Where Am I? Indoor Localization and Navigation

Matěj Mrázek Árni Bjarnsteinsson Ross Roessler Fabian Blatter

Abstract

In the localization problem, we attempt to find the position and rotation of the user with respect to some reference frame. Classical techniques like GPS are well suited to outdoor localization, where approximate position and missing height is enough to provide high quality localization and navigation capabilities. However, the intricacies of indoor environments require more accurate estimates.

In this work, we explore a method to localize the user inside a known indoor environment using camera data, using off the shelf localization algorithms and models. We further design an AR app for the Magic Leap 2 headset that automatically localizes the user, allows him to preview the current position, and supports in-world navigation to any other place in the environment.

1. Introduction

In this work, we design an AR application that is capable of localizing the user in a known environment based on the AR headset camera data, then previews the estimated position and allows navigating to other places in the world.

Our application follows a client-server architecture, where the client is the AR headset responsible for showing the world overlay and navigation, while the server runs the localization algorithm, capable of returning the estimated world-space pose given a single camera image.

We structure this report as follows - Section 2 shows the entire architecture of the app, section 3 describes how we use the Hierarchical Localization algorithm to perform localization on the server side, section 4 describes the features of the Unity application running on the headset, and finally, section 5 describes the results of the user study performed to analyze the typical user experience.

2. Architecture

The architecture of the application is illustrated in Figure 1. The application first queries the headset for the camera data, which is then encoded as a Base64 string. The result, along with the local pose of the headset, is then converted into a JSON representation and sent to the server running the localization algorithm (Section 3). Depending on whether we are running in the automatic or manual localization modes (as described in section 4.2), the server returns either the pose in world space or the required reposition for the headset.

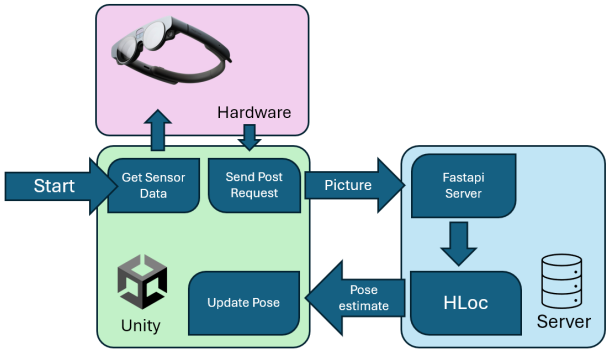

Figure 1. Control flow of the application's algorithm

Since the quality of the prediction is highly correlated with the sharpness of the image, we schedule the server queries at times when the headset is moving relatively slowly. We only send the next request after the current one has finished processing, or after a timeout duration of 10 seconds.

3. Localization

For indoor localization, we use the Hierarchical Localization algorithm [9], subsequently referred to as HLOC, an existing algorithm well suited for both indoor and outdoor environments. Using this algorithm was recommended as part of the initial project description, and we did not explore other methods for two reasons: the main focus of our project was to create a successful implementation, and HLOC supported several keypoint finding and matching algorithms that we could test and choose from.

To localize the user, HLOC requires an existing database of images, their poses in world space, and their camera intrinsics. For this, we utilize the LaMAR [11] dataset, an extensive collection of multi-sensor data streams captured by AR devices and laser scanners over several sessions, filling all the requirements imposed by HLoc. To our knowledge, this was the best available dataset of image and localiza-

tion data within the ETH HG building where our final demo would take place.

In the following sections we describe how LaMAR and HLOC work and our modifications and contributions to create a working indoor localization application.

3.1. LaMAR Dataset

The LaMAR [11] dataset contains multi-sensor data streams captured by HoloLens, iPhone, and NavVis laser scanner devices. Initially, we tried to use the NavVis laser scan data, as we thought it would be the most accurate, but we realized that it didn't have a way to access the 3D point or depth information that was required by HLOC. Ultimately, all we could use was the raw image and depthmap data from the HoloLens and iPhone devices. The HoloLens data was more comprehensive, but in worse quality - the main issues were with the lower resolution and the a slightly different pose of the depth cameras relative to the main image camera, requiring camera transformations, cropping, and interpolation to use. Augmenting with iPhone data, which had image data at 20x the resolution, produced better results.

Another challenge of the data was the irregularity. The HoloLens had only one depth image for every 20 camera images, and even that one was not taken at the exact same time. Another part we required that was often missing in LaMAR was a known global pose for every camera image, which was only available for a small subset of images. Ultimately, we decided to only use the camera image data that had an associated global trajectory, and for each camera image use the temporally closest depth image. For our use case, precise position and orientation information was more important than precise depth information, and we felt that attempting to interpolate position and orientation information was error-prone and hard to implement.

