mathbf{c}$, half-lengths along each of its local axes (h_x, h_y, h_z), and three orthonormal direction vectors $\mathbf{u}_x$, $\mathbf{u}_y$, and $\mathbf{u}_z$ that represent

the local axes of the box. Mathematically, the OBB can be described as the set of points $\mathbf{p}$ such that:

$$\mathbf{p} = \mathbf{c} + \lambda_x \mathbf{u}_x + \lambda_y \mathbf{u}_y + \lambda_z \mathbf{u}_z$$

4.2.2. Ray - AABB Intersection

The slab method is one of the most efficient ways to compute ray-box intersections. Every pair of parallel planes defines a slab, so an axis-aligned bounding box (AABB) can be considered as the intersection of three slabs — one for each axis (x, y, z). For each slab, the ray is checked to see if it passes through, and the intersection is computed using the parametric form of the ray equation. By determining the range of valid values of the ray parameter t across all axes, the algorithm efficiently determines if and where the ray intersects the box. This algorithm can also be used for a Ray - OBB intersection test when transforming the ray into the local coordinate system of the OBB. [41]

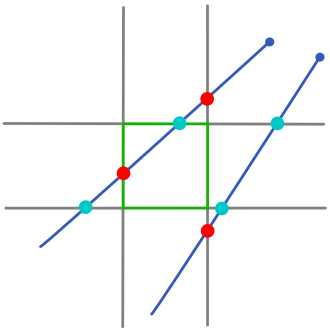


Figure 4.1.: 2D Slab Method for Two Rays (Blue) and Box (Green), with Intersection Order Determining Collision.

4.2.3. OBB - OBB Intersection

To check for an intersection between two Oriented Bounding Boxes, the *Separating Axis Theorem (SAT)* is commonly used (See Figure 4.2). The SAT states that two convex objects do not overlap if a plane (called a separating axis) exists along which the projections of the two objects are disjoint.

In the case of two OBBs, the potential separating axes include:

- The three axes corresponding to the local axes of the first OBB: $\mathbf{u}_x^1, \mathbf{u}_y^1, \mathbf{u}_z^1$,
- The three axes corresponding to the local axes of the second OBB: $\mathbf{u}_x^2, \mathbf{u}_y^2, \mathbf{u}_z^2$,
- The nine axes formed by the cross products of the axes of the two boxes: $\mathbf{u}_i^1 \times \mathbf{u}_j^2$, where $i, j \in \{x, y, z\}$.

To test whether the two OBBs intersect, we project the vertices of both boxes onto each of these 15 axes and check for overlaps. Mathematically, for each separating axis $\mathbf{a}$, we compute the projections of the two OBBs and check if the intervals overlap.

For each axis $\mathbf{a}$, the projection interval $[p_{\min}, p_{\max}]$ of an OBB is computed as:

$$p_{\min} = \min(\mathbf{v}_i \cdot \mathbf{a}), \quad p_{\max} = \max(\mathbf{v}_i \cdot \mathbf{a}) \quad \text{for } i = 1, 2, \ldots, 8$$

where $\mathbf{v}_i$ are the vertices of the OBB and $\mathbf{a}$ is the separating axis.

If, for any axis, the projection intervals of the two OBBs do not overlap, the boxes do not intersect. If all intervals overlap, then the boxes intersect.

The SAT method is efficient and robust for detecting intersections between OBBs, as it reduces the problem to a series of 1D overlap checks along potential separating axes and allows for early returning as we know there is no collision as soon as one axis without overlap is found.

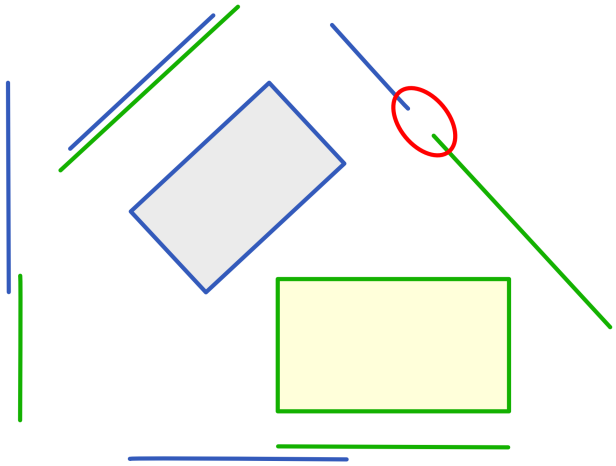


Figure 4.2.: SAT Method in 2D. No Overlap as one Axis is Separated.

4.2.4. Occupancy Grid - OBB Collision Detection

Two distinct methods for obstacle detection on occupancy grids have been implemented. One is GPU-accelerated, making it suitable for handling large numbers of bounding boxes simultaneously, while the other is a CPU-based variant, optimized for individual collision checks.

GPU-Accelerated Method

The GPU-accelerated method only offers significant performance improvements when processing multiple colliders [40]. The CPU method might be faster for single checks as the

communication time between CPU and GPU provides additional overhead. This method initially transfers the occupied grid cells, represented as a list of positions, to the GPU. For subsequent collision checks, only the bounding boxes are transferred. The GPU then processes each occupied cell in parallel, performing collision checks using the OBB-OBB method with SAT in 2D. Given the parallel nature of the GPU, this approach results in a maximum of $o \times b$ collision checks, where o represents the number of occupied grid cells and b represents the number of bounding boxes. While not achieving optimal performance, brute-forcing through all occupied cells has demonstrated sufficient efficiency for medium-sized maps. Although hierarchical data structures could be employed to enhance collision-checking efficiency, the current implementation favors simplicity and is suited for the scale of the problem.

CPU-Based Method

In contrast, the CPU-based method utilizes a quadtree data structure [42], which stores spatial data in two dimensions and is analogous to a segment or binary tree in one dimension and an octree [43] in three dimensions. By hierarchically grouping spatial data, the quadtree allows large, unused areas to be efficiently grouped, conserving memory. During collision detection, the algorithm recursively checks each node against the input bounding box. If a collision is detected at a given node, its child nodes are recursively checked until reaching a leaf node. The algorithm can terminate when a collision is detected at a leaf node. Similar Bounding volume hierarchies are often employed to test against multiple obstacles [44].

4.2.5. Collision Checking for Continuous Movement

To ensure safe traversal between two oriented bounding boxes (OBBs), we must perform continuous collision checking along the path from one OBB a, to another OBB b. This process can be achieved by discretizing the movement into small steps based on translation Δs and rotation $\Delta\phi$ influenced by the assumptions made in section 3.3.2.

At each step, an intermediate OBB is generated, which is then checked for collisions against the environment. Alternatively, all intermediate OBBs can be generated in parallel and evaluated together for potential collisions. The accuracy of the collision check depends significantly on the step size used for both translation Δs and rotation $\Delta\phi$.

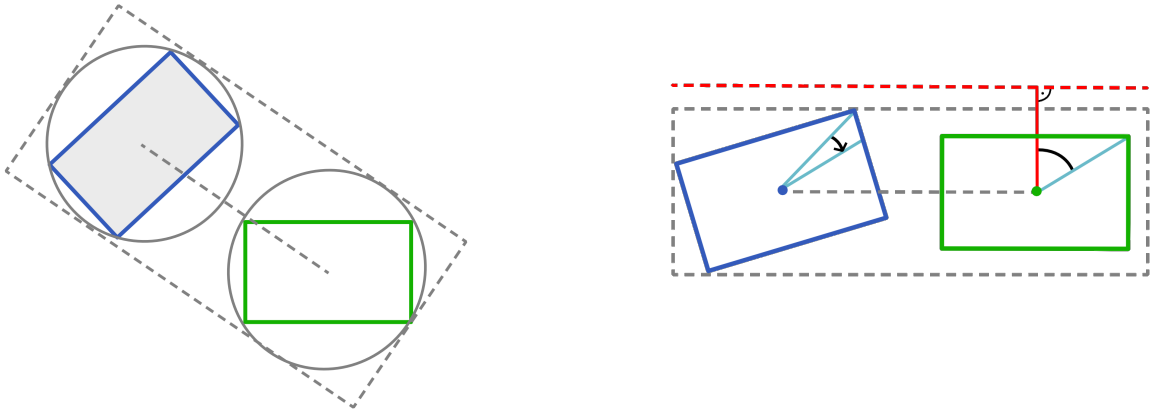
It is crucial to select appropriate values to avoid false negatives (i.e., missing a collision). These step sizes should be chosen in relation to the resolution of the occupancy map and the dimensions of the OBB. Larger OBBs or finer grid resolutions will require smaller step sizes to maintain precision in collision detection.

Performance Improvement

To further reduce the number of collision checks between two OBBs when no collision is detected, a heuristic approach can be applied. This involves generating a combined bounding box that encompasses both bounding boxes across the two poses (refer to Figure 4.3a). To avoid false negatives, the box width is determined by the maximum radius extending to the

OBB corners. If this combined box does not intersect with any obstacles, subsequent collision checks can be omitted.

However, since this approach may produce broad bounding boxes, which can be overly large, reducing effectivity in narrow corridors, a refined version is employed in this work. The refinement begins by calculating the angles at which the outer box's width and length reach their maximums. Following this, the minimum and maximum angles during the transition between the two poses are computed. By performing this calculation for each axis of the bounding box, the width and length can be optimized to minimize the bounding box dimensions along the transition path (refer to Figure 4.3b).



(a) Rough Heuristic Bounding Box

(b) Refined Heuristic for one Axis

Figure 4.3.: Heuristic Bounding Box for Pose Transitions. Minimal Bounding Box does not Require Full Radius Extents.

4.2.6. Other Intersection Methods

This work also implements additional intersection algorithms for different shapes like triangles and planes. However, these methods are not central to the path-planning process and are therefore not extensively discussed here for simplicity as their details are well-documented in the literature [41].

4.3. Mesh Processing

Mesh Processing aims to provide a data structure that can then be used in the path-planning process to check for the nearest obstacles or collisions. Unity already provides Navigation Meshes for this purpose, and the Recast library also enables dynamic updates during runtime when the mesh changes [15]. However, as these systems internally only support path planning for objects without rotation — effectively reducing such objects to a single point — they are unsuitable for this problem. Additionally, directly using the mesh with Unity's provided

collision checks is not feasible due to the lack of parallelization, which results in performance bottlenecks in scenarios requiring numerous collision checks.

To address these limitations, an intermediate octree-based spatial partitioning method is implemented to handle changes in the spatial mesh. This voxel grid is then converted into an accelerated 2D Distance Field, on which path planning can be performed. The additional intermediate octree representation has the advantage of offering 3D-collision checks with the already implemented collision methods and also eases the conversion to 2D maps as the world is already discretized into voxels.

4.3.1. Voxel Representation

Voxelization serves as the foundational step in the mesh processing pipeline. Each time a spatial mesh is added or updated, voxelization is applied, and the octree data structure — storing voxelized representations of all meshes — is subsequently refreshed. Each voxel encodes its spatial position, as well as the minimum and maximum mesh heights contained within that voxel (Figure 4.4). The voxelization algorithm utilized in this work follows a straightforward iterative procedure in which each mesh triangle is individually evaluated for voxel intersections within its bounding volume using SAT. Although more efficient GPU-based voxelization methods exist, they were not adopted here due to the small performance impact of this step within the overall pipeline. [45]

4.3.2. Octree Representation

All voxels are organized using an octree, hierarchically subdividing the space into progressively smaller regions [43]. Each leaf node in the octree, corresponding to an individual voxel, maintains a reference counter indicating the number of spatial meshes referencing the voxel (see Figure 4.4). This approach allows for the management of overlapping spatial meshes. When the reference count of a voxel reaches zero, the corresponding voxel can be removed safely from the octree.

