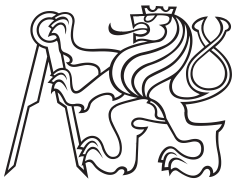
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Visual Localization in Dynamic Environments

Martina Dubeňová

Supervisor: doc. Ing. Tomáš Pajdla, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2022

Acknowledgements

I would like to thank my supervisor Tomáš Pajdla for an opportunity to work on this thesis. He never let me down anytime I needed his help. I also want to thank my family for providing the perfect environment so that I did not need to worry about anything else besides the thesis. My friends also deserve a great amount of appreciation for their eternal support. Last but certainly not least, my greatest gratitude goes to Ing. Michal Polic for his never ending help — for sharing his knowledge with me, being patient and the best moral support. This work could not have been done without him.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 20. May 2022

.....
Martina Dubeňová

Abstract

This thesis aims to predict the 6 degree-of-freedom (6DoF) pose of a query image containing moving objects in a 3D map. First, we create a fully automatic generation of datasets for localization from a Matterport scanner. A method for creation of a dataset with moving objects is presented. The dataset is then used to evaluate a new proposed way of localization in dynamic environments. We demonstrate that our pipeline improves localization compared to not accounting for the dynamic objects.

Keywords: Indoor visual localization, Localization in Dynamic Environments, Creation of localization dataset from Matterport

Supervisor: doc. Ing. Tomáš Pajdla, Ph.D.

Abstrakt

Cieľom tejto práce je odhadnúť pózu v šiestich stupňoch volnosti pre query obrázok s pohybujúcimi sa objektami. Najprv vytvoríme plne automatický skript pre vytváranie datasetov pre lokalizáciu z Matterport skeneru. Predstavíme metódu na vytváranie datasetov s dynamickými objektami. Tento dataset je potom použitý na vyhodnotenie nového navrhnutého spôsobu lokalizácie v dynamickom prostredí. Ukážeme, že naše spracovanie zlepší lokalizáciu v porovnaní s prípadom, kedy sa dynamické objekty neuvažujú.

Kľúčové slová: Vizuálna lokalizácia v interiéri, Lokalizácia v dynamickom prostredí, Vytváranie datasetu pre lokalizáciu z Matterportu

Preklad názvu: Vizuálna lokalizácia v dynamickom prostredí

Contents

Chapter 1

Introduction

Visual localization is a method to estimate the 6 Degree-of-Freedom (DoF) camera pose given a RGB image from the scene, *i.e.* the position and orientation of the camera. It is a very important task in computer vision. The possibility of accurate localization leads to many applications, such as self-driving cars, mobile robots, or augmented reality.

For outdoor localization, there is a possibility to use GPS systems. It can be accurate down to several meters, which is not enough for uses such as self-driving cars. Moreover, it does not provide any information about the orientation of the camera. This can be significantly improved using visual localization. For indoor applications it is not possible to use GPS at all, and it needs to rely on visual localization if not wanting to use any additional equipment such as lidar, accelerometers, gyroscopes, or compasses. Even though the mentioned devices are more used nowadays, but they are still expensive or inaccurate and complicated to use.

Indoor visual localization is a complicated task, since even small changes in the pose of the camera lead to great changes in the scene. In addition, the indoor environment is often repetitive [1] and not only on small scales - there are usually many of the same chairs, tables and doors. It also applies on greater scales - hallways and offices often look very similar. Moreover, there are also many plain walls (or windows) that do not provide any useful information for localization. Another challenge is that the environment is only seldom static and often contains a lot of moving objects.

The main goal of this thesis is to improve an already existing algorithm for visual localization [2] so that it also works in dynamic environments, *i.e.,* environments with changing or moving objects. The motivation for this step is the fact that moving objects often lead to big changes in the scene, which can make the original algorithm work imprecisely. Also, dynamic environments are more common in the real world, making this improvement even more important.

1.1 SPRING project

The thesis is part of the EU Horizon 2020 project SPRING [3]. Its goal is to develop socially assistive robots capable of performing not only basic robotic skills such as navigation, grasping and manipulating objects, but also communicating with people naturally. Robots must be able to move, hear and communicate in complex and unstructured public places. This thesis deals with the visual localization of the robot. The project will be tested at Broca Hospital, which is a gerontology hospital in Paris. This is the reason why the thesis assignment is focused on the home / medical environment. Also, hospitals are very dynamic environments, with a lot of moving equipment, people in the rooms or even visitors all around the hallways. The original—static—localization would not work and we assume that our improvements will bring great advancement in this field.

1.2 Objectives

The objectives of the work are the following:

1. Review the state-of-the-art methods for indoor visual localization, see [2, 4–7].
2. Create a new dataset in a medical environment. The dataset should also contain dynamic objects.
3. Make adjustments to [2] to provide robustness against moving objects.
4. Demonstrate and evaluate the improved method on the new dataset.

1.3 Codes

All the codes used in this thesis are publicly available on Github [8].

The links to other repositories are presented:

- Improved InLoc implementation,
- Implementation in Habitat,
- Creation of cutouts in Matlab,
- Blender scripts.

Please note, that I commited the changes under several usernames — either dubenma or dubenma1. Most notably though, the implementation in Habitat was accidentally committed under the name of Michal Polic, since we shared the same computer in the office. All the changes in the repository were done by me only.

Chapter 2

Previous work

As have been said, visual localization is problem of estimating 6DoF pose of the camera. Two main approaches are used in the literature: (i) pre-build 3D map based localization; (ii) image retrieval based localization.

The 3D maps are usually created from 3D point clouds. The point clouds itself are often created from 2D images taken in some local motion—this is known in the literature as Structure-from-Motion (SfM) [9]. A feature descriptor is assigned to each 3D point. One way to obtain the pose is using a neural network (geometric regression [10], regression forests [11], or Long short-term memory (LSTM) [12]) from the input point cloud (RGBD space) to the pose parameters. A second way is using feature matching and solving a Perspective-n-Point (PnP) problem to obtain the query pose ([13–16]). Using 3D maps can directly give us the 6DoF pose, but the method is not efficient in large-scale datasets, *e.g.*, big rooms or entire houses.

Using image retrieval techniques can cope with large datasets, but on the other hand, it can only provide us approximate location of the query. Still, methods based on this principle are widely used. The location of the query is obtained from the most similar image retrieved from image database ([1,2,4,17–19]). To scale onto large datasets, local descriptors are used. More modern approaches are using dense descriptors using Convolutional Neural Networks (CNN) ([2, 4, 20]). While the other are using local information ([21–23]). These methods can be further improved with feature selection ([23, 24]) or feature weighting ([14, 25]).

2.1 InLoc

InLoc [2] is a state-of-the-art visual localization method. There have been a lot of improvements in the field by then. However, since it is part of the assignment, the thesis focuses on improving this method.

The paper describes the Inloc dataset as follows:

- 356 query RGB photos with reference poses,
- 9972 database RGBD images with reference poses.

The base for database images were 277 panoramic RGBD images that were obtained from scanning two buildings at the Washington University in St. Louis with a Faro 3D scanner. For each panorama, 36 RGBD perspective images also called cutouts were obtained. The RGB images for the query were taken using a smartphone camera on 2 floors of the building. The other floors play the role of a confuser in the dataset.