The HoloLens captured data from four different image cameras, but we only used one, the front left, because it had the most spatial overlap with the depth camera. We did not try using the data from other cameras.

Additionally, the LaMAR dataset had a lot of data captured around the exterior of the building, which we were not interested in for the task of indoor localization. We decided to filter these out to reduce processing time and potential false positive matches. To do this, we converted the poses of all images into the global space and filtered out the images outside the manually set building bounds.

We implemented our own preprocessing solution that created the final database after all of the above-mentioned filtering steps, including all of the images and the relevant metadata. The final dataset contains about 10,000 images.

3.2. Hierarchical Localization

Hierarchical Localization is an algorithm that attempts to localize a query image in a given environment, using a database of existing reference images and their poses. It does this in three steps - first, it extracts keypoints in the query, then it tries to find images with similar keypoints in the database, and finally, it attempts to reconstruct the best 3D pose that would map the keypoints in one image to the other. Finally, we find the K best-matching images. Since we know their absolute pose in world space and the relative pose of the query image, we can then combine the transforms to obtain the estimated pose of the query in world space, which is required by our application.

We describe the three main steps and our contributions to each in the sections below.

3.2.1. Keypoint Extraction

First, all the database reference images are preprocessed by extracting two different types of feature points, one for fast image matching and one for more precise matching. We did this offline using the reduced image dataset as described in subsection 3.1 and stored the results on the server. During localization, the same types of feature points are extracted for each query image sent to HLOC.

We use NetVLAD [1] feature points for the fast image matching and SuperPoint [3] features for the more precise image matching. To determine which feature points perform best for our use case, we manually captured a query dataset of images from the hallway where our demo would take place and tested several different feature algorithms. The demo hallway was a challenging environment, as there were not many salient and unique areas, and instead the patterns of the doorways and floors were almost identical to several other areas in the ETH HG building that our reference dataset included. Empirically, we found that some newer algorithms such as LightGlue [8] produced more false positive matches. NetVLAD and SuperPoint worked best for our dataset, the feature extraction speed was not a consideration as we could preprocess the reference database of images.

3.2.2. Matching

In the matching step, we begin by using NetVLAD [1] for fast image matching, which returns the top N most similar images in the reference database to the query image. Then, for each of these images, we use SuperGlue [10] to match the previously extracted SuperPoint [3] feature descriptors.

In our tests, we found that NetVLAD returned very accurate matches for at least the top 3 images. However, we found that these images sometimes did not work for the next step of 3D pose estimation. We describe how we handled this in section 3.2.3. We observed that we can obtain better final results when using 40 images from NetVLAD, while

still keeping acceptable performance levels at about 7 seconds per query, as opposed to 4.5 seconds with only the top 5 NetVLAD images.

3.2.3. 3D Pose Estimation

Getting an accurate 3D pose estimate was the most difficult part of our HLOC implementation.

For each of the reference images returned from NetVLAD, we load, rescale and interpolate the 3D points from corresponding depth images before transforming them to global coordinates. This requires some manual configuration to account for depth sensor differences between HoloLens and iPhone data. Notably, the HoloLens depth sensor is not in the same place as the cameras, requiring camera transformations as well as cropping and shifting.

We send the points from the top K images with the highest number of feature points with valid depth points to COLMAP [12] [13], which uses RANSAC to estimate the camera position and orientation. Although RANSAC should be robust to outliers, we found that sending 5 or more images to COLMAP generally gave a worse prediction than fewer images, as can be seen in table 1. This seemed to be because some images would have one or several points that would match incorrectly and induce an error in the final estimates of up to 20 meters. Changing the RANSAC error threshold did not mitigate this issue.

We also found that the top 3 images returned by NetVLAD, despite being correct matches and visually extremely similar to the query image, were not always the best images to send to COLMAP. We hypothesize that this is because of inaccurate or missing depth data, though it could also be from poor SuperPoint feature matches, and we did not have time to do a full analysis. We instead chose to query NetVLAD for more than 3 images. As seen in table 1, our test set analysis showed that processing the top 5 NetVLAD matches but only sending the top 3 with the highest number of SuperPoint feature matches with valid depth points to COLMAP produced the best 3D pose estimation results. However, empirically, sending more images seemed to work even better, and for our final application we actually queried NetVLAD for 40 matches. A deeper discussion of this is in section 3.3.