Since individual voxel information is preserved (e.g., maximum and minimum mesh height), a more storage-efficient implementation — such as merging fully populated tree nodes — could not be utilized, as this would result in a loss of data.

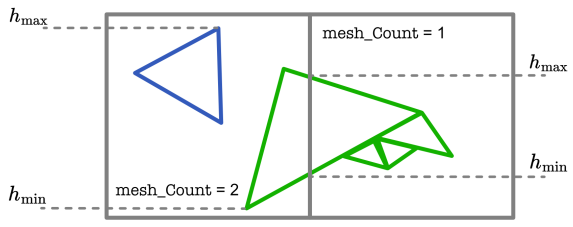


Figure 4.4.: Conceptual Representation of Two Voxels (Gray) with Two Meshes (Green and Blue), Storing Mesh Height and Mesh Count.

4.3.3. Occupancy Grid Representation

The final step before path planning involves converting the octree-based representation into a 2D occupancy grid.

Agent Interference

One challenge when converting a voxelized environment into an occupancy grid is that the agent may modify the spatial mesh and be included as part of the voxels. Therefore, all voxels in the octree intersecting with the agent are removed during preprocessing. The conversion algorithm then uses the bottom adjacent voxel as the starting points for occupancy grid generation.

Algorithm for Converting Voxel Grid to Occupancy Grid

The conversion process begins by defining key constraints, such as the minimum clearance height and the maximum incline angle for slopes between voxels. The approach involves iteratively processing voxels, starting from given positions. Neighboring voxels are then evaluated to determine if they can be reached while respecting the agent's constraints and, if so, added to the processing queue.

The following pseudo-algorithm outlines the key steps:

Algorithm 1 Conversion to Occupancy Grid

Input: O: Octree, s: Agent Position, h_{min}: Agent Height, θ_{max}: Max Incline
Initialize: Exclude agent voxels, initialize grid G
Add starting voxel v_s to queue Q
while Q is not empty **do**
    $v_c \leftarrow \text{Dequeue}(Q)$
    if $\text{Clearance}(v_c) < h_{min}$ **or** $\text{Slope}(v_c) > \theta_{max}$ **then**
        $G[v_c] \leftarrow$ occupied
        continue
    $G[v_c] \leftarrow$ free
    for each neighbor v_n of v_c **do**
        if $\text{Slope}(v_c, v_n) < \theta_{max}$ **and** v_n is unprocessed **then**
            Enqueue(Q, v_n)
        else
            $G[v_c] \leftarrow$ occupied
Output: Occupancy Grid G

Limitations

This approach has several limitations. For instance, it does not support environments with multiple stacked floors, and it imposes additional CPU overhead, as the occupancy grid must

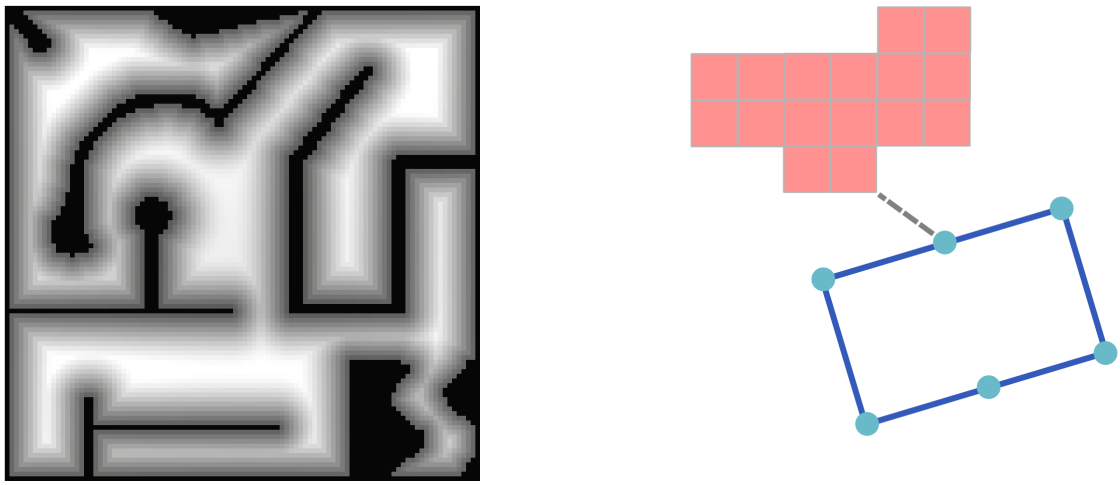
be regenerated prior to each path planning process if the spatial mesh has been modified. For multi-level structures, alternative approaches such as multi-floor occupancy grids may be implemented [46].

4.4. Signed Distance Field

Distances to the nearest obstacle might also be queried in the path-planning process. For this purpose, it is crucial to have an efficient method to query the proximity of nearby obstacles. Distance fields have been demonstrated as a valuable asset in computer vision, particularly in the context of surface reconstruction and obstacle detection, and they have also been widely adopted in path planning applications [17]. Consequently, during the map generation process, an Euclidean Distance Field (EDF) is constructed by performing a nearest neighbor search across all unoccupied cells within the map, with the outcomes stored in a hash map to enable $\mathcal{O}(1)$ access (an example EDF can be seen in Figure 4.5a). Alternative, more efficient conversion methodologies employ image transformations, resulting in enhanced performance [47].

4.4.1. Distance Calculation Between the Agent and Obstacles

To determine the nearest distance between the agent and obstacles using the EDF, the agent's bounding box is uniformly sampled at regular intervals. The minimum distance across all sampled points is then calculated to establish proximity to the nearest obstacle (see Figure 4.5b). Special cases, such as obstacles located within the agent's bounding volume, are not addressed, as it is assumed that the agent has already navigated through obstacles to reach its current position. Therefore, sampling exclusively along the bounding box boundary is considered adequate for this approach.



(a) Euclidean Distance Field

(b) Bounding Box Proximity Check

Figure 4.5.: Distance Fields and Proximity Check with Sampling the Bounding Box

4.5. Path-Planning

The algorithm for path-planning employed in this work is a hybrid of various methods, primarily based on the RRT* algorithm. Initially, the fundamental concept is introduced, followed by concrete implementation.

4.5.1. RRT* Algorithm

The core concept of RRT* is to explore the configuration space by generating random samples, assessing valid transitions, and incrementally building a tree structure. For each new sample, RRT* seeks to connect it to the existing tree through the shortest possible path, updating the tree to store the optimal path to each sample and iteratively improving the overall path to the target.

Algorithm 2 Pseudo RRT* Algorithm [48]

Input: x_{init}: initial configuration
Initialize: $V \leftarrow \{x_{\text{init}}\}$, $E \leftarrow \emptyset$
for $i \leftarrow 1$ to n **do**
    $x_{\text{rand}} \leftarrow \text{RandomSample}()$
    $x_{\text{nearest}} \leftarrow \text{Nearest}(V, x_{\text{rand}})$
    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$
    if $\text{CollisionFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
        $X_{\text{near}} \leftarrow \text{NearestNodes}(V, x_{\text{new}})$
        $V \leftarrow V \cup \{x_{\text{new}}\}$
        $x_{\text{min}} \leftarrow x_{\text{nearest}}$
        $c_{\text{min}} \leftarrow \text{Cost}(x_{\text{nearest}}) + \text{Cost}(x_{\text{nearest}}, x_{\text{new}})$
        for $x_{\text{near}} \in X_{\text{near}}$ **do** ▷ Find Closest Node
            if $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}})$ **and** $\text{Cost}(x_{\text{near}}) + \text{Cost}(x_{\text{near}}, x_{\text{new}}) < c_{\text{min}}$ **then**
                $x_{\text{min}} \leftarrow x_{\text{near}}$
                $c_{\text{min}} \leftarrow \text{Cost}(x_{\text{near}}) + \text{Cost}(x_{\text{near}}, x_{\text{new}})$
        $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$
        for $x_{\text{near}} \in X_{\text{near}}$ **do** ▷ Rewire the tree
            if $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}})$ **and** $\text{Cost}(x_{\text{new}}) + \text{Cost}(x_{\text{new}}, x_{\text{near}}) < \text{Cost}(x_{\text{near}})$ **then**
                $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}})$
                $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
Output: $G = (V, E)$

At each iteration, a new random sample x_{rand} is generated, followed by the creation of a new configuration x_{new} by steering from $x_{nearest}$ toward x_{rand} with a predefined step size. Subsequently, valid neighboring nodes X_{near} within a radius determined by the size of the tree are queried. An edge is then established from the node x_{min} (the node offering the minimal

cost) to x_{new}. For all remaining nodes, the tree is rewired from x_{new} whenever the recalculated cost to any x_{near} is less than the previously recorded cost, provided that a valid, collision-free transition is possible:

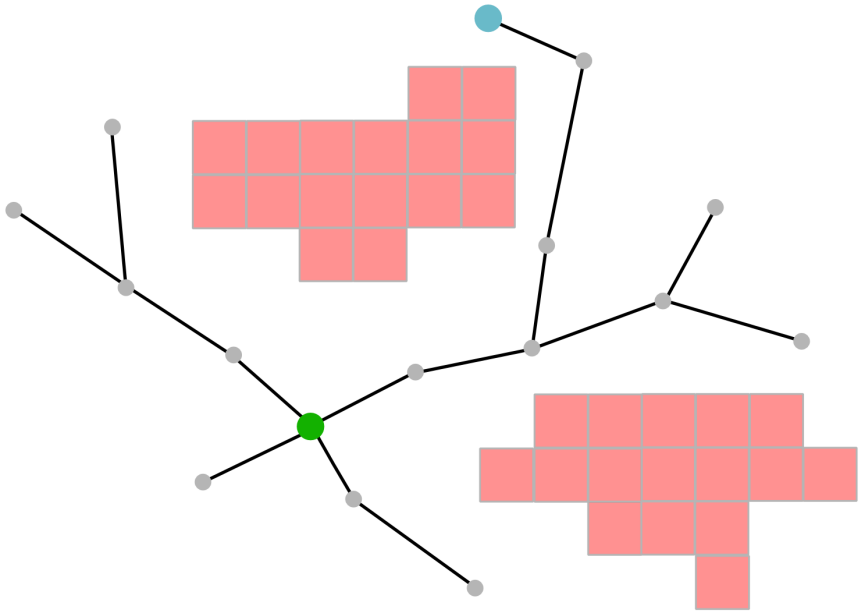


Figure 4.6.: RRT* Algorithm Finds Path between two Obstacles

4.5.2. Improved Sampling

As random state sampling comes with a high number of samples needed, a big variance in search time, and bad performance on narrow passages, other sampling methods are explored in this work [24]. To improve this critical part, 3 different optimization methods are being used.