The InLoc pipeline has the following 4 steps:

1. **Candidate image retrieval:** Given an RGB image as an input query, it first identifies a set of locations in the scene potentially visible in the query via image retrieval.
2. **Feature matching:** For each location, it performs feature matching and re-ranks the locations based on the number of matches passing a 2D geometric verification stage.
3. **Pose estimation:** Camera poses are then estimated for the top-ranked locations only.
4. **Pose verification:** Camera poses are visually verified and the image with least error is retrieved.

In the first step, the feature descriptor vectors are extracted using NetVLAD [4] CNN for both the database and the query images. The top 100 database images most visually similar to the query are identified based on the score

similarity matrix. In the second step InLoc performs mutual matching of the densely extracted CNN features for top 100 retrieved images and performs spatial verification by fitting homographies. The tentative matches are geometrically verified by estimating up to two homographies using RANSAC.

In the Pose estimation step, InLoc estimates a 6DoF camera pose for the top 10 candidates with the largest number of homography inliers. As perspective images in our database have depth values, and hence associated 3D points, the query camera pose can be estimated by finding pixel-to-pixel correspondences between the query and the matching database image followed by P3P-RANSAC. The dense 2D-2D matches between the query image and a retrieved database image define a set of 2D-3D matches when taking the depth map of the database image into account. The pose is then estimated using standard P3P-RANSAC.

For evaluation purposes, query reference poses needed to be estimated. The reference camera poses of the query were computed as in Algorithm 1. First, the most visually similar cutouts for each query image are selected (line 5). The cutouts are then automatically matched to the query images (line 7), and the camera pose is computed using the found correspondences (line 9). The poses are reprojected and visually verified. If a significant misalignment is found, the correspondences are manually annotated, and the camera pose is recomputed (lines 10-13). Finally, the poses are quantitatively and visually inspected (line 14).

Algorithm 1 Calculation of the query reference pose

1: **Inputs:** set of query images $\mathbf{Q}$;
2: **Outputs:** reference poses $\mathbf{P}_{ref}$;
3: $\mathbf{P}_{ref} \leftarrow$ empty set of reference poses;
4: **for each** $q \in \mathbf{Q}$ **do**
5: Select N visually most similar cutout images to q;
6: **for each** $n \in N$ **do**
7: Automatically match the cutout$_n$ to the query;
8: **end for**
9: Compute the camera pose P_q and visually verify the reprojection;
10: **if** Significant misalignment is present **then**
11: Manually annotate correspondences for difficult queries;
12: Repeat from 9;
13: **end if**
14: Quantitatively and visually inspect the poses;
15: $\mathbf{P}_{ref} \leftarrow \mathbf{P}_{ref} + P_q$
16: **end for**
17: **return** $\mathbf{P}_{ref}$;

2.2 NetVLAD

NetVLAD is a CNN based architecture that is used for feature extraction. The NetVLAD layer offers a powerful pooling mechanism with learnable parameters that can be easily plugged into any other CNN architecture. In this thesis, we used the same architecture as in [2] and [7]. The NetVLAD layer was combined with VGG-16 [26] and pre-trained on the Pitts30k dataset [4]. The input to the neural network is an image and the output is a NetVLAD descriptor vector.

2.3 Visual Localization with Hololens

The master theses [7] brings improvements to InLoc implemenatation. It also comes with a new dataset of 2 rooms from Czech Institute of Informatics, Robotics and Cybernetics (CIIRC) building in Prague. Matterport 3D scanner was used to acquire panoramic RGBD images. The RGB panoramas had to be downloaded manually from the site. Unfortunately, their rotations were unknown. To fix this, the orientations were estimated by minimizing the differences between the downloaded panoramic image and the projection of the point cloud from the reference position. The found rotations of panoramic images were assumed as ground-truth. Perspective images were created from the rotated panoramic image. The query images were taken in two ways. The first one was using a smartphone camera. To determine the pose, Vicon (pose estimation system) was used. The second one was using HoloLens camera, which is capable of taking images and estimating their poses internally by a SLAM-based algorithm.

The goal of the thesis was to use a sequence of query images for improving the localization. Two methods were implemented. One of them succeeded to bring improvements to the localization, while the other did not perform very well. This thesis is a continuation of the work and builds on the implementation. However, since there were still known errors in the code for localization of sequences, we worked only with a sequence of length 1.

Chapter 3

Creation of dataset

The general content of the dataset for InLoc is: 3D model of the space, RGBD perspective images for database and RGB perspective images for query and known poses for both sets of images. In general query poses are usually not known. In the InLoc paper [2] the poses were estimated and manually verified. In the master thesis [7] the positions were obtained from Matterport SDK but the rotations also needed to be estimated. Manual work always leads to errors in some extends so we aimed to prepare a dataset for which the poses did not need to be estimated.

3.1 Scanning the scene with Matterport camera

The data for the thesis were acquired at 2 different spaces of the Broca hospital. We will refer to those spaces as Hospital Figure 3.1 and Living Lab Figure 3.2. Hospital consists of places such as reception, waiting room, big hallways with elevators and rooms for treating the patients. Living Lab is a smaller space with office environment.

For the creation of our dataset only the *Matterport 3D scanner* was used. We used *Matterport Pro 3D Camera* [27]. The camera was mounted on a tripod and placed at a spot where we wanted to scan the space. The scanning was triggered using *Matterport Capture* app on an iPad. The camera rotates around its z-axis to create 360° picture. Once it finished, the tripod was moved to a new spot. Generally we aimed to have around 1.5 meters between

the spots covering the whole space. The app shows the resulting point cloud from above. The scans automatically align with each other and therefore show the whole scanned space so far, so it is easy to see which spots still need to be scanned. After the scanning is finished, mirrors and windows are needed to be marked in the app and also parts outside of our desired space need to be cut off.

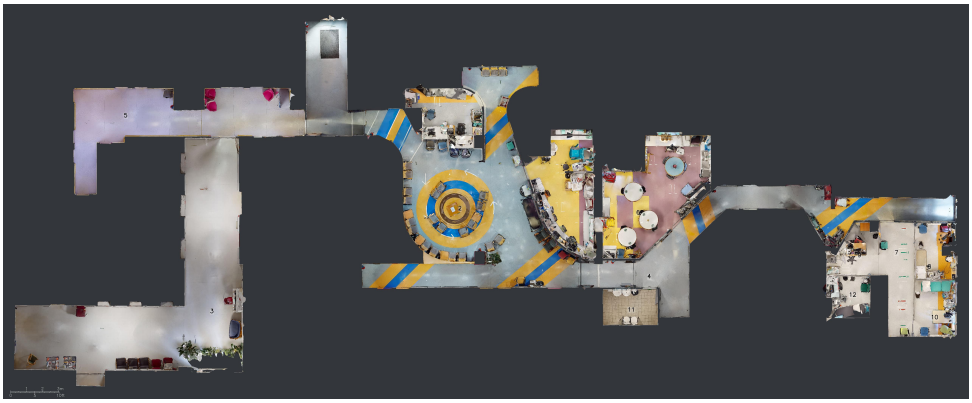


Figure 3.1: The floor plan for the scene Hospital.

Figure 3.2: The floor plan for the scene Living Lab.