Finally, we take the result of COLMAP, and find the rigid transform between the local camera origin and the global prediction. Then we post-process this transformation by first rejecting it if the total number of inliers in the top 2 images is less than 40 and then doing simple clustering of all accepted local to global translations found so far and update the transform to the current transform only if it is in the largest cluster. As future work, we would need to do some proper smoothing on these clusters and better outlier detection so the prediction doesn't jump around as much.

Netvlad N	Pose estim. k	Time (s) ↓	Distance ↓
40	1	7.50	8.47
40	3	7.63	6.85
40	40	7.96	7.16
5	1	3.71	**4.29**
5	3	3.73	4.42
5	5	3.75	4.59
3	1	3.43	4.51
3	3	3.48	4.81
1	1	**3.17**	10.06

Table 1. Localization Experiment Results. Last column shows the mean Euclidean distance to the ground truth.

3.3. Localization Results

For the validation of our localization method we held out one entire data collection session from the image retrieval database. We predicted the location and computed the average prediction time and prediction accuracy, i.e., the mean Euclidean distance from the ground truth, of 50 images from that session. The results are in table 1. These results indicate that the best method would be to retrieve 5 images using NetVLAD and then use the top 3 images in the COLMAP pose estimation. However, we empirically found that retrieving the top 40 images worked better in real-world scenarios with data from the actual AR device. We hypothesize that this might be because our real world query images are more dissimilar to the retrieval image dataset than the test set is, due to camera differences or the fact that the retrieval image dataset was taken a few years in the past, and that therefore retrieving more images from NetVLAD increased the chance of some having more keypoint matches.

4. Unity Application

To preview the current location of the user and enable navigation capabilities, we use the Unity Game Engine [5] to develop an AR application for the Magic Leap 2 [7] headset.

This section is dedicated to describing all the features available in our application, particularly, we first describe how the world is rendered in subsection 4.1, second, we discuss the two options for aligning the world in subsection 4.2, and third, we discuss the 2D map and navigation capabilities in subsection 4.3.

4.1. World Rendering

To render the world, we elegantly combine several data sources from the LaMAR [11] dataset. We show a subset of the captured lidar point clouds as well as trajectories taken by the researchers capturing our dataset to preview where localization data is available. We also process the data to obtain a 3D model of the entire environment, this is then

used to perform correct depth testing, to prevent us from rendering trajectories or points occluded by walls or objects in the real world.

4.1.1. Point clouds

We take all the point clouds from the lidar sessions in dataset, convert the positions of the points into global world coordinates and subsample them (taking each point with uniform probability) to obtain a set of 800 thousand points, including their normals and colors.

We render the points as circles in the game engine, and we implement our own shader for animating them that pulses the points in a wave originating from the position of the user, giving him a sense of the distance to all points in the scene. To ensure points won't be rendered inside walls and culled during depth testing, we move them along their normals. The color of the points is set to match the real world. A preview of the final look is shown in Figure 4.

To improve performance, we split the points into several disjoint axis-aligned bounding boxes based on their position, ignore the ones too far away from the camera ($> 30m$), and fade them in as the player moves closer. Preview is shown in Figure 7.

4.1.2. Trajectory rendering

To preview where localization data is available, we render the paths taken by the researchers collecting our dataset. To minimize visual clutter we only show the sessions for collecting Lidar data. We take the trajectory points from the dataset and apply a cubic bezier curve interpolation to obtain a smoothed version. We render the resulting line as a mesh in Unity, with a custom shader on top that occasionally shows footsteps moving forward on each trajectory and pulses alongside the point clouds. A preview is shown in Figure 5.

4.1.3. The environment overlay

To guarantee that we will not be able to see points through walls in our AR overlay, we render 3D mesh of our environment into the depth map before the point clouds, allowing us to hide the points and trajectories occluded by building geometry.

Since we failed to load and process the mesh provided by LaMAR due to the high memory requirements, we instead construct our own version. We use the point clouds to generate a voxel grid representation of the entire scene, where each cell stores the number of points inside it. We then use Blender [6] to convert the density voxel grid representation into a lower resolution 3D mesh, and use the available retopology tools to simplify it further. We show the obtained mesh in Figure 8.

4.2. Aligning The World

Aligning the world refers to the process of figuring out the relative transformation needed to map from the headset space into the absolute environment space. This transform can be then used to set the position and rotation of the headset inside our application to match the overlay to the real world.