Goal-Biased Sampling

One key approach in enhancing exploration within sampling-based motion planning is the integration of goal-biased sampling. This encourages the planner to expand towards the goal by prioritizing close nodes and thereby facilitating a more efficient search [49]. However, this method introduces a risk of becoming trapped in local minima, as the nearest feasible path to the goal may initially diverge from the target configuration. To encounter this effect, a randomized offset around the goal is introduced to mitigate the likelihood of local minima: $x_{\text{rand}} = x_{\text{goal}} + x_{\text{random\_offset}}$.

A* based Sampling

Also, combining the A* algorithm with RRT*, as discussed in previous works [22, 23], can offer significant improvements in convergence. Although A* is computationally and memory intensive, particularly in high dimensional spaces, its selective use in proximity to the goal has been shown to accelerate convergence and enhance the overall efficiency of the pathfinding process [23].

Inspired by this idea, the proposed sampling strategy incorporates A*, already at the initial stage of path planning. This approach allows for the early generation of a preliminary path in a simplified configuration space. The method begins by applying A* with a relatively coarse grid, restricting the configuration space to two dimensions and focusing solely on the robot's position. Each node in this space must maintain a minimum clearance from obstacles, ensuring the path is at least as wide as the agent's minimum expansion radius.

Additional costs are assigned to nodes positioned close to obstacles to further optimize the path. This incentivizes the algorithm to favor paths that maintain a safer distance from potential collisions, enhancing both path safety and feasibility. If the A* algorithm cannot find a path under these conditions, it conclusively indicates that no feasible path exists. However, in cases where A* successfully generates a path, there remains a possibility that the path might not be feasible in practice due to dynamic constraints or environmental factors.

The key aspect of this sampling strategy is the selection of a random node from the generated path within a certain radius, as depicted in the following Figure 4.7:

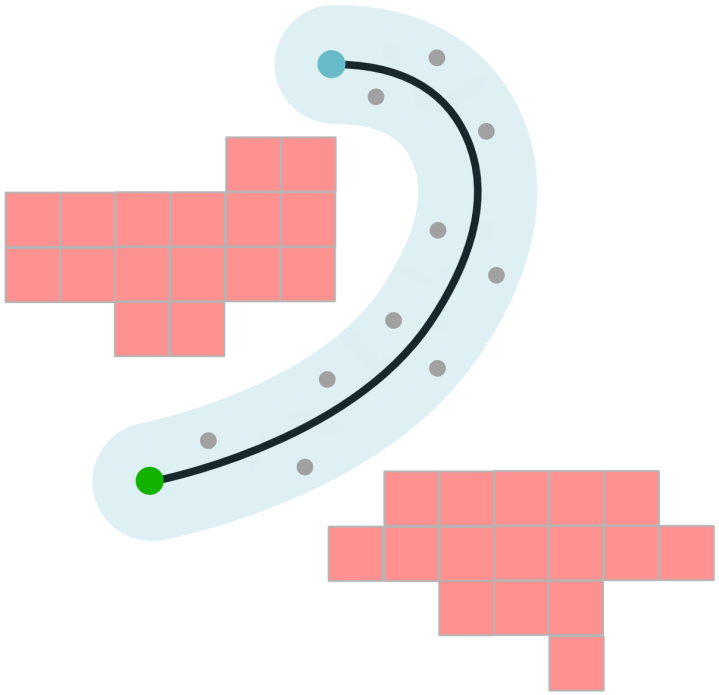


Figure 4.7.: A* Sampling with Random Configurations Generated in Proximity to the Path

Density based Sampling

Another issue that arises with random sampling is the potential formation of disproportionately dense node clusters. Ideally, less dense regions should be explored first to accelerate the overall exploration process. To address this, the Fast-RRT algorithm has been proposed [24].

The core idea of Fast-RRT is that for each newly generated configuration x_{rand}, the node density in its vicinity is evaluated. The configuration is discarded if the local density exceeds a predefined threshold, and a new sample is generated [24]. This mechanism promotes more balanced exploration by prioritizing less dense areas of the configuration space, thereby improving the efficiency of the sampling process.

In our implementation, density is defined as the number of explored nodes in a given radius $r_{max} = 0.4[m]$ and a rotation less than $\phi_{max} = 30^{\circ}$.

Combination of Proposed Sampling Methods

To mitigate the individual limitations of each sampling technique, all presented methods are combined into a unified sampling strategy. In each iteration, one of the random sampling methods is selected to generate x_{rand}.

Subsequently, a density correction is applied to x_{new}. If the local density exceeds a predefined threshold, the sample is discarded, and the next iteration begins.

This combined approach leverages A*-based sampling and goal-biased sampling to facilitate rapid exploration toward the goal, while density-based sampling acts as a mechanism for random exploration, helping to escape local minima.

Optimized Sampling Strategies

As sampling plays a critical role in accelerating convergence, adapting the strategy to reflect both the typical geometry of the environment and the physical constraints of the agent is beneficial. For instance, the Loop-X, with its near-rectangular structure, faces difficulties executing full turns in confined spaces. In response to these constraints, a sampling distribution of 10% Goal-Biased, 50% A*, and 40% Random Sampling has been found to yield good performance under these conditions. Eventually, dynamic adjustment of these values in response to environmental factors might be beneficial to achieve maximum efficiency.

4.5.3. Cost Calculation

Another critical aspect of the path-planning process is the cost of traversing between two nodes a and b. In our approach, the path is influenced by the operational characteristics of the Loop-X system. The primary objective is to minimize the time required for the robot to traverse between nodes, which is determined by the time needed for wheel adjustments and the travel speed.

In addition to time efficiency, it is desirable to maintain a safe distance from obstacles, avoiding close encounters. Therefore, the cost of moving from a to b is affected by the

Euclidean distance traveled, the proximity to an obstacle, and the orientation change required between the two nodes, which yields a complex cost space for navigation.

Given these constraints, the following cost function is used:

$$\text{Cost}(a,b) = d(a,b) \cdot f_d + f_o(d_{\text{obs}}(a,b)) + |\theta(a) - \theta(b)| \cdot f_\theta + w_{\text{adj}} \tag{4.1}$$

Where:

- $d(a,b)$ is the Euclidean distance between node a and node b,
- $d_{\text{obs}}(a,b)$ is the minimum distance to an obstacle along the path from a to b,
- $|\theta(a) - \theta(b)|$ is the absolute difference in orientation between node a and node b,
- f_d is the weight for the distance traveled,
- f_o is the weight function that yields the cost given the distance to the nearest obstacle,
- f_θ is the weight for orientation changes,
- w_{adj} is the wheel adjustment offset.

4.5.4. Performance Improvements

Collision detection remains a key factor influencing the computational performance of RRT*. A particularly resource-intensive operation is the rewiring step, where multiple collision checks are required for each neighboring node. This process often includes nodes that are distant from the optimal path, which can lead to unnecessary computation and inefficiencies. One approach to mitigate this issue is to defer the rewiring process until a feasible path has been identified. A method similar to the "rewiring from the root node" as implemented in the RT-RRT* algorithm has been adopted:

Algorithm 3 Rewiring From the Tree Root [50]

Input: Q_s: Queue of nodes, $G = (V, E)$: Tree as graph
if Q_s is empty **then**
    Enqueue(Q_s, x_0)
while Q_s is not empty **do**
    $x_s \leftarrow$ Dequeue(Q_s)
    $X_{\text{near}} \leftarrow$ NearestNodes(G, x_s)
    for $x_{\text{near}} \in X_{\text{near}}$ **do**
        $c_{\text{old}} \leftarrow \text{Cost}(x_{\text{near}})$
        $c_{\text{new}} \leftarrow \text{Cost}(x_s) + \text{Cost}(x_s, x_{\text{near}})$
        if $c_{\text{new}} < c_{\text{old}}$ **and** ObstacleFree(x_s, x_{near}) **then**
            $E \leftarrow (E \setminus \{\text{Parent}(x_{\text{near}}), x_{\text{near}}\}) \cup \{x_s, x_{\text{near}}\}$
        if x_{near} is not in Q_s **then**
            Enqueue(Q_s, x_{near})

An additional enhancement is introduced in our implementation: only nodes promising a more optimal path are considered for insertion into Q_s. This selective rewiring strategy prevents the entire graph from being unnecessarily rewired, focusing instead on nodes proximal to the optimal solution. Though differing in implementation, the core concept of this method aligns with the Lazy-RRT variant's approach to bypassing collision checks [51].

4.5.5. Path Refinement

As the initial path obtained from RRT* is far from optimal — due to our implementation halting as soon as the target is reached — the next logical step is to refine the path to bring it closer to an optimal solution. Path refinement is essential to improve the overall cost and efficiency of the solution, particularly in high-dimensional spaces where RRT* tends to converge slowly [48]. Several approaches can be employed to minimize the path's cost, which is grounded in enhancing the sampling strategies and post-processing refinement techniques.

Extend Runtime

The first and most straightforward approach is to extend the run-time of the RRT* algorithm. By continuing the exploration after a feasible path has been found, the algorithm can eventually converge to a near-optimal solution. While effective in theory, this technique can be computationally expensive, especially in high-dimensional configuration spaces where the convergence rate of RRT* is inherently slow.

Informed RRT*

A more sophisticated and efficient approach in motion planning is the Informed RRT* algorithm, which refines the sampling process following the discovery of an initial path. Informed RRT* constrains subsequent sampling to a hyper ellipsoid that encloses the start and target configurations, thereby concentrating the search space on regions with a higher probability of yielding improved solutions. This method exploits the geometric properties of the planning problem to enhance convergence rates while preserving the asymptotic optimality of RRT*. [20]

In our application, identifying the precise hyperellipsoid can be challenging. As a practical alternative, we employ a simplified version of Informed RRT* that samples within a designated radius around points along the shortest path. This radius is iteratively reduced, converging toward a locally optimal solution. Although this modified approach lacks a formal guarantee of convergence to the global optimum, it has demonstrated satisfactory performance across typical scenarios in this work.

Path Triangulation

A further technique for path optimization involves triangulating the path incrementally, as proposed in the RRT*-Smart algorithm [25]. At each iteration, nodes are evaluated to determine if they can be rewired to their respective grandparent nodes. If this rewiring

reduces the overall path cost, the connection to the grandparent node is made, resulting in the formation of *beacon nodes*, which act as key points for optimizing path cost and structure:

Algorithm 4 Path Optimization [25]

1: **Input:** X_{init}, X_{goal}
2: **Initialize:** $X_{\text{vc}} \leftarrow X_{\text{goal}}$
3: **while** $X_{\text{vc}} \neq X_{\text{init}}$ **do**
4: **if** ObstacleFree$(X_{\text{vc}}, \text{GrandParent}(X_{\text{vc}}))$ **then**
5: Parent$(X_{\text{vc}}) \leftarrow$ GrandParent(X_{vc})
6: **else**
7: $X_{\text{vc}} \leftarrow$ Parent(X_{vc})

The triangulation process, as illustrated in Algorithm 4 and Figure 4.8, promotes path efficiency by enhancing the probability of sampling configurations near beacon nodes, which reduces the cumulative path cost.

However, in this specific context, a direct implementation of the proposed algorithm is not feasible without an additional cost check. Specifically, it is necessary to verify that the rewiring step from the grandparent node to the current node actually reduces the path cost, as the triangular inequality may not hold consistently in complex cost spaces.