The next step is to upload the space to *Matterport cloud* for further processing. According to the size of the space it could take up to several hours to process. Once it was processed the data were available for downloading after purchase in the form of *Matterpak bundle*. It consist of:

- 3D mesh file in .obj format together with corresponding texture map images,
- colorized point cloud in *.xyz* format,
- ceiling and floor plan images in *.pdf*.

The data are available here [8]. For the Hospital space 116 scans and for

Living Lab space 36 scans Figure 3.3 were taken giving us a total of 152 scans.

Figure 3.3: Coordinate systems of the sweeps are shown in the 3D map of the Living Lab scene.

3.2 Acquiring data using Matterport API

The scans are called sweeps in Matterport terminology, so we will refer to them as such. Matterport does not provide direct access to the poses of the sweeps or the panoramic images. These data need to be acquired through Matterport API [28]. It is available after activation of Developer tools. An API key has to be generated to connect to Matterport API endpoint. Then a GET request with the authentication and a query string is sent. The reply is in the form of a *.json* file Listing 3.1.

Listing 3.1: Example of a json listing for one panorama.

```
"data": {
    "model": {
        "locations": [
            {
            "panos": [
                {
                    "id": "pano_id",
                    "position": {
                        "x": 11.188,
                        "y": -2.095,
                        "z": 1.628
                    },
                    "rotation": {
                        "x": -0.015,
                        "y": -0.008,
                        "z": -0.707,
                        "w": 0.707
                    },
                    "skybox": {
                        "children": [
                            "skybox_urls"
                        ]
                    }
                }
            ]
        }
    ]
}}
```

Matterport API does not support retrieving panoramic images as panoramas but as skyboxes only. The skybox consists of 6 square images, each image individually projected onto the shape of a cube: up, down, front, back, left and right. They come bundled with position and rotation data. The data

was saved in a *.csv* file with 9 values for each panorama: id of the panorama, name of the panorama, x, y, z for camera position, w, x, y, z as quaternion for camera orientation.

For the skybox, the json file contained a link for each image to be downloaded. The links expire in 10 minutes. For all of the 152 sweeps, it led to 912 images. The links expire in 10 minutes. We implemented a script to download all the images. Using the skybox images, it is possible to create panorama images using the library cube2sphere [29]. It is a Python script that maps 6 cube faces into an equirectangular map, resulting in a panorama image (Figure 3.4). It requires Python 3 and Blender [30] to be installed. The skyboxes were downloaded with a resolution of 2048x2048 pixels and resulted in a panorama with a resolution of 8192x4096 pixels.

Figure 3.4: Example of a panorama image created from skybox.

All of the above scripts are available on GitHub [31].

3.3 Generating cutouts

Cutouts are RGB perspective images from a scene with a known pose, *i.e*, the position, and the rotation of the camera in the map coordinate system. Having panoramas with their poses, it is possible to generate as many cutouts from the panoramas as needed. The more, the better [19]. However, we were limited to cutouts with positions of the Matterport camera and were able to change their rotations only. We opted for the same sampling of the panoramas as in the original InLoc dataset, *i.e.* 36 images using a sampling of 30° in yaw and ±30° in pitch angles. This gave us 5472 cutouts in total [8].

There are still no data from the robot that will be used in the hospital, and also the parameters of the camera that will be used on the robot are not known. We needed to decide ourselves what the parameters will be, so we are able to generate cutouts. We could have chosen any parameters. It does not influence the code and can easily be changed as soon as there is any further specification.

The focal length f, principal point $[c_x, c_y]$, and resolution $[w_c, h_c]$, which is the width and height of the cutout, of the Hololens camera [32] was used. We opted for these parameters since our lab already worked with Hololens on different projects such as ARtwin [33], where the localization in a factory and construction environment is developed. However, the parameters were later slightly changed due to limitations of the simulator AI Habitat [34] we used as mentioned in subsection 3.4.1.

f	1037.301697
c_x	664.387146
c_y	396.142090
w_c	1344
h_c	756

Table 3.1: Parameters of a calibrated Hololens camera in pixels.

Once the parameters were set, the cutouts could be generated. Panoramas are oriented in the way that when the respective position and orientation from the Matterport API that are given in the map coordinate system are applied, we get to the sweep coordinate system S Figure 3.3 for which it applies that the z_S axis is pointing up and the x_S axis is pointing inside of the image in the middle of panorama Figure 3.5.

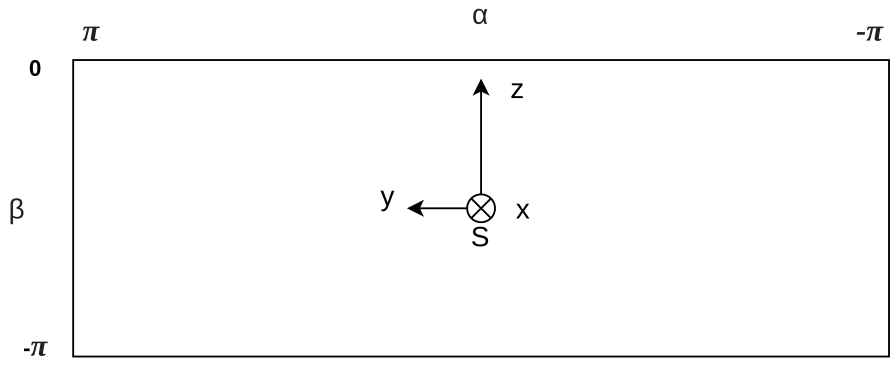


Figure 3.5: Panorama with the sweep coordinate system S and angles α and β. x-axis is pointing to the image, to make the system right-handed.

A cutout with zero additional rotation applied will have the center of the sweep coordinate system in the middle of the image with x_S facing right, y_S facing inside, and z_S facing up Figure 3.6.

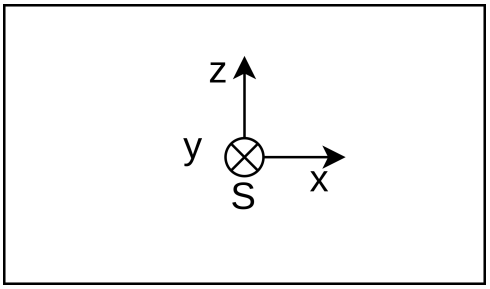


Figure 3.6: Sweep coordinate system S in a cutout with zero **R** rotation applied. The S coordinate system is then identical with the camera coordinate system.

The matrix **R** is used to determine the direction for the camera to generate a cutout. The matrix **R** is given as

$$\mathbf{R} = \mathbf{R}_x^T \mathbf{R}_z^T. \tag{3.1}$$

The matrices $\mathbf{R}_z$ and $\mathbf{R}_x$ are rotation matrices around the axes z and x, respectively. Any form of rotation matrix could have been chosen, even more complicated ones that would include also rotation around y axis. However, we opted for only yaw and pitch since it represents the scenario for looking up, down, and around, since this is the most likely one for the robot.

The goal is to obtain a sphere with the center located at the point $[0, 0, 0]^T$ in the coordinate system S. The panorama is then projected onto the sphere from which we acquire a cutout.