Our application supports two alignment modes. First, a manual one where we only refine an initial estimate, allowing us to preview the world overlay even without relying on the localization algorithm. Second, an automatic one, where we query the server using the current camera data to obtain an estimate for the current camera transformation.

4.2.1. Manual Alignment

When a button is held on the controller, we ignore the movement of the user, ensuring that the view of the world is kept static - contrary to the normal AR experience where the user is able to move throughout the world by changing the position of the headset.

This allows for an intuitive way to change the relative position of the headset and world origins, letting the user to move the overlay to align it with the current world better. However, a limitation is that for each change in alignment, the user has to travel the equivalent distance in the real world, making this method unsuitable for large changes in position.

4.2.2. Automatic Alignment

To perform automatic alignment, we take an image using the main camera of the headset and send it to the server, which responds with the estimated pose of the image. By then taking the estimate and the original headset pose at the time of taking the image, we can solve for the relative transform of the headset to world coordinates and apply it to move the camera.

Since this method is quite prone to noise, particularly when exposed to many similarly-looking locations throughout the dataset, we only show the estimates on the map, described in the next section. The stability and shortcomings of this system are evaluated in more detail in section 3.

4.3. The Location Map and Navigation

4.3.1. The Minimap

To allow the user to preview his position, view automatic localization estimates, and allow navigation capabilities, we implement a 2D map as shown in Figure 6. The map appears when a button on the controller is pressed, and by pointing to a different point in the environment and confirming, a path can be planned to the targeted location. The trajectory is then shown both on the map and in world space, and can be followed.

4.3.2. Navigation

We allow the user to navigate from his position to any position selected on the minimap. To create the navigation graph, we use the vertices and edges from the LaMAR trajectories, then add edges between each vertex and the closest vertex of every possible other trajectory, if the distance is below a given threshold. We empirically find that this gives us a reasonably good and strongly connected navigation graph while preventing us from introducing quadratically many edges, hindering performance.

To plan the path, we first find the closest point to the user and to the target in the graph. Then we use a standard implementation of the A* algorithm [4] to find the shortest path between the two.

4.4. Windows Demo

To allow anybody to explore our visualization, without the alignment part, we also release a 3D demo available on Windows. A preview is shown in Figure 9, and a build is available for download on Google Drive.

5. User Study

To get a good idea of the typical user experience, allowing us to further improve our application, we conducted a user study. In particular, we wanted to find out how intuitive the interaction and visualization of our maps and trajectories is to the user, as well as whether the usefulness and the required localization accuracy would be deemed to be acceptable.

We asked 9 students who attended the demo day of the Mixed Reality course and tested our application for feedback. We used the system usability scale [2] [15] to assess the usability of our system (Figure 2). After evaluating all responses, we obtained a mean of 74 points, giving our application a rating of usable, but with room for improvement.

We also asked the participants to rate how much they agreed with these statements on a five point Likert scale [14] and compiled the results. Our questions were as follows:

- I understood what the "Where Am I" app was supposed to do.
- I was able to get aligned in the world.
- It was easy to get aligned in the world.
- I was able to understand where I was in the building by using the minimap.
- My position on the minimap often "jumped around".
- I was able to successfully navigate to a target.
- I enjoyed the navigation experience.

90% percent of users understood what the app was supposed to do and enjoyed the navigation experience, and 80% were able to successfully navigate to a target. We found this quite promising, as we thought this was a great way to showcase why somebody would actually want to use the app.