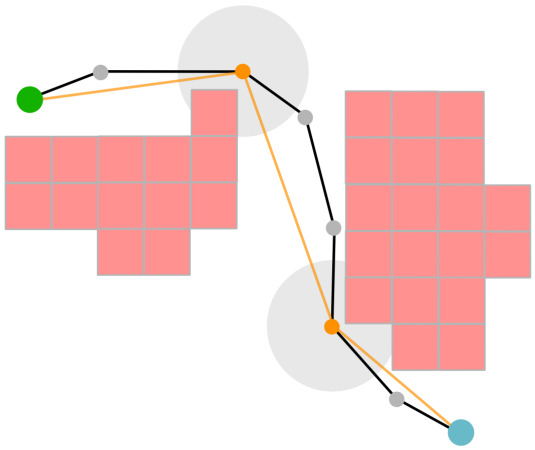


Figure 4.8.: RRT* Path Optimization with Beacon Nodes (Orange)

4.6. Loop-X Integration

Communication gets established via a UDP interface, requiring the device to connect to the Wi-Fi network hosted by the Loop-X. This connection enables the system to dispatch motion commands, either autonomously or with explicit user confirmation. Commands are issued

relative to the robot's local coordinate system, allowing for precise control over movement and providing functionality to pause and resume as necessary.

4.6.1. Transforming Unity Coordinates to Loop-X Coordinates

To accurately transmit motion data to the Loop-X, it is crucial to address the differences in coordinate systems between Unity and Loop-X. Unity operates in a Y-up, left-handed 3D coordinate system, while the Loop-X uses a 2D coordinate system where the X-axis is inverted relative to Unity, and Unity's Z-axis aligns with the Y-axis.

To perform this conversion, consider two consecutive waypoints, wp_i and wp_{i+1}. The transformation begins by defining matrices from the Unity origin to each waypoint. The transformation matrix from the Unity origin to the waypoint at index i, denoted as ${}^{U}\mathbf{T}_{\text{wp}_i}$, and the transformation matrix to the waypoint at index $i+1$, denoted as ${}^{U}\mathbf{T}_{\text{wp}_{i+1}}$, are computed as follows, where U stands for the Unity origin frame:

$$ {}^{U}\mathbf{T}_{\text{wp}_i} = \begin{bmatrix} R(\theta_{\text{start}}) & \mathbf{p}_{\text{start}} \\ 0 & 1 \end{bmatrix} $$

$$ {}^{U}\mathbf{T}_{\text{wp}_{i+1}} = \begin{bmatrix} R(\theta_{\text{target}}) & \mathbf{p}_{\text{target}} \\ 0 & 1 \end{bmatrix} $$

The relative transformation between the waypoints wp_i and wp_{i+1} is then calculated by inverting the matrix for wp_i and applying it to the matrix for wp_{i+1}:

$$ {}^{\text{wp}_i}\mathbf{T}_{\text{wp}_{i+1}} = \left({}^{U}\mathbf{T}_{\text{wp}_i}\right)^{-1} \cdot {}^{U}\mathbf{T}_{\text{wp}_{i+1}} $$

To map this transformation into the Loop-X coordinate frame, the transformation from the Loop-X origin to its current position, denoted as ${}^{L}\mathbf{T}_{\text{LoopX}}$, is retrieved via the UDP interface. The final position and rotation relative to the Loop-X origin are computed as following, where L will be used to denote the Loop-X origin frame:

$$ {}^{L}\mathbf{T}_{\text{wp}_{i+1}} = {}^{L}\,\mathbf{T}_{\text{LoopX}} \cdot {}^{\text{wp}_i}\,\mathbf{T}_{\text{wp}_{i+1}} $$

Since both Unity and Loop-X use meters and degrees for positional and angular measurements, no unit conversion is necessary.

4.6.2. Sending Motion Execution

Due to the current limitations of the robot's UDP interface, which does not support direct control over the robot's wheels, the robot executes movements step by step for each waypoint before proceeding to the next. This limits the ability for continuous motion across waypoints and leverages the importance of paths minimizing the total number of waypoints.

At each waypoint, the algorithm performs two main actions:

1. **Enqueue Pose**: First, the target pose is enqueued to allow Loop-X to calculate the new wheelbase and internal configuration required to achieve the desired position and orientation.

2. **Execute Movement**: After a short 1-2 seconds delay, an "Execute Movement" command is sent, which triggers the robot to move towards the enqueued pose. This delay allows LoopX to complete internal calculations and prepare for movement.

The system also supports functionality to *pause* and *resume* motion at any time, which enables the operator to react to moving obstacles. The flowchart in Figure 4.9 gives an overview of the communication process:

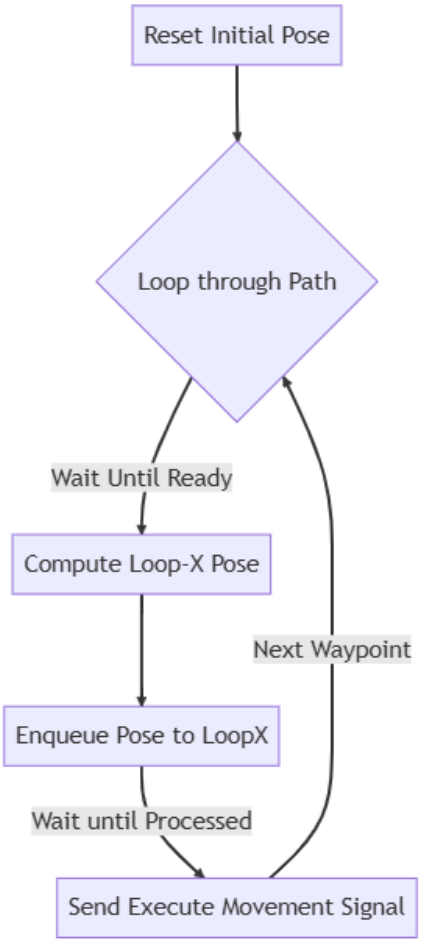


Figure 4.9.: Flowchart of Sending Motion Data to the Loop-X

4.7. User Interface Design Choices

The User Interface (UI) serves as the primary medium through which the user interacts with the application, enabling the specification of waypoints, path generation, and controlling the Loop-X to follow the generated path. The UI is designed to require minimal user experience while providing immediate feedback to facilitate efficient interaction in a clinical setting. Also, the "Microsoft style" is kept by utilizing MRTK UX components with a similar color scheme. A comprehensive understanding can be gained by viewing the provided application showcase (A.3).

4.7.1. User Interaction

The MRTK framework offers a range of prefabs, such as slates, buttons, and other UI elements, which can be readily used in the editor to accelerate both development and ensure a consistent user experience across applications. Additionally, object manipulators allow for intuitive hand-based interactions with virtual objects, enabling movement, translation, and scaling using one or both hands. Moreover, eye gaze and ray-based interactions guide users through the virtual environment or menus by highlighting specific fields and enabling interaction with distant objects.

4.7.2. Menus

The interface offers two distinct menus that users can access based on their interaction needs. The first is a hand-attached menu, which provides quick access to essential features, while the second is a larger menu designed for more comprehensive path-planning settings.

Hand-Menu

The hand-attached menu provides quick access to essential features, reducing the need for extensive navigation through multiple screens. Users activate the menu by looking at their open flat hand, presenting an intuitive interface for core system functions [52]. The primary functions offered in the hand menu, as illustrated in Figure 4.10a, include:

- **Show Main-Menu**: Activates the main menu, which allows the user to access a broader range of path-planning options and system settings.
- **Abort Loop-X Motion**: This action immediately halts the robot's motion for safety purposes. It is designed for quick access, ensuring that the user can stop the robot in emergency situations or if an unexpected event occurs.
- **Activate Help-Menu**: Opens the help menu, which displays the user manual and instructional guidelines.

Main-Menu

The main menu interface consists of a slate divided into a top bar and a canvas containing the menu options.

The top bar is structured with several buttons, arranged from left to right in Figure 4.10b, providing access to key functionalities:

- **Home**: The starting point for initiating path planning.
- **Path Visualization**: Displays a visualization of the Loop-X following the generated path.
- **Settings**: Provides access to various settings, including communication preferences.
- **Attach/Detach**: A toggle function to attach or detach the menu to the user's head, enabling a following menu that moves with the user.
- **Close**: Closes the menu window.

(a) Hand-Menu

(b) Main Menu Interface

Figure 4.10.: Hand-Menu and Main-Menu

Path-Planning View

The path planning view, as shown in Figure 4.10b, is divided into three sections: the waypoint section, the action section, and the log section. To begin path planning, the user can add waypoints by pressing the button located at the bottom left of the interface. Then, a virtual 3D model representing the physical robot is created in front of the user, and a corresponding entry is added to the waypoint section as shown in Figure 4.11.

Waypoint Section
The waypoint section serves as the primary interface for managing the waypoints. Each added waypoint generates an entry that consists of five action buttons arranged from left to right:

- **Toggle Visuals**: Toggles the visibility of the waypoint in the 3D environment.
- **Toggle Editing**: Enables or disables the ability to edit the waypoint.
- **Toggle Waypoint Mode & Height Adjustment**: The top button controls whether the waypoint is included in path generation, while the bottom button specifies if the waypoint is auto-aligned with the floor or allows the user to manually adjust its height.
- **Delete**: Removes the waypoint from the path planning process.

Figure 4.11.: Waypoint Entry Interface

Toggling visibility helps reduce visual clutter, especially in complex environments. Disabling editing after adjusting the position and rotation of a waypoint prevents accidental changes. Height adjustment is particularly useful for the first waypoint, as spatial mesh data may not always be accurate, and the exact position on the floor might not be determined correctly. The waypoint mode decides whether or not the waypoint is being used in path planning. This enables manual path specification in environments where path planning cannot be achieved directly.

Action Section
Once waypoints have been added, the user can generate a path using the top button in Figure 4.10b. A progress indicator is displayed beneath the button to provide feedback during the path generation process. The following two buttons can resume the robot's movement along the generated path or immediately pause the robot's motion in case of any issues or emergencies. The last button displays the connection status of the Loop-X, using color to indicate the state: green for connected and red for disconnected. Clicking this button redirects the user to the settings menu, where connection configurations can be managed.

Log Section:
The log section serves as a means of providing the user with real-time information about

ongoing processes and system status. It displays notifications about issues that may arise during path generation or map creation and contains system status updates. It can also be used for debugging purposes when turning on the developer mode in the settings.

Motion Preview
Once a path has been successfully generated, the motion of the Loop-X can be previewed in the motion preview panel (see Figure 4.12). In this visualization, the outlined hologram represents the designated waypoints, while the more solid hologram previews the Loop-X's anticipated movement along the path.