First, the submatrix **iQ** of the projection matrix **P** is computed. It transforms points from image coordinate system to camera coordinate system and is given as:

$$\mathbf{iQ} = \mathbf{R}^T \mathbf{K}^{-1}, \tag{3.2}$$

where **K** is the camera calibration matrix and is given as:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.3}$$

The parameters f_x, f_y, c_x and c_y are the intrinsic parameters of the camera.

f_x and f_y describe horizontal and vertical focal lengths and c_x and c_y the principal point. The origin of the digital image in the image coordinate system is typically located in the upper left corner, so the image must be translated using the vector $[c_x, c_y, 1]^T$.

Then, for each point in the image, its corresponding direction vector $\vec{v}$ in the camera coordinate system is computed by Equation 3.4 and then normalized by Equation 3.5.

$$\vec{v} = \mathbf{iQ} \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} \tag{3.4}$$

$$\vec{n} = \frac{\vec{v}}{||\vec{v}||} \tag{3.5}$$

This way, points on a sphere are obtained onto which we can project the panorama image. Then the angles α and β are calculated from $\vec{n}$:

$$\alpha = -\arctan\frac{n_y}{n_x}, \quad \beta = -\arcsin n_z. \tag{3.6}$$

The pixel positions u and v in the panorama image were computed from these angles given:

$$u = \frac{\beta}{\pi} w_p + w_p + 1, \quad v = \frac{\alpha}{2\pi} h_p + h_p + 1, \tag{3.7}$$

where w_p and h_p are the width and height of the panorama. A value of one needed to be added because it was implemented in Matlab and it indexes from 1.

Bilinear interpolation is then used to acquire the final cutout. The value of the pixel is assigned based on the four closest pixels from the source image.

Each cutout was saved in the *.jpg* format and a *poses.csv* file was saved that consists of: name of a cutout, $[x, y, z]$ position and $[w, x, y, z]$ quaternion

for orientation. The position is the same as the one from panorama while the rotation is computed as:

$$\mathbf{R}_c = \mathbf{R}_p \mathbf{R}^T, \tag{3.8}$$

where $\mathbf{R}_c$ is the cutout rotation, $\mathbf{R}_p$ is the panorama rotation and $\mathbf{R}$ is given in Equation 3.1. It is then transformed from a rotation matrix to a quaternion. An example of generated cutout can be found in Figure 3.7.

(a) : Cutout with rotations $\mathbf{R}_x(0)$ and $\mathbf{R}_z(0)$.

(b) : Cutout with rotations $\mathbf{R}_x(30)$ and $\mathbf{R}_z(-30)$.

Figure 3.7: Example of cutouts generated from the panorama shown in Figure 3.4.

For debugging purposes, the point cloud of the map was projected onto the cutout to determine whether the poses are correct (Figure 3.8)

Figure 3.8: Generated cutout with a projected point cloud of the map.

3.4 AI Habitat

AI Habitat [34] is a simulation platform for training embodied AI agents such as virtual robots. It provides a photorealistic and efficient 3D simulator to test the learned skills before transferring them to reality. Advantages over testing/training in the real world include:

- **faster** - the possibility of parallel processing,
- **no danger** - poorly trained robots do not have any consequences,
- **efficient** - possibility of replicating any conditions easily,
- **availability** - in real life the robot or the right environment may not be available.

The last property was the main reason why AI Habitat was used for this thesis. Since there was no access to the robot that will be used in the SPRING project [3] and the hospital is in France, it is much more practical to make use of a simulator to test the methods for filtering dynamic objects.

AI Habitat [34] consists of two parts: Habitat-Sim and Habitat-API. Habitat-Sim is a flexible, high-performance 3D simulator with configurable agents, sensors, and generic 3D dataset handling. Habitat-API is a modular high-level library for end-to-end development of embodied AI algorithms – defining tasks (e.g. navigation, instruction following, question answering), configuring, training, and benchmarking embodied agents.

We used it to obtain synthetic images from mesh, semantic segmentation images and depth maps for each cutout. Semantic segmentation was first implemented in Blender [30], however, it was very slow, taking several seconds for only one image. Habitat is much faster. It creates a mesh image, semantic image and also a depth map for one cutout in about 0.35 second. Moreover, Blender was not a very good option since it did not support creation of depth maps.

3.4.1 Implementation in Habitat

First we needed to load the mesh 3D model. Habitat supports maps in *.glb* format only. Therefore, the *.obj* mesh map from Matterport needed to be converted to *.glb*. It was done manually in Blender [30].

Later, it was switched to a more complicated model because of future integration with dynamic objects. The map was manually segmented in Meshlab. It resulted in an empty map with mostly only walls, floor, ceilings, and pillars. The segmented objects were saved individually in a folder specifying the type of the object, for example: chair, sofa, table, sink, *etc.*

Since there were now too many objects to convert to *.glb*, a tool *obj2glb.py* was created to do it automatically. It uses the obj2gltf [35] library.

Structure of *habitat_ros_semantic_2/spring_simulation* [36]:

- a script for generating all the accompanying files for cutouts is implemented in *generate_dataset_cutout.py*,
- configuration file in *configs/tasks/pointnav.yaml*,
- empty model in *data/scene_datasets/<space name>_empty.glb*,
- segmented objects in *objects_categorized/<space name>/<type of object>*

The configuration file specifies which sensors are used. For the purpose of the thesis a RGB sensor, a semantic sensor and a depth sensor were used. For each, the parameters width, height, horizontal field of view, position, and orientation of the camera needed to be set. This brings us to the reason why the parameters of our calibrated Hololens camera could not be used Table 3.1. The configuration file does not support decimal values for the field of view. Therefore, the closest value of the horizontal field of view was computed from the focal length f of Hololens and the cutout width w_c:

$$HFOV = 2 \arctan\left(\frac{w_c}{2f}\right). \tag{3.9}$$

There was also no option to change the principal point. It is expected to be in the center of the image. The camera parameters that were used are specified in 3.2. The position and orientation in the configuration file were filled with only zero values. The pose of the camera changes for every cutout and it was more practical to change it in the code.

$HFOV$	66
f	1034
c_x	672
c_y	378
w_c	1344
h_c	756

Table 3.2: Parameters of the camera for cutouts.

The poses of the cutouts are loaded from the *poses.csv* file that was created in Matlab.

Habitat uses a coordinate system that follows OpenCV [37] orientation: x going right, y going down, z going forward. This differs from the one we used in matlab to genereate RGB cutouts which was: x going right, y going forward and z going up. We needed to tranform the poses to habitat coordinate system. For the position it was a simple rotation around the x-axis by -90 degrees. The orientation of the camera in Habitat $\mathbf{R}_h$ was given as:

$$\mathbf{R}_h = \mathbf{R}_x(-90)\mathbf{R}_c\mathbf{R}_x(180), \tag{3.10}$$

The $\mathbf{R}_x$ is a matrix that represents rotation around the axis x (Equation 3.11), $\mathbf{R}_c$ is the rotation of a cutout from Matlab. $\mathbf{R}_c$ is given in the coordinate system used in Matlab and needed to be transformed by -90 degrees around x. The resulting cutout from this was facing the other direction so we also added an additional rotation by 180 degrees to fix this.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 \sin\theta & & \cos\theta \end{bmatrix} \tag{3.11}$$

Habitat has a method visitor_utils.show_observations() that returns observations from all the sensors specified in the configuration file. The input is the position and rotation of the camera. RGB sensor returns a RGB synthesized image from the 3D model.

(a) : Cutout generated from a panorama.