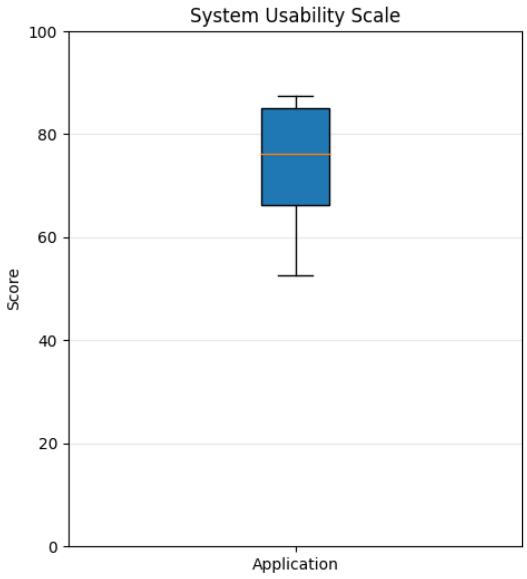

Figure 2. System Usability Scale

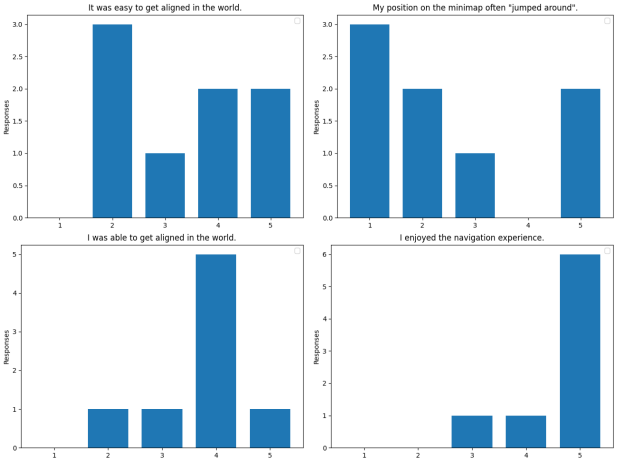

Figure 3. Responses from the user study

The main negative responses were regarding the localization, see Figure 3, which in some cases could be highly inaccurate. This is partly because we haven't been able to devise a good enough heuristic for whether an image is useful or not before predicting, so sometimes it would take long before getting an accurate guess.

This would be a good area to conduct future work, as selecting better images would prevent us from spending valuable time on predicting the location for low quality samples. In particular, looking at the tilt or movement of the headset or the number of salient features might give us better estimates. We would also like to look further into robustly combining several subsequent predictions to obtain a better and smoothed estimate of the current position less prone to errors induced by outliers.

6. Figures

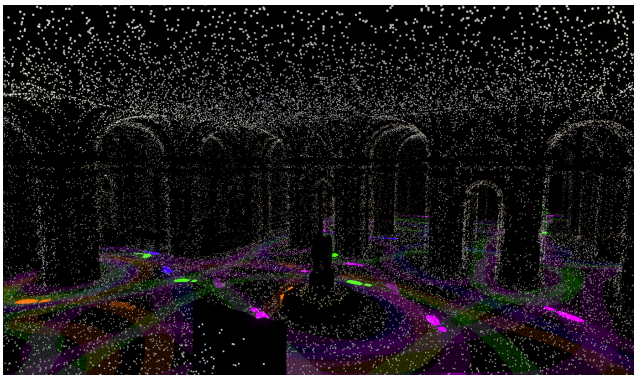

Figure 4. Point Rendering Preview. We show the surroundings of the fountain right after entering ETH.

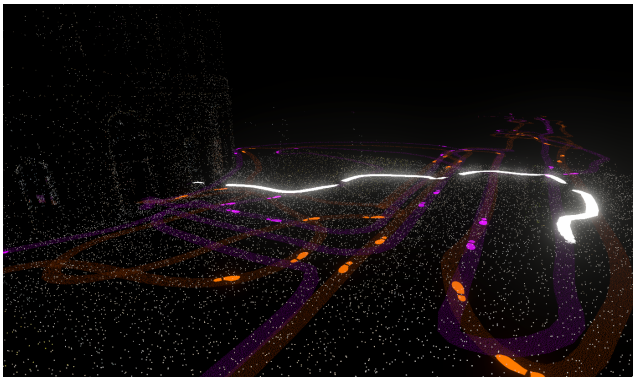

Figure 5. Trajectory Rendering Preview. We show the trajectories in the front of the HG building. Pink and orange trajectories depict the paths taken when collecting several runs of the LaMAR dataset, white trajectory is the current planned navigation path.

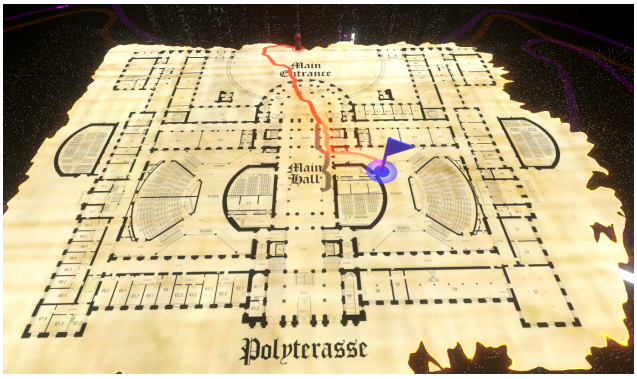

Figure 6. The Minimap. The red pawn is the current location of the user, the blue flag is the localization target. Red trajectory is the currently planned path, shown in white in 5, the trajectory currently pointed to by the user and awaiting confirmation is shown in brown.