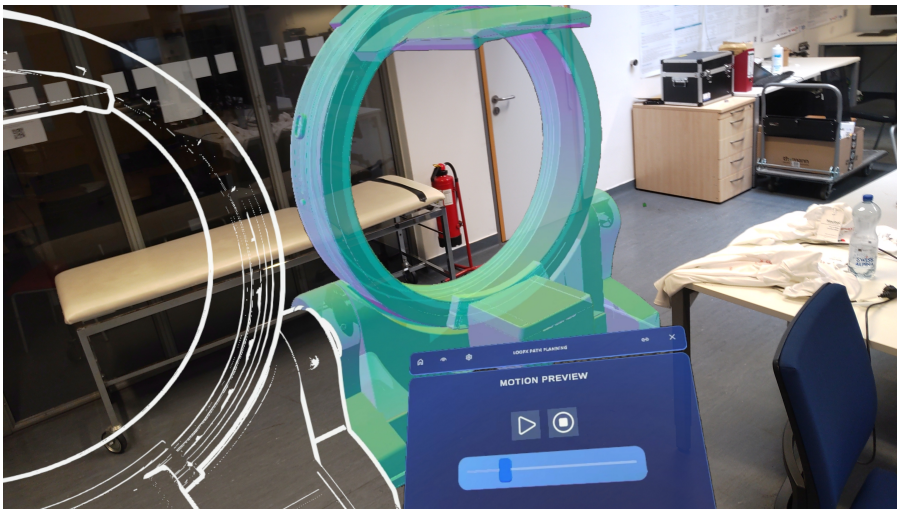


Figure 4.12.: Motion Preview Panel

The system provides two movement control methods. By default, the planned motion sequence executes automatically. Alternatively, users can manually adjust the position along the path with a slider, enabling detailed inspection of close encounters with obstacles and potential collision points.

Settings
The final section of the user interface is the settings panel, where global application configurations can be adjusted and debugging information can be retrieved.

The settings panel is organized into four columns, each grouping related functionalities:

Visualization: The first section allows enabling different visualizations like showing generated voxels (see Figure 4.14a), display the spatial mesh (see Figure 4.14b), or an overlay of the generated map onto the floor to distinguish walkable and non-walkable areas (see Figure 4.14c).

Map: The generated map can be viewed in a dedicated window (see Figure 4.14d), and there is an option to regenerate it starting from the current position.

Tracking: This section provides information on whether the Loop-X is currently being tracked. Users can enable or reset tracking as needed. Additionally, it is possible to specify whether the first waypoint is implicitly set by the tracked position or if a manual waypoint should be used.

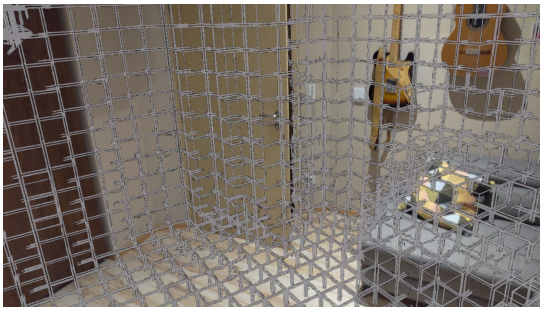
Loop-X Connection & Debugging: In this section, the connection to the Loop-X device can be established by providing its IP address and Port. Debug logging can be enabled, and the path generation variant can be chosen. Additionally, map generation can be frozen once a good map has been generated.

Figure 4.13.: The Settings Panel

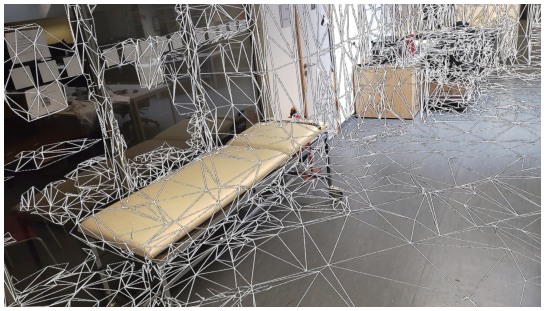
4.7.3. Hologram Visuals and Spatial Placement

The accurate placement and visual representation of holograms, particularly three-dimensional objects, are essential for ensuring seamless integration with the physical environment in MR systems [53]. In this context, the holographic depiction of waypoints and path-following simulations is accomplished through ghost models (see Figure 4.12). These enable users to interact with virtual objects at a real-world scale while still preserving clear visibility of the physical surroundings.

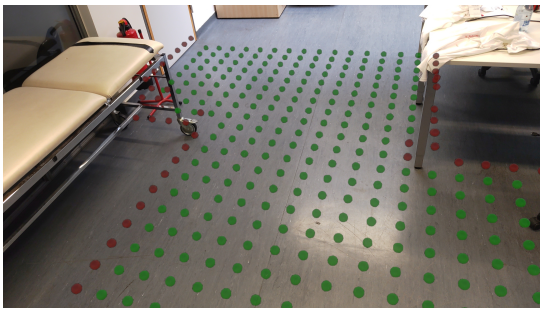
The waypoint model strategically implements visual indicators to enhance user experience and functionality. For instance, a warning icon appears if a waypoint intersects with the spatial mesh, accompanied by additional markers at intersection points to alert users to potential placement conflicts. Moreover, another icon indicates whether the waypoint is currently editable or locked.



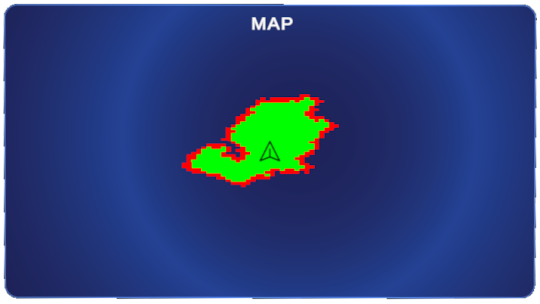
(a) Voxel Visualization



(b) Spatial Mesh Display



(c) Map Overlay Visualization



(d) Dedicated Map Window

Figure 4.14.: Collection of Visual Elements Related to the Settings Panel, Including Voxel Visualization, Spatial Mesh Display, Map Overlay, and the Dedicated Map Window.

4.7.4. 3D User Interaction

The editing of waypoints, encompassing both their translation and rotation, is supported by spatial manipulation functionalities available within the MRTK Toolset. The Object Manipulator enables users to adjust waypoints by using single or dual-hand interactions to control the position and Y-axis orientation within the 3D space [54]. Remote waypoints may also be repositioned through ray-based interaction, minimizing the physical effort required [32]. To enhance interaction precision, a subtle damping effect is applied, which mitigates unintended abrupt movements or rotations. Furthermore, waypoints are automatically aligned with the detected ground plane, ensuring spatial consistency and accuracy in their placement.

4.7.5. Feedback and Alerts

Consistent with the design principle of delivering comprehensive user feedback [55], the application employs the MRTK 3 dialog component to convey critical information to users (see Figure 4.15). For instance, alert notifications appear before the commencement of path-following operations when path planning is requested but the Loop-X device is not actively

tracked. Additionally, confirmation messages are displayed once the Loop-X is successfully tracked. These notifications aim to reduce the likelihood of unwanted actions, ensuring that users are fully informed of the system's operational status during essential phases.

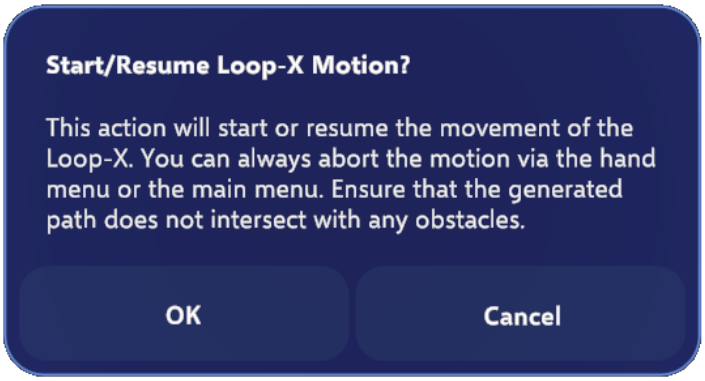


Figure 4.15.: Example Alert

5. Discussion

This chapter focuses on analyzing and interpreting the results obtained from the implementation and evaluation of the HoloLens 2-guided clinical robot system.

5.1. Enhanced RRT Methodology

During testing, occasional failures in path planning were observed, primarily due to minor errors and artifacts present within the map. To address this, a modification to the RRT (Rapidly-exploring Random Tree) algorithm was explored. This adjustment involves allowing collisions without fully inhibiting the path planning process; instead, collisions are assigned high-cost values within the cost function. This modification requires minimal alteration to the original RRT algorithm, as it merely updates the cost function to impose a higher penalty upon encountering obstacles while collision checks are bypassed. In the subsequent lab study, this approach is referred to as RRT* Method 2.

5.2. Test Scenarios and Use Cases

For testing path planning functionality, one virtual test and one lab study were performed for the application, with detailed results provided in A.2.

5.2.1. Virtual Testing

To assess the effectiveness of path-planning within a simulated environment, a virtual occupancy grid was implemented, as depicted in Figure 5.1. This virtual playground, spanning 16x16 meters, served as a controlled setting for testing algorithmic performance. The RRT* algorithm was evaluated with a maximum limit of 2500 iterations, with scenarios exceeding this threshold marked as failures. For each test case, the success rate and average iteration count were documented. Performance metrics for different paths are presented in Table 5.1.

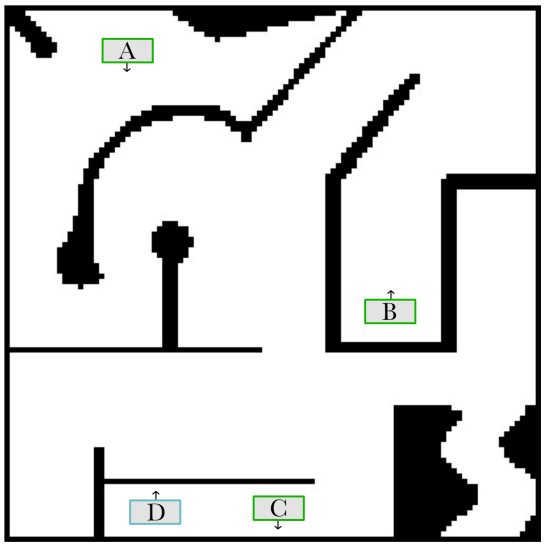


Figure 5.1.: Playground (16x16 m)

Waypoint (From → To)	Average Success Rate	Average Iterations
A → D	80%	1992
B → D	70%	1943
C → D	50%	1864
A → B	100%	1412

Table 5.1.: Virtual Experiment Results for Various Paths

Path variant A → B has the highest success rate as rotation does not play a big role, as turning is also possible at last waypoint. Consequently, the algorithm can rely on the A* path, achieving a 100% success rate with an average of only 1412 iterations. However, challenges arise when rotational choices become important, particularly in paths where later adjustments to orientation are restricted. This is visible in paths A → D and B → D, where the rotation on intermediate waypoints is crucial for entering the narrow corridor at the bottom.

In path C → D, where only 50% of trials yielded a path within the 2500-iteration, the current sampling method reaches its limit. Because of the opposite orientation of C and D, the agent must first maneuver backward to reach an area where rotational adjustments are feasible. Figure 5.2 illustrates such a failing scenario, highlighting the explored graph nodes and also A* path nodes in red. It can be seen that the density-randomized sampling demonstrates the ability to explore other areas of the map. However, the probability of randomly generating nodes leading back to the target is low. This challenge becomes more pronounced as waypoint C moves closer to D. Therefore, many samples are needed for such cases. Potential solutions include refining the random exploration sampling strategy and reducing the weight of the A* sampling strategy in such cases. Introducing a Voronoi bias into the sampling process may also alleviate these issues [56].