(b) : Synthesized imaged from a RGB sensor in Habitat.

Figure 3.9: Comparison of a cutout generated from Matlab and Habitat.

The semantic image is discussed in the following section regarding dataset with dynamic objects. The depth image represents a depth map: the value of each pixel is the destination in meters to the object shown in the pixel. By default the maximal range of the sensor is 10 meters so we changed the range to infinity to have precise data. We used the depth map to compute the point cloud for each cutout Figure 3.10a.

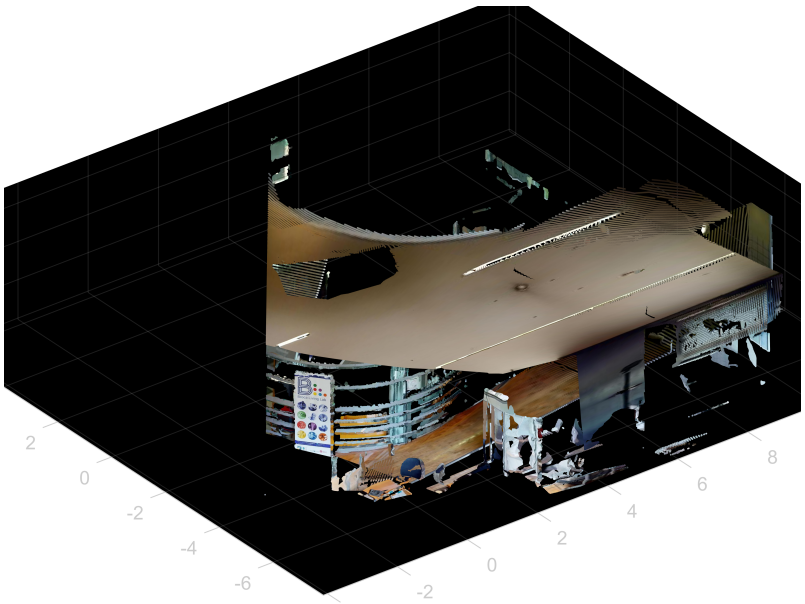
The implementation is accessible on Github [36].

3.5 Dataset with dynamic objects

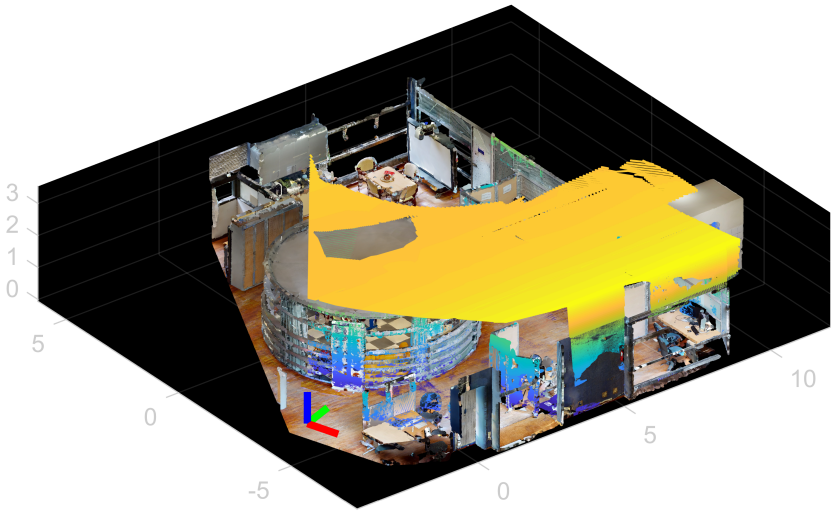
In real life static scenes are not very realistic. There are objects that are moving everywhere. For public places it is especially true for people. They are not a static part of the environment and therefore cannot be used in the localization pipeline. The general idea of the dataset with dynamic objects is that we would still have static database RGBD images, but the queries would contain dynamic objects.

In practice, there could be a neural network responsible for segmentation of objects in a query image such as YOLACT [38]. It would provide us with a mask for dynamic objects. The matches between queries and database image in the area of the mask would not be considered. Which could, in theory, improve the localization.

In this thesis we tried to simulate such dataset. For the dynamic dataset we first manually added objects to an already existing 3D map. We created two new dynamic datasets for each scene using the objects we already had. It was created in Meshlab in the following way:



(a) : Colored point cloud for a cutout created from its depth map.



(b) : The true colored point cloud is a part of the point cloud provided from Matterpak. The yellow-blue point cloud is the point cloud from **(a)**. It shows that the two point clouds align. The axes shown is the camera coordinate system.

Figure 3.10: Poincloud generated from a depth map.

1. The 3D model from Matterpak was loaded.
2. The objects that were manually segmented were also loaded.
3. We moved each object to a new location and saved it.

We decided on this method because it seemed as the best one to simulate

movement of objects. It could have been done in a way that the objects would be also moved away from their original location. However, that would lead to change not only in the new position but also in the old position, since it would result in holes in the final mesh. We needed to create masks for the dynamic objects that would give us information about every change in the scene. That is why the dynamic objects were added to the scene.

We created two sets of datasets with dynamic objects: *dynamic_1* and *dynamic_2*. The dataset without any dynamic objects is referred to as *static* dataset. The dataset *dynamic_1* has the objects placed on the ground in the remaining empty spaces. The dataset *dynamic_2* had twice that many dynamic objects. Half of them are at the locations from *dynamic_1* and half were added to new locations. This time the objects were added also to locations where they would cross other objects. Some of the objects were not placed on the ground but left flying to create greater occlusions.

Unfortunately, we lost the texture maps for the objects so the dynamic objects did not have any color and the default white material was applied. This way it is easily recognizable which objects are the dynamic objects, though. The objects are then converted to *.glb*. After that, they can be loaded to Habitat.

We will need masks for the dynamic objects. The semantic sensor in Habitat was used for the purpose. It gives us the information about what objects are in the cutout. Each object loaded into the scene has its own id associated. The value of the pixel in the semantic image is the id of the object that is in the pixel. This is not very useful on its own. There is a list of ids of the objects for each category of objects. A new semantic image was created that has the ids of the category names as values of the pixels at the place of the objects. A file named *semantic.csv* was created to save the information about object categories. Each line is for one object category and data is following: id of the category, value of the pixels in the semantic image, name of the category. These masks can be used as training data for segmentation in YOLACT [38]. It could also be useful if we wanted to add any of the categories to the dynamic objects.

Finally, the masks for dynamic objects were created from these semantic images. A new image was created. The pixels in the position with the value of id of the dynamic category in the semantic image were given a value of 1 in the new image. In the rest of the positions in the image pixels have the value of 0.

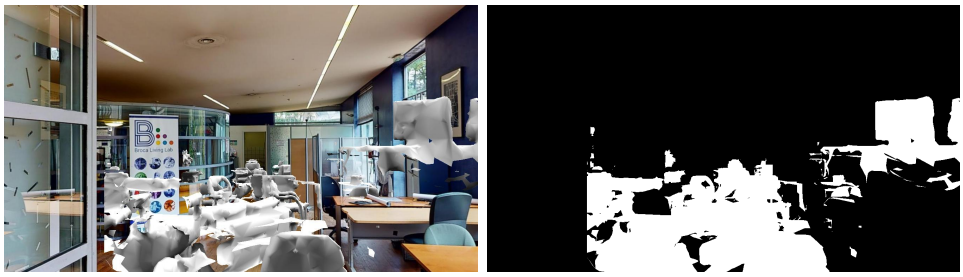
All the necessary data were obtained, and they needed to be changed to

InLoc format.