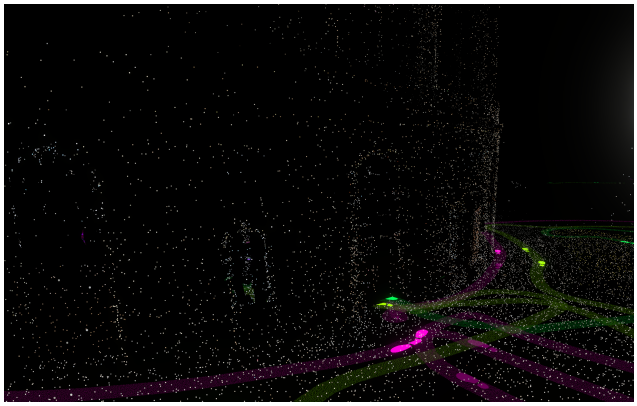

Figure 7. The Point Clipping Preview. We show a similar scene to 5 - however, as we have moved closer to the entrance, points on the building are now fully present and not only fading in.

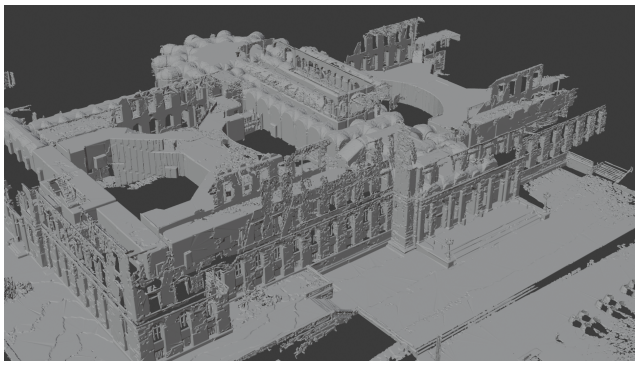

Figure 8. The Building Model, the spot in the bottom right is the Polyterasse. Some parts in the front are missing, however, the most important parts - walls on the inside of the building responsible for most of the occlusions - are present.

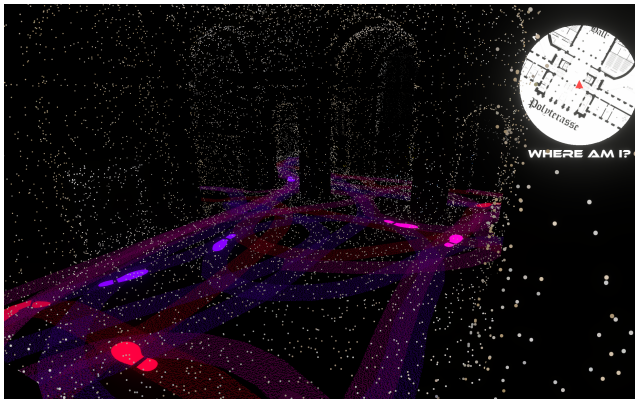

Figure 9. The Windows Demo. We also made an application running on windows only for the demo day. As an additional feature, it has an always-present minimap in the top right (this was excluded in the AR version as it was unpleasant to have it blocking the view).

References

[1] Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition, 2016. 2

[2] John Brooke. *SUS – a quick and dirty usability scale*, pages 189–194. ResearchGate, 1996. 5

[3] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description, 2018. 2

[4] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4 (2):100–107, 1968. 5

[5] https://unity.com. The unity game engine, 2025. Accessed on 6.1.2025. 3

[6] https://www.blender.org. The blender free and open source 3d creation suite, 2025. Accessed on 6.1.2025. 4

[7] https://www.magicleap.com/magic-leap-2. The magic leap 2 headset, 2025. Accessed on 6.1.2025. 3

[8] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. Lightglue: Local feature matching at light speed, 2023. 2

[9] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale, 2019. 1

[10] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks, 2020. 2

[11] Paul-Edouard Sarlin, Mihai Dusmanu, Johannes L. Schönberger, Pablo Speciale, Lukas Gruber, Viktor Larsson, Ondrej Miksik, and Marc Pollefeys. LaMAR: Benchmarking Localization and Mapping for Augmented Reality. In *ECCV*, 2022. 1, 2, 3

[12] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3

[13] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 3

[14] https://en.wikipedia.org/wiki/Likert_scale. Likert scale wikipedia, 2025. Acessed 1.6.2025. 5

[15] https://en.wikipedia.org/wiki/System_usability_scale. Usability study wikipedia, 2025. Accessed on 1.6.2025. 5