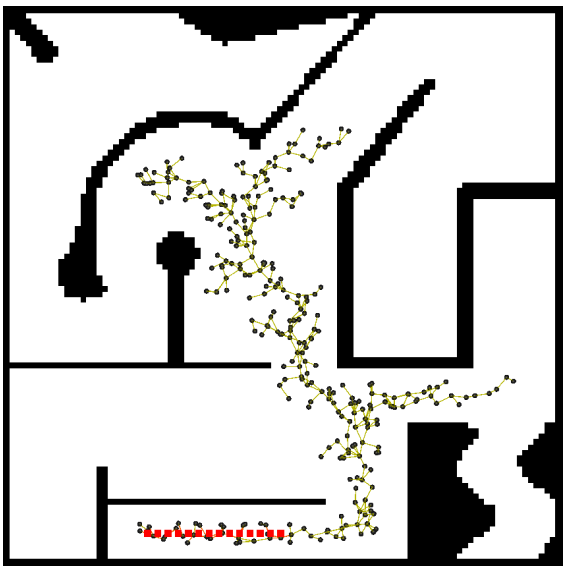
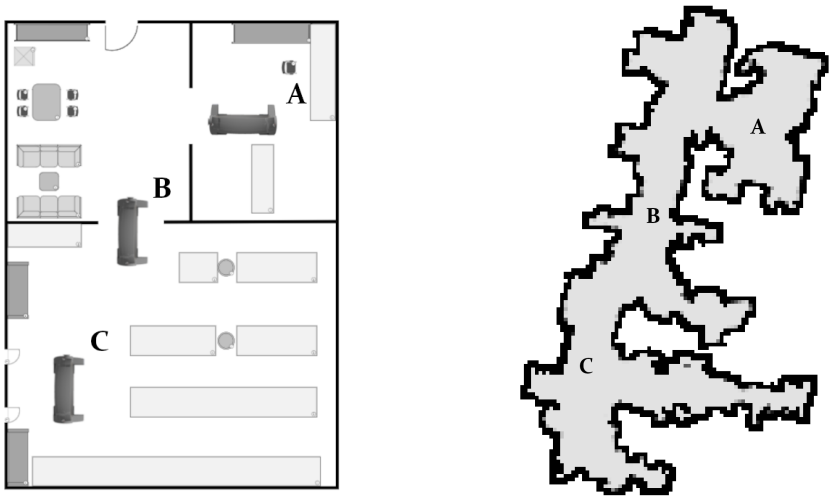


Figure 5.2.: RRT Graph and A* Path Nodes in red for a Failing Path C → D

5.2.2. Lab Study

The lab study was conducted within the IFL Lab at Klinikum Rechts der Isar following a comprehensive scan of the entire room. Various short-distance configurations, with a maximum distance of 15 meters, were examined under these conditions. Figure 5.3 presents a conceptual room plan illustrating the waypoints used in the experiments, along with the corresponding occupancy grid generated from the scan.



(a) Conceptual Room Plan of the IFL Lab

(b) Occupancy Grid of the Scanned IFL Lab

Figure 5.3.: Room Plan and Occupancy Grid of the IFL Lab

The test cases involved navigating between waypoints A to B and A to C, with the longest path covering approximately 15 meters. Notably, this experiment requires the agent to perform a rotation at a specific point along the path. Since rotation between points A and B is not feasible from the initial position, the path planner must first direct the agent along a route that allows for rotation. This maneuver can, for example, be accomplished by navigating toward the upper door, where the agent can then alter its direction.

Evaluation

For each experiment, 12 repetitions were performed. The success rate (i.e., the ratio of successfully generated paths that avoided obstacles to the total number of trials) and the average path generation time were recorded. For *Method 1*, a maximum of 2500 iterations was permitted, while *Method 2* was limited to 2000 iterations due to the increased runtime caused by its more complex cost function. The results, detailing the success rates and average path generation times for both RRT* Method 1 and RRT* Method 2, are presented in Table 5.3:

Measurement A → B	RRT* Method 1	RRT* Method 2
∅ Time (s)	5	8
∅ Remote Time (s)	< 2	/
∅ Iterations	1168	2000
Success Rate	75%	25%

(a) Path Planning Performance (A to B)

Measurement A → C	RRT* Method 1	RRT* Method 2
∅ Time (s)	6	8
∅ Remote Time (s)	< 2	/
∅ Iterations	1481	2000
Success Rate	67%	0%

(b) Path Planning Performance (A to C)

Table 5.2.: Comparison of Path Planning Performance between RRT* Method 1 and RRT* Method 2 for Two Paths

S

Measurement A → B	RRT* Method 1
∅ Time (s)	5
∅ Remote Time (s)	< 2
∅ Iterations	1168
Success Rate	75%

(a) Path Planning Performance (A to B)

Measurement A → C	RRT* Method 1
∅ Time (s)	6
∅ Remote Time (s)	< 2
∅ Iterations	1481
Success Rate	67%

(b) Path Planning Performance (A to C)

Table 5.3.: Comparison of Path Planning Performance between RRT* Method 1 for Two Paths

The experiments were conducted using validated maps free from hallucinations. In both trials, the shorter path ($A \rightarrow B$) consistently demonstrates a higher likelihood of being valid. Consequently, increasing the number of waypoints — and thus reducing the distance between each waypoint — tends to improve overall results.

It can be observed that *Method 1* performs consistently better in both cases. By contrast, *Method 2* exhibits poor performance, especially over longer distances; for example, no valid path was found from A to C. A potential cause is the reduced sampling efficiency when

obstacle checks are disabled, as this increases the likelihood of extending invalid nodes and building an invalid tree. This adjustment appears ineffective for our sampling methods. While smarter node exploration might mitigate this issue, it falls outside the scope of this work.

Path Following

Path following was evaluated for a single path from A to B. The complete process of transmitting motion data to the Loop-X and achieving the target position required approximately 170 seconds for a path consisting of 5 waypoints.

5.3. Performance Evaluation

Achieving high performance on the HoloLens 2 is essential for ensuring a positive user experience, as the device's target frame rate is 60 fps. Maintaining a frame rate near this target is critical for responsive user interfaces. During typical application execution, the frame rate averages approximately 50 fps when the UI menu is displayed, which is due to the high amount of interactables in the menu. Additionally, each additional waypoint added to the scene typically reduces the frame rate by approximately 2-5 fps. Furthermore, active tracking introduces an additional frame rate decrease of approximately 5 fps.

It is important to note that this performance impact is highly variable and dependent on the HoloLens' operational state—factors such as low battery levels and elevated device temperatures can significantly degrade performance.

Overall, the application's performance is primarily constrained by CPU resources, particularly during path-planning tasks. For instance, path-planning calculations complete in under 2 seconds, compared to 10 seconds, for 2500 iterations on a laptop CPU (Intel i7-10750H), notably when using the Unity MONO backend without IL2CPP conversion. This result underscores that the computational demands of this aspect of the application are predominantly CPU-bound.

5.4. Tracking

The tracking performance was evaluated in two distinct environments: first, within a clustered environment (Figure 5.4) where the table was positioned in the portal of the Loop-X, and second, in a clean environment with a substantial distance from nearby obstacles. In both scenarios, the tracking began with a side viewing angle of 45° (see Figure 5.4). During the experiment, the observer walked around the Loop-X, ensuring it remained within their field of view. The experiment was conducted at least five times for each environment. The metrics evaluated included the elapsed time since tracking started, the number of timeouts ($t > 60$ s) when tracking was unsuccessful, and the average positional deviation.

The data collected from both environments are summarized in Table 5.4, which outlines the number of timeouts and average distance offsets for each condition.

Figure 5.4.: Tracking in Clustered Environment

Measurement	Clustered Environment	Clean Environment
Timeouts	1	0
∅ **Deviation (cm)**	8 ± 9 cm	4 ± 3 cm
∅ **Time (s)**	18 ± 12 s	5 ± 5 s

Table 5.4.: Tracking Performance in Different Environments

Tracking performance in clean environments is generally effective. However, in densely clustered environments, Vuforia occasionally encounters difficulty identifying the object. Clearing the tracking cache often mitigates this issue, though overall performance tends to be lower in such settings. It might be possible to improve this performance by providing different visual markers to the Loop-X.

5.5. Limitations

5.5.1. Long-Range Path-Planning

The current system is limited in long-range path-planning. Presently, only local path-planning has been implemented, which introduces inefficiencies for long-distance navigation. Incorporating a global path-planning algorithm that utilizes a pre-mapped floor layout of the entire building could significantly reduce computation time and improve overall performance. Additionally, the current setup requires the user to manually travel to the target location to set the destination, as Loop-X's Wi-Fi system only supports short-range communication. This process necessitates the user to return to the starting position, which diminishes usability. A potential solution could be introducing a "follow" feature, allowing the operator to maintain control of the robot and intervene if any issues arise, thereby enhancing operational efficiency.

5.5.2. Movement Execution

The current system operates by iteratively transmitting a pose offset at each waypoint along the generated path. Typically, in response to the environment, an intermediate pose is created every 1-2 meters during the path-planning process. When this positional data is communicated to the Loop-X, each movement incurs a preparation period for the wheelbase, thereby introducing a latency. Implementing manual control over each wheel and following a continuously smoothed trajectory could substantially reduce the required movement time. Achieving this improvement would require modifications to the UDP interface, which are presently not implemented.

Drift

A further challenge is the cumulative positional error during movement. With each pose adjustment, a minor error is introduced, which can be as big as a 5 cm deviation for 5 meters traversed. This issue could be mitigated through continuous tracking; however, in the current program iteration, this approach is limited by performance constraints.

5.5.3. Additional Sensors

Currently, only the spatial mesh data from the HoloLens 2 is utilized, which restricts the sensing range to a maximum of 3 meters, as the depth camera's capabilities do not extend beyond this distance [1]. This limitation necessitates thorough scanning of the environment beforehand to ensure accurate navigation. Integrating additional sensors could expand the range and improve system accuracy, particularly in larger or more complex environments.

5.5.4. Multi-Floor Navigation

The current implementation faces challenges when navigating multiple floors or inclined surfaces. The occupancy grid generated is suitable for two-dimensional path-planning only, which restricts the robot's ability to navigate in environments with vertical elements. Although a three-dimensional voxel representation could address this issue, it would demand significantly higher computational resources, particularly for collision detection. Furthermore, the system is also currently unable to support elevator navigation.

5.5.5. Complex Environments

Another limitation concerns the robot's ability to navigate in constrained or cluttered spaces, where factors such as the tilt of the Loop-X and the positioning of its display screen become critical. For instance, if an obstacle, such as a table, is located within the robot's pathway, it may impede movement. Therefore, currently a manual waypoint system allows the robot to move along predefined routes without performing real-time path planning. This option can be used, for example, to navigate around a table that would otherwise be detected as an

obstacle. While the 3D voxel representation enables collision detection, path-planning in a 3D environment has not yet been implemented due to concerns over computational efficiency.