3.6 Creation of new InLoc dataset

The data obtained will be used for both database and query images. The dataset is split to sweeps that will be used as queries and the rest serves as data for database images. The data structure used in the previous work [7] was preserved. All the data were copied to a server to have them stored at one place. A script was implemented to convert the data to the right structure. All the needed scripts can be found in *localization_service/Build_SPRING/prepareDataset/*.

For each scene, there is a folder for data from Matlab and a folder for data from Habitat. The Matlab folder consists of a single folder named *cutouts* that contained all the created cutouts and the *poses.csv* file created in Matlab. Then there is a folder for data from Habitat. There are 3 subfolders specifying the dataset type: *static*, *dynamic_1* or *dynamic_2*. Each of them contains folders *rgbs*, *depth* and *semantic*. The folder rgbs contains synthetic images from the 3D model. The folder *depth* contains *.mat* files with colored point clouds of each cutout. The folder *semantic* contains semantic images of objects, semantic images specifying the category of objects and a *semantic.csv* file. These are not needed for our dataset. In the dynamic datasets there is also *masks* folder that contains masks for the dynamic objects.



(a) : Dynamic cutout. **(b) :** Mask of dynamic objects.

Figure 3.11: Cutout in a dynamic environment together with its mask of moving objects.

The right structure Figure 3.12 is achieved by running the script *prepareDataset.m*. At the beginning of the script the scene and the type of dataset need to be specified. If a dynamic dataset is chosen, it creates cutout images with dynamic objects Figure 3.11a. It takes a cutout and at the place of a dynamic mask Figure 3.11b it replaces the image with a synthesized image with dynamic objects. The image is saved to a new *cutout* folder in the

habitat directory.

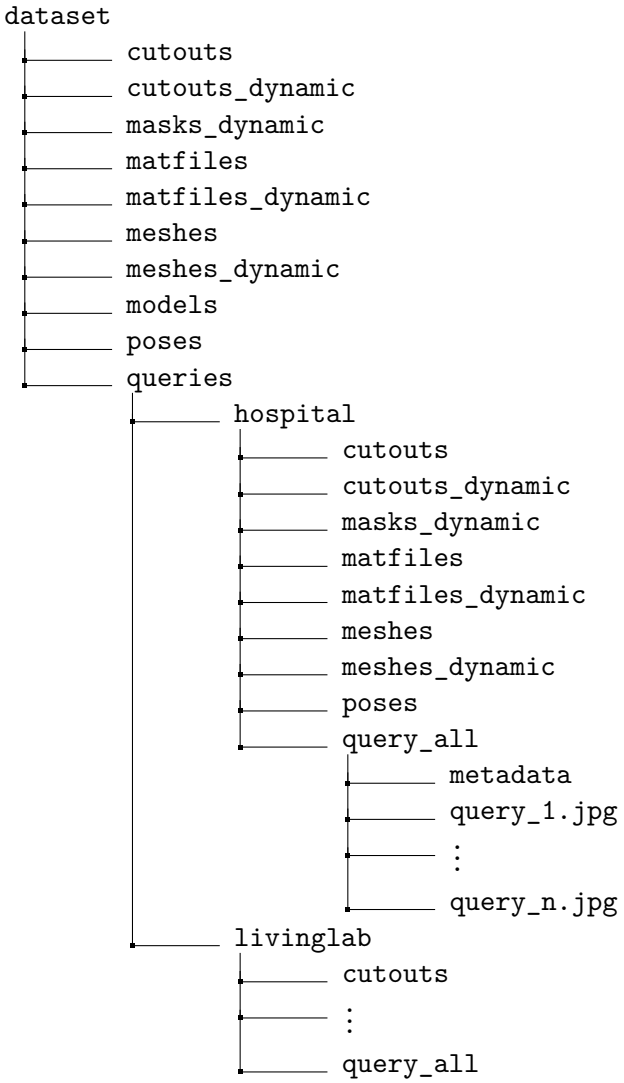


Figure 3.12: Structure of a dynamic dataset.

3.6.1 InLoc dataset without queries

Table 3.3 shows how the data are distributed. The first column specifies from which of the datasets the data were taken. For the dynamic dataset the number of the dataset needs to be added. The second and the third columns specify the source of data and the fourth column the target location in the InLoc dataset. Its full path is *<path to dataset>/<type of dataset>/<space name>/<data type>*. It is further separated. For each of these folders the structure is following: *<space name>/<sweep number>*.

Type of dataset	Data from Matlab	Data from Habitat	Inloc data
static	cutouts/		cutouts/
static		depth/	matfiles/
static		rgbs/	meshes/
dynamic		cutouts/	cutouts_dynamic/
dynamic		depth/	matfile_dynamic/
dynamic		rgbs/	meshes_dynamic/
dynamic		masks/	masks_dynamic/
static	cutouts/poses.csv		poses/

Table 3.3: Comparison of the structure of the obtained data and the new dataset in InLoc format Figure 3.12 for each dataset type. Only the static dataset is true to the original InLoc dataset. Dynamic dataset is our addition.

All the variables are the following:

- Type of dataset: static, dynamic_1, dynamic_2,
- Space name: hospital, livinglab,
- Data type: folder names from the fourth column in Table 3.3,
- Sweep number: 1 to n, where n is the number of sweeps,
  - For hospital space n = 116,
  - For livinglab space n = 36.

The space name is in the path twice. It is for practical reasons. If there was a need to change something in the dataset from one of the scenes it was easier to just delete one whole folder and create the dataset again. In the final dataset with queries the first use of *<space name>* is ignored.

The data in the directory *poses* is created from the file *poses.csv* that contains poses for all of the cutouts. The directory contains *.mat* files with rotation and position for each cutout. For each cutout, the poses are saved in a separate file.

For the static data set, only the directories that are specified as static in the first column of Table 3.3 are used. For dynamic datasets, all of the directories are used. It is because the database data is static and the query data is dynamic.

Finally, we add a folder *models* to which the Matterpak bundle is copied for each scene. We need only the mesh map with textures. Matterport models

have their own unique names, so the *.obj* model needs to be renamed to a general name *model.obj* so that it can be used in the InLoc pipeline without any other changes in the code.

3.6.2 Separating query and database data

The data then need to be separated to database and query. It is implemented in *splitQuery.m*. First, it copies the whole dataset from the previous step. The ids of sweeps we want as queries are specified, and those are then moved to the folder *queries*. We aimed to use around 12% of the sweeps for queries and the rest for database. For the Hospital scene we used 15 out of 116 sweeps Figure 3.13, and for the LivingLab scene 5 out of 36 sweeps Figure 3.14.

For the queries, we actually need only the cutout images and reference poses. However, we keep all of the data in case of any future work. Therefore, the structure of the query data is the same as for the database data, except for two cases. The models are present only in the database, and a folder that contains the query images that will be used for localization is only in the queries data.

Finally, we create a new folder that contains all the query images. These are the same cutouts as in the *cutout* folder. However, these are all together in the folder *query_all* and named as *<number of query>.jpg*. The queries in this folder are the only ones that will be used when running InLoc. We chose not to use all of the cutouts but only the ones looking in the forward direction. We also have cutouts looking up or down but these mostly contain floor or ceiling and are not practical for localization.

The creation of the queries is implemented in *prepare_queries.m*. It copies all the cutouts looking in the forward direction from the *cutout* folder from *queries* for a static dataset or from *cutouts_dynamic* in the case of a dynamic dataset.

Then the script *metadata/query_mapping.m* is created. It contains an array of the true names of the queries from the cutouts folders that are in the *query_all* folder. This is important so that we can retrieve the queries' reference poses and also masks.

For each type of dataset we got:

- 180 queries for Hospital scene,
- 60 queries for Living Lab scene,
- 240 queries in total.

3 new datasets from the Broca hospital for InLoc are complete:

- Broca_dataset_static,
- Broca_dataset_dynamic_1,
- Broca_dataset_dynamic_2.

Figure 3.13: Hospital scene with the locations of the coordinate systems of the sweeps that were used for queries in the Hospital scene. The pink dots are locations of the sweeps that were used for database images.

Figure 3.14: Living Lab scene with the locations of the coordinate systems of the sweeps that were used for queries. The pink dots are locations of the sweeps that were used for database images.

Chapter 4

New implementation of InLoc

The new implementation is based on the code from [7]. The main idea of the new improvement is that in the step of feature matching of InLoc we want to filter out the correspondences between a query image and a database image in the area of a dynamic mask of the query.

There are three different scenarios we want to test:

1. original InLoc on a static dataset,
2. original InLoc on a dynamic dataset,
3. new improved InLoc on a dynamic dataset.

The first scenario is expected to bring the best results since there are no tentative matches lost in the process.

The second scenario has 2D-2D correspondences between a query and a database image in the area of a dynamic mask of the query. Since the cutout provides depth information for every pixel, the 2D-2D correspondences are turned to 2D-3D correspondences. Those are then passed to P3P for pose estimation. The correspondences in the area of the dynamic mask act as confusers and the estimated pose is expected to be worse compared to if there were no matches in the area.

Mode of Inloc	Filtering of dynamic objetcs	Type of dataset
original	off	Broca_dataset_static
static_1	off	Broca_dataset_dynamic_1
dynamic_1	on	Broca_dataset_dynamic_1
static_2	off	Broca_dataset_dynamic_2
dynamic_2	on	Broca_dataset_dynamic_2

Table 4.1: For each of the modes for InLoc there is it's corresponding dataset being used and an implementation with or without filtering of dynamic objects.

In the third scenario, these correspondences are removed and only the tentative matches from the area of the query representing the static environment are used. This is expected to estimate more accurate poses. It is still expected to behave worse than original Inloc on a static dataset since some of the correspondences are lost and in general the more correspondences there, are the more accurate is the pose estimation.

The whole pipeline then consists of these steps:

1. The feature vectors are extracted for all the cutouts and queries using NetVLAD [4].
2. The top 100 visually similar cutouts to the query are retrieved.
3. For each cutout-query pair feature matching is performed. In case of a dynamic mode, matches from the area with dynamic objects are filtered out.
4. The matches are geometrically verified using RANSAC [39].
5. The top 10 cutouts are retrieved for each query ranked based on the number of RANSAC inliers.
6. For each of the cutouts the query pose is estimated using P3P-LO-RANSAC [40].
7. Syntetic images are genereated from the estimated poses.
8. A *score* that measures the similarities of the query image and the synthesized image is computed by DenseSIFT [41].
9. The query is localized by the best pose based on the *score*.

There are 5 modes for the newly implemented Inloc based on which dataset and implementation is used. They are shown in Table 4.1

The main changes were done in the file *parfor_denseGV.m*. GV stands for geometric verification and the steps are the following:

1. Correspondences are first found using the descriptors from the coarse layer conv5 containing high-level information. Additional matches are found using the fine conv3 layer restricted by the conv5 correspondences.
2. Filtering of the dynamic objects.
3. Using RANSAC to find inliers.

4.1 Filtering of dynamic objects

The matches from the step 1. have correspondences on the query image that is resize to the first two dimentions of the output of the fine conv3 layer from NetVLAD. The features from the conv3 layer are the size 94x168x256 and for the conv5 layer 47x84x512. So, the query image needed to be resized to 94x168 in grayscale. A mask for the query is loaded and resized to the same size as the query. It is resized using the nearest-neighbor interpolation method. The output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered. Then we loop through each pixel with a correspondence in the query image. The matches with correspondences in the query image in the areas without dynamic, and the ones with dynamic objects are ignored. We get a new smaller set of matches. Inlier matches are then computed from this set of matches.

4.2 Running the InLoc

The InLoc is run from the script *demo_SPRING_dataset.m*. First, some parameters need to be set to run it correctly. In the file *startup.m* paths need to be set for NetVLAD. Then there is a script *setupParams.m* that sets all the remaining parameters. Most of the paths' parameters contain relative paths, so there is no need to change them. It also runs the script *SPRINGDemoParams.m* that contains most of the setting up of parameters for our purpose.

These parameters need to be set:

- params.dynamicMode: 'original', 'static_1', 'dynamic_1', 'static_2' or 'dynamic_2',
- params.dataset.query.space_names: {'livinglab'}, {'hospital'} or {'livinglab', 'hospital'}

The dynamic mode determines whether the filtering of dynamic objects will be used and which dataset is used as shown in Table 4.1. The space names specify from which scenes the queries will be used. The same scenes are used for the database images.

Chapter 5

Evaluation

Three new datasets were created. One dataset is from a static environment, and two datasets have moving objects included. Our method was tested on the newly created datasets. All of the experiments were tested on the whole datasets with the Hospital scene and the Living Lab scene. There were 5 experiments:

1. original InLoc on the Broca_dataset_static,
2. original InLoc on the Broca_dataset_dynamic_1,
3. new improved InLoc on the Broca_dataset_dynamic_1,
4. original InLoc on the Broca_dataset_dynamic_2,
5. new improved InLoc on the Broca_dataset_dynamic_2.

First, we evaluated the newly created dynamic datasets. The average area of a dynamic masks in the image cover 12.20% of the whole query image for Broca_dataset_dynamic_1. For Broca_dataset_dynamic_2 it is 15.46%. The second dataset contains twice as many dynamic objects as the first one, but as we can see, it did not make much difference. Figure 5.1 and Figure 5.2 show the distribution of the mask area in the query images. There is still a quite high proportion of query images that do not contain any dynamic objects. Around 35% of the queries have between 0 and 10% of the area covered with dynamic objects. For a higher percentage of the area, it is decreasing.