5.5.6. Tracking Limitations

The tracking system, which utilizes Vuforia, also presents limitations. In complex environments with multiple nearby obstacles or objects, such as a table near the Loop-X, the system may experience delays in detecting the robot's position and may also suffer from reduced accuracy. In contrast, tracking performance improves in simpler environments where the Loop-X has an unobstructed view, and the camera can capture a full view of the robot. Although an alternative method exists, wherein a model is placed at the position of the Loop-X instead of relying on tracking, this approach is less user-friendly.

6. Conclusion and Future Work

6.1. Conclusion

The integration of automated path-planning with MR for the Loop-X robot demonstrates the feasibility of navigation workflows within clinical environments. Utilizing the RRT* algorithm alongside the spatial awareness capabilities of HoloLens 2, this research enables automated navigation that addresses complex environmental factors in hospitals. The findings support that viable solutions are achievable with the current advancements in augmented reality technology, offering the potential to significantly improve operational efficiency and reduce cognitive burdens on healthcare personnel.

The evaluation demonstrates that RRT* for path-planning produces promising results, supporting the development of a fast and reliable local path planner. The current sampling strategies are effective in generating feasible paths in most cases. Before investigating alternative path-planning algorithms, it may be worth exploring other sampling strategies first. Ultimately, a combination of diverse path-planning algorithms may provide the most effective solution.

Nevertheless, certain limitations were observed, particularly in handling dynamic, moving obstacles and addressing potential latencies during real-time processing, which could impact navigation accuracy in high-demand situations. These insights pave the way for future enhancements that could address these challenges and further extend the utility of MR-guided navigation systems in healthcare and other complex environments. This work, therefore, opens opportunities for continued research on advanced path-planning algorithms and broader MR applications.

6.2. Future Work

Future research should focus on augmenting environmental scanning capabilities by integrating additional sensors, allowing the system to detect more complex objects and obstacles beyond the current 3-meter range of the HoloLens 2 depth camera. This would also help to alleviate some of the problems with hallucinations in the hololens sensors.

Optimizing the path-planning algorithm to reduce computation time and improve adaptability in dynamic environments could further enhance system performance. This may involve refining the current implementation or exploring alternative algorithms better suited to the unique demands of clinical environments. Investigating effective methods for managing noisy map data and tracking moving objects would also be a valuable direction.

As tracking capabilities are currently unavailable with holographic remoting — despite

its enhancement of application responsiveness — developing this feature could substantially enhance user experience.

Furthermore, expanding the HoloLens tracking and path planning to support multi-floor navigation, like elevators or ramps, would also enhance its applicability in hospital settings.

Finally, conducting more extensive field tests in real-world clinical settings is essential for evaluating the system's robustness and practicality. Such testing would provide valuable feedback from users, guiding iterative improvements to refine the application and ensure its effectiveness in routine hospital operations.

A. General Addenda

A.1. Code

The code for the implementation can be found on the gitlab repository **https://gitlab.lrz.de/i16-narvis-theses/2024_christiansen_pathfindingrobotop** or at this LRZ share link **https://syncandshare.lrz.de/getlink/fiMm1g6rnm5Hi4ynJgMPYx/GitRepository**.

A.2. Experimental Results

This section presents comprehensive data for all experimental results, along with a description of the data generation methodology.

A.2.1. Virtual Testing

Virtual Testing was conducted on the Unity scene "TestScene.unity".

Success	Iterations
true	1635
true	2500
true	1510
false	-
true	2049
true	2500
true	1800
true	2240
false	-
true	1700

(a) Path A → D

Success	Iterations
true	2500
true	2500
false	-
true	1168
true	2500
false	-
true	1312
false	-
true	1338
true	2285

(b) Path B → D

Success	Iterations
true	2021
false	-
false	-
false	-
false	-
true	1096
true	2500
true	2468
true	1237
false	-

(c) Path C → D

Success	Iterations
true	1448
true	1287
true	1643
true	1416
true	1457
true	1313
true	1396
true	1289
true	1208
true	1667

(d) Path A → B

Table A.1.: Detailed Results for Each Virtual Experiment

A.2.2. Field Study

All data were collected using a valid map, with the HoloLens battery maintained at 90% or higher. Remote timing measurements were conducted exclusively for RRT* Method 1. Success rates were evaluated solely on the HoloLens device, not in the remote setting; remote timing reflects measurements taken with a comparable iteration count. Note that remote timing was recorded using Mono; faster results could be anticipated with the use of IL2CPP.

RRT Method	Success	Time (s)	Remote Time	Num Iterations
1	True	5	<1	412
1	True	6	<1	1340
1	False	3	<1	400
1	False	8	<2	2500
1	True	8	<2	2500
1	True	3	<1	308
1	True	4	<1	1287
1	True	4	<1	1506
1	True	7	<2	2102
1	True	4	<1	631
1	False	-	<1	-
1	True	6	<1	425
2	False	8	-	2000
2	False	7	-	2000
2	True	8	-	2000
2	False	8	-	2000
2	False	7	-	2000
2	False	8	-	2000
2	False	8	-	2000
2	True	8	-	2000
2	False	7	-	2000
2	False	8	-	2000
2	False	8	-	2000

(a) Detailed Trial Data (A to B)

RRT Method	Success	Time (s)	Remote Time	Num Iterations
1	False	-	-	-
1	False	-	-	-
1	True	3	<1	400
1	True	8	<2	2500
1	True	6	<2	1700
1	True	5	<1	700
1	False	-	-	-
1	True	9	<2	2500
1	True	5	<2	1253
1	False	-	-	-
1	True	6	<1	1577
1	True	6	<2	1221
2	False	6	-	2000
2	False	8	-	2000
2	False	7	-	2000
2	False	7	-	2000
2	False	8	-	2000
2	False	8	-	2000
2	False	8	-	2000
2	False	9	-	2000
2	False	8	-	2000
2	False	8	-	2000
2	False	9	-	2000
2	False	8	-	2000

(b) Detailed Trial Data (A to C)

Table A.2.: Detailed Path Planning Performance Data for Trials (A to B and A to C)

A.2.3. Tracking Results

Tracking time was measured after resetting the Vuforia cache. It is important to note that all tracking was initiated from a 45° angle, as effectiveness is significantly reduced when starting from a front or side view.

Environment Type	Deviation (m)	Time (s)	Timeout
clustered	0.06	20	false
clustered	0.03	30	false
clustered	0.17	7	false
clustered	0.05	12	false
clustered	0.07	25	false
clustered	-	-	true
clear	0.03	3	false
clear	0.02	10	false
clear	0.04	4	false
clear	0.05	3	false
clear	0.04	5	false
clear	0.02	5	false
clear	0.06	5	false

Table A.3.: Tracking Performance Metrics for Different Environment Types

A.3. Videos of Application

Two videos are provided to illustrate the general usage of the application. The first **video** demonstrates the path generation process within the application, along with visualization aids such as the generated map and voxel representation.

The second **video** demonstrates the Loop-X path-following process for a generated path.

Both videos were recorded using the remote application setup, as video recording on the HoloLens significantly impacts performance.

List of Figures

List of Tables

List of Algorithms

Glossary

$\mathbb{R}^2$ Two-dimensional Euclidean space, i.e., the set of all points (x, y) where $x, y \in \mathbb{R}$. 9, 53

RRT* Method 1 Refers to RRT* variant where only poses without any obstacle obstruction are used for path planning. 40, 41, 48, 52

RRT* Method 2 Refers to RRT* variant where obstructed poses are used for path planning. Additional cost is added to these nodes to ensure minimal obstacle encounters. 38, 40, 41, 52

S^1 Set of points on the unit circle, i.e., the set of angles $[0, 2\pi)$. 9, 53

Acronyms

AABB Axis Aligned Bounding Box. 14

APF Artificial Potential Field. 7

AR Augmented Reality. 3, 12

EDF Euclidean Distance Field. 21

HCI Human Computer Interaction. 11

MR Mixed Reality. v, 1, 5, 11, 13, 35, 46

MRTK Mixed Reality Toolkit 3. 5, 13, 31, 36

OBB Oriented Bounding Box. 14, 15, 17, 18

PRM Probabilistic Roadmap. 7

RRT Rapidly Exploring Random Tree. 7, 14, 22, 24–27, 38, 46

SAT Separating Axis Theorem. 15–17, 19

UI User Interface. 31

Bibliography

[1] Microsoft. *HoloLens 2 Technical Specifications*. Accessed: 2024-09-25. 2024. URL: https://www.microsoft.com/en-us/hololens/hardware#document-experiences.

[2] Microsoft. *HoloLens 2 Hardware Overview*. Accessed: 2024-09-24. 2024. URL: https://learn.microsoft.com/de-de/hololens/hololens2-hardware.

[3] Brainlab. *Loop-X*. Accessed: 2024-09-24. 2024. URL: https://www.brainlab.com/surgery-products/overview-platform-products/robotic-intraoperative-mobile-cbct/.

[4] Brainlab. *Loop-X*. Accessed: 2024-09-24. 2024. URL: https://www.brainlab.com/de/chirurgie-produkte/uebersicht-ueber-plattformprodukte/robotische-intraoperative-cbct-bildgebung/.

[5] L. Chen, T. W. Day, W. Tang, and N. W. John. "Recent Developments and Future Challenges in Medical Mixed Reality". In: *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2017, pp. 123–135. DOI: https://doi.org/10.1109/ISMAR.2017.29.

[6] S. Park, S. Bokijonov, and Y. Choi. "Review of Microsoft HoloLens Applications over the Past Five Years". In: *Applied Sciences* 11 (Aug. 2021), p. 7259. DOI: https://doi.org/10.3390/app11167259.

[7] R. Van Krevelen and R. Poelman. "A Survey of Augmented Reality Technologies, Applications and Limitations". In: *International Journal of Virtual Reality (ISSN 1081-1451)* 9 (June 2010), p. 1. DOI: https://doi.org/10.20870/IJVR.2010.9.2.2767.

[8] W. Zhang, Y. Li, P. Li, and Z. Feng. "A BIM and AR-based Indoor Navigation System for Pedestrians on Smartphones". In: *KSCE Journal of Civil Engineering* (2024), p. 100005. ISSN: 1226-7988. URL: https://www.sciencedirect.com/science/article/pii/S1226798824051511.

[9] A. Yamashita, K. Sato, S. Sato, and K. Matsubayashi. "Pedestrian Navigation System for Visually Impaired People Using HoloLens and RFID". In: *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. 2017, pp. 130–135. DOI: https://doi.org/10.1109/TAAI.2017.9.

[10] A. Angelopoulos, A. Hale, H. Shaik, A. Paruchuri, K. Liu, R. Tuggle, and D. Szafir. "Drone Brush: Mixed Reality Drone Path Planning". In: *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2022, pp. 678–682. DOI: https://doi.org/10.1109/HRI53351.2022.9889504.

[11] C. Hutton, N. Sohre, B. Davis, S. Guy, and E. S. Rosenberg. "An Augmented Reality Motion Planning Interface for Robotics". In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2019, pp. 1313–1314. DOI: https://doi.org/10.1109/VR.2019.8798010.

[12] A. M. Yasar, R. Voûte, and E. Verbree. "Direct Use of Indoor Point Clouds for Path Planning and Navigation Exploration in Emergency Situations". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLVIII-4/W11-2024 (2024), pp. 175–181. DOI: https://doi.org/10.5194/isprs-archives-XLVIII-4-W11-2024-175-2024. URL: https://isprs-archives.copernicus.org/articles/XLVIII-4-W11-2024/175/2024/.

[13] P. Yu, A. Arora, and Y. S. G. S. Soh. "A Mixed Reality Platform for Online Motion Planning of Mobile Robots via Hololens 2". In: *The 9th International Conference of Asian Society for Precision Engineering and Nanotechnology (ASPEN 2022)*. 2022. DOI: https://dx.doi.org/10.3850/978-981-18-6021-8_OR-18-0303.html.

[14] Microsoft. *HoloLens 2 Research Mode*. Accessed: 2024-09-25. 2024. URL: https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/research-mode.

[15] M. Mononen. *Recast Library Unity*. Accessed: 2024-09-25. 2024. URL: https://recastnav.com/.

[16] P. Hübner, M. Weinmann, and S. Wursthorn. "Voxel-Based Indoor Reconstruction From HoloLens Triangle Meshes". In: *CoRR* abs/2002.07689 (2020). DOI: https://doi.org/10.5194/isprs-annals-V-4-2020-79-2020. eprint: 2002.07689. URL: https://arxiv.org/abs/2002.07689.

[17] L. Han, F. Gao, B. Zhou, and S. Shen. "FIESTA: Fast Incremental Euclidean Distance Fields for Online Motion Planning of Aerial Robots". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 4423–4430. DOI: https://doi.org/10.1109/IROS40897.2019.8968199.

[18] P. I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Second. ISBN 978-3-319-54413-7. Springer, 2017. DOI: https://doi.org/10.1007/978-3-319-54413-7.

[19] F. Gul, I. Mir, L. Abualigah, P. Sumari, and A. Forestiero. "A Consolidated Review of Path Planning and Optimization Techniques: Technical Perspectives and Future Directions". In: *Electronics* 10.18 (2021), p. 2250. DOI: https://doi.org/10.3390/electronics10182250.

[20] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. "Informed RRT*: Optimal Sampling-Based Path Planning Focused Via Direct Sampling of an Admissible Ellipsoidal Heuristic". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2997–3004. DOI: https://doi.org/10.1109/IROS.2014.6942976.

[21] B. Luders, M. Kothari, and J. How. "Chance Constrained RRT for Probabilistic Robustness to Environmental Uncertainty". In: *AIAA Guidance, Navigation, and Control Conference* (Aug. 2010). DOI: https://doi.org/10.2514/6.2010-8160.

[22] H. Suwoyo, A. Adriansyah, J. Andika, A. U. Shamsudin, and M. F. Zakaria. "An Integrated RRT*SMART-A* Algorithm for Solving the Global Path Planning Problem in a Static Environment". In: *IIUM Engineering Journal* 24.1 (2023), pp. 269–284. DOI: https://doi.org/10.31436/iiumej.v24i1.2529.

[23] S. Al-Ansarry and S. Al-Darraji. "Hybrid RRT-A*: An Improved Path Planning Method for an Autonomous Mobile Robots". In: *Iraqi Journal for Electrical and Electronic Engineering* 17 (June 2021), pp. 1–9. DOI: https://doi.org/10.37917/ijeee.17.1.13.

[24] Z. Wu, Z. Meng, W. Zhao, and Z. Wu. "Fast-RRT: A RRT-Based Optimal Path Finding Method". In: *Applied Sciences* 11 (Dec. 2021), p. 11777. DOI: https://doi.org/10.3390/app112411777.

[25] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad. "RRT*-SMART: A Rapid Convergence Implementation of RRT*". In: *International Journal of Advanced Robotic Systems* 10.7 (2013), p. 299. DOI: https://doi.org/10.5772/56718.

[26] D. Devaurs, T. Siméon, and J. Cortés. "Enhancing the Transition-Based RRT to Deal with Complex Cost Spaces". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 4120–4125. DOI: https://doi.org/10.1109/ICRA.2013.6631158.

[27] J. Jermyn. "A Comparison of the Effectiveness of the RRT, PRM, and Novel Hybrid RRT-PRM Path Planners". In: *International Journal for Research in Applied Science and Engineering Technology* 9 (Dec. 2021), pp. 600–611. DOI: https://doi.org/10.22214/ijraset.2021.39297.

[28] D. Kim, J. Lee, and S.-e. Yoon. "Cloud RRT*: Sampling Cloud Based RRT*". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 2519–2526. DOI: https://doi.org/10.1109/ICRA.2014.6907211.

[29] P. Vadakkepat, K. C. Tan, and W. Ming-Liang. "Evolutionary Artificial Potential Fields and Their Application in Real Time Robot Path Planning". In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*. Vol. 1. 2000, 256–263 vol.1. DOI: https://doi.org/10.1109/CEC.2000.870304.

[30] L. Tang, S. Dian, G. Gu, K. Zhou, S. Wang, and X. Feng. "A Novel Potential Field Method for Obstacle Avoidance and Path Planning of Mobile Robot". In: *2010 3rd International Conference on Computer Science and Information Technology*. Vol. 9. 2010, pp. 633–637. DOI: https://doi.org/10.1109/ICCSIT.2010.5565069.

[31] Microsoft. *HoloLens 2 Spatial Mapping*. Accessed: 2024-09-25. 2024. URL: https://learn.microsoft.com/en-us/windows/mixed-reality/design/spatial-mapping.

[32] A. Duenser, Rapha, H. Seichter, and M. Billinghurst. "Applying HCI Principles to AR Systems Design". In: Jan. 2007. URL: https://api.semanticscholar.org/CorpusID:38316922.

[33] Unity Technologies. *Unity Real-Time Development Platform*. https://unity.com/de/products/unity-engine. Version 2022.3.28f. 2024. URL: https://unity.com/de/products/unity-engine.

[34] Unity Technologies. *Unity IL2CPP*. Accessed: 2024-11-2. 2024. URL: https://docs.unity3d.com/6000.0/Documentation/Manual/scripting-backends-il2cpp.html.

[35] Unity Technologies. *Unity Burst*. Accessed: 2024-09-28. 2024. URL: https://docs.unity3d.com/Packages/com.unity.burst@1.8/manual/index.html.

[36] Unity Technologies. *Unity XR Template Documentation*. Version 2.0, Mixed Reality Package. 2024. URL: https://docs.unity3d.com/Packages/com.unity.template.mixed-reality@2.0/manual/index.html.

[37] Microsoft. *MRTK 3*. Accessed: 2024-09-24. 2024. URL: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/.

[38] Unity Technologies. *Unity AR Foundation Documentation*. Accessed: 2024-11-04. 2023. URL: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@6.0/manual/index.html.

[39] PTC Inc. *Vuforia Engine*. Accessed: 2024-09-24. 2024. URL: https://developer.vuforia.com/home.

[40] J. Bialkowski, S. Karaman, and E. Frazzoli. "Massively Parallelizing the RRT and the RRT*". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 3513–3518. DOI: https://doi.org/10.1109/IROS.2011.6095053.

[41] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire. *Real-Time Rendering 4th Edition*. Boca Raton, FL, USA: A K Peters/CRC Press, 2018, p. 1200. ISBN: 978-1-13862-700-0. DOI: https://doi.org/10.1201/b22086.

[42] R. A. Finkel and J. L. Bentley. "Quad Trees: A Data Structure for Retrieval on Composite Keys". In: *Acta Informatica* 4.1 (Mar. 1974), pp. 1–9. ISSN: 1432-0525. DOI: 10.1007/BF00288933. URL: https://doi.org/10.1007/BF00288933.

[43] H. Samet. "An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures". In: *Theoretical Foundations of Computer Graphics and CAD*. Ed. by R. A. Earnshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 51–68. ISBN: 978-3-642-83539-1. URL: https://api.semanticscholar.org/CorpusID:873842.

[44] M. C. Lin, D. Manocha, J. Cohen, and S. Gottschalk. "Collision Detection: Algorithms and Applications". In: *Algorithms for robotic motion and manipulation* (1997), pp. 129–142.

[45] M. Schwarz and H.-P. Seidel. "Fast Parallel Surface and Solid Voxelization on GPUs". In: *ACM Trans. Graph.* 29.6 (Dec. 2010). ISSN: 0730-0301. DOI: https://doi.org/10.1145/1882261.1866201.

[46] Y. Tian, W. Huang, Y. Wang, X. Yi, Z. Wang, and X. Yang. "Multi-level Occupancy Grids for Efficient Representation of 3D Indoor Environments". In: *PRICAI 2016: Trends in Artificial Intelligence*. Ed. by R. Booth and M.-L. Zhang. Cham: Springer International Publishing, 2016, pp. 517–528. ISBN: 978-3-319-42911-3. DOI: https://doi.org/10.1007/978-3-319-42911-3_43.

[47] P. F. Felzenszwalb and D. P. Huttenlocher. "Distance Transforms of Sampled Functions". In: *Theory of computing* 8.1 (2012), pp. 415–428. DOI: https://doi.org/10.4086/toc.2012.v008a019.

[48] S. Karaman and E. Frazzoli. "Sampling-based Algorithms for Optimal Motion Planning". In: (2011). DOI: https://doi.org/10.1177/0278364911406761. arXiv: 1105.1186 [cs.RO].

[49] X. Jin, Z. Yan, H. Yang, Q. Wang, and G. Yin. "A Goal-Biased RRT Path Planning Approach for Autonomous Ground Vehicle". In: *2020 4th CAA International Conference on Vehicular Control and Intelligence (CVCI)*. 2020, pp. 743–746. DOI: https://doi.org/10.1109/CVCI51460.2020.9338597.

[50] K. Naderi, J. Rajamäki, and P. Hämäläinen. "RT-RRT*: a Real-Time Path Planning Algorithm Based on RRT*". In: Nov. 2015, pp. 113–118. DOI: https://doi.org/10.1145/2822013.2822036.

[51] K. Hauser. "Lazy Collision Checking in Asymptotically-Optimal Motion Planning". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 2951–2957. DOI: https://doi.org/10.1109/ICRA.2015.7139603.

[52] Microsoft. *UX Components*. Accessed: 2024-09-24. 2024. URL: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-uxcomponents/packages/uxcomponents/overview.

[53] A. Dirin and T. H. Laine. "User Experience in Mobile Augmented Reality: Emotions, Challenges, Opportunities and Best Practices". In: *Computers* 7.2 (2018). ISSN: 2073-431X. DOI: https://doi.org/10.3390/computers7020033. URL: https://www.mdpi.com/2073-431X/7/2/33.

[54] Microsoft. *Object Manipulator in MRTK3 Spatial Manipulation*. Accessed: 2024-11-08. 2023. URL: https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-spatialmanipulation/packages/spatialmanipulation/object-manipulator.

[55] A. Ejaz, S. A. Ali, M. Y. Ejaz, and F. A. Siddiqui. "Graphic User Interface Design Principles for Designing Augmented Reality Applications". In: *International Journal of Advanced Computer Science and Applications* (2019). URL: https://api.semanticscholar.org/CorpusID:86421919.

[56] S. Lindemann and S. LaValle. "Incrementally Reducing Dispersion by Increasing Voronoi Bias in RRTs". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*. Vol. 4. 2004, 3251–3257 Vol.4. DOI: https://doi.org/10.1109/ROBOT.2004.1308755.