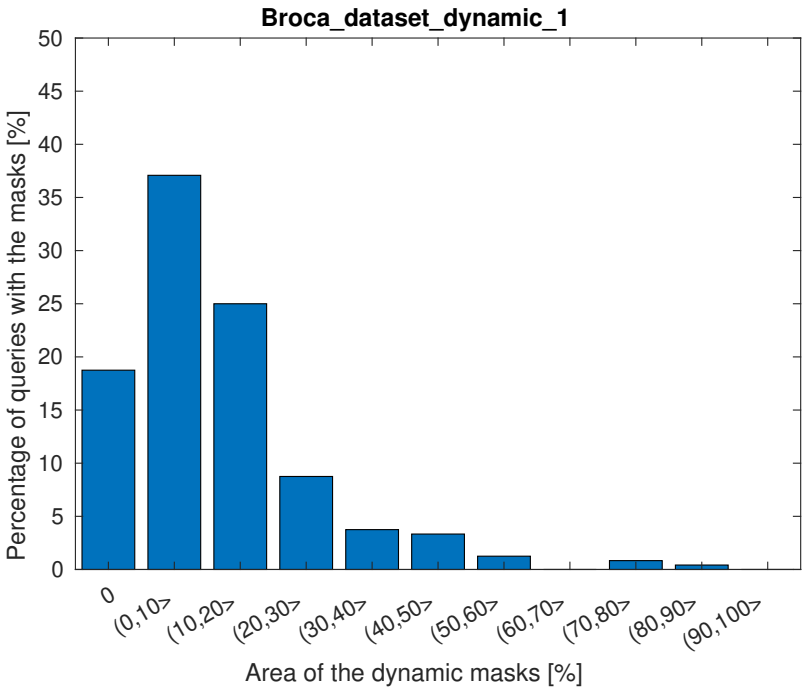


Figure 5.1: The graph shows distribution of dynamic masks in the query images for the dataset Broca_dataset_dynamic_1.

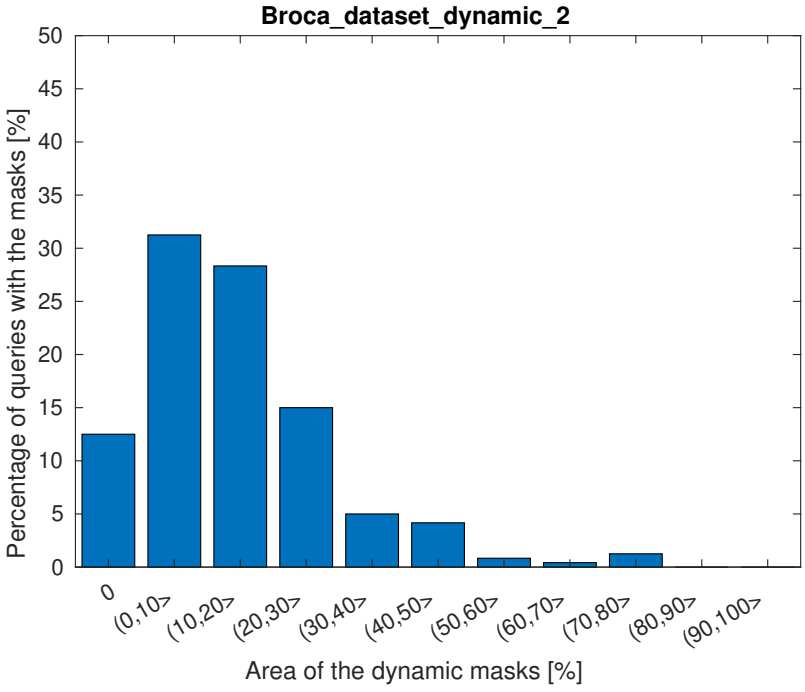


Figure 5.2: The graph shows distribution of dynamic masks in the query images for the dataset Broca_dataset_dynamic_2.

Then we run all five experiments. The results can be seen in Figure 5.3 and Figure 5.4. The green lines show the performance of InLoc on the dataset Broca_dataset_static that does not contain any moving objects. The red lines show the performance of InLoc without filtering the objects and the blue lines with the filtering. Both use a dynamic dataset.

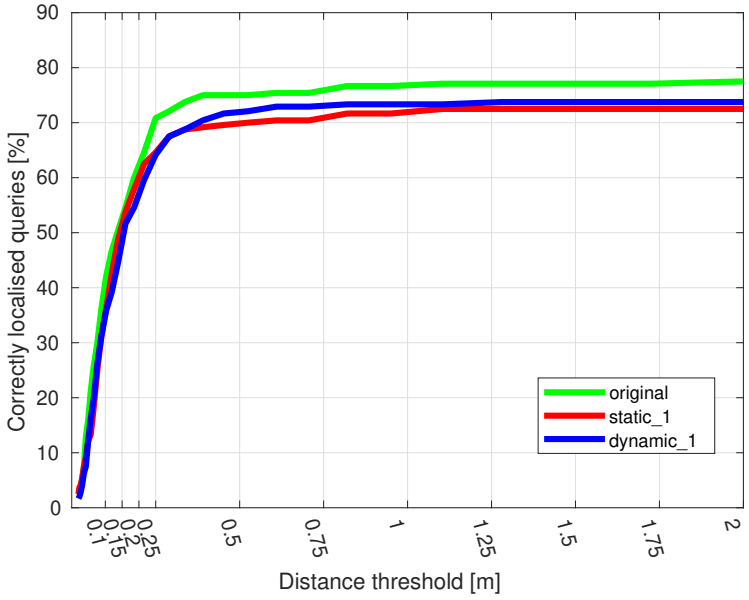


Figure 5.3: Evaluation of InLoc on the Broca_dataset_static and Broca_dataset_dynamic_1

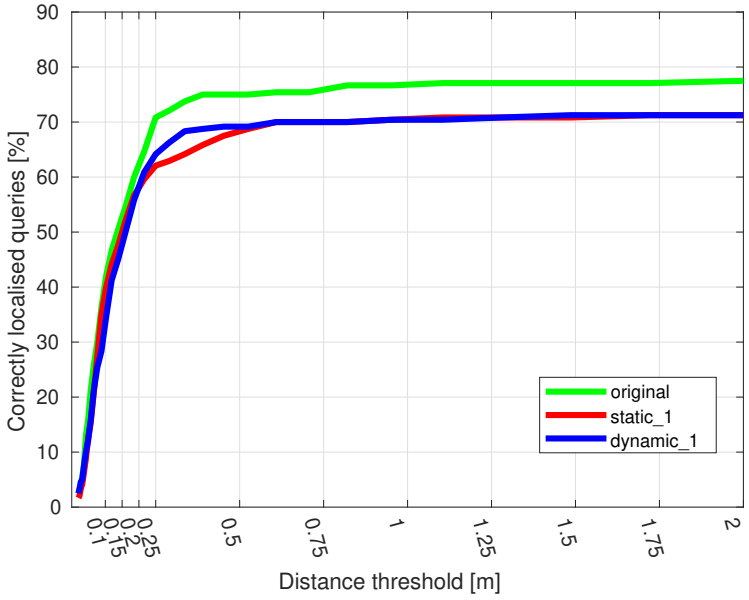


Figure 5.4: Evaluation of InLoc on the Broca_dataset_static and Broca_dataset_dynamic_2

Figure 5.3 and Figure 5.4 show how many of the queries had translation errors within the distance thresholds. As expected, a static environment leads to better localization. In the dynamic environment, the number of inliers is significantly reduced, which makes the pose estimation more complicated.

Figure 5.3 shows an improvement with the filtering of objects. In Figure 5.4 there is not much difference. When we compare Figure 5.5 with filtering and Figure 5.6 without filtering, we can see that there were not many matches found in the area of the dynamic masks. That is why we expect the two methods performed almost the same. The dynamic objects used were white and did not have any texture. Such objects were not present in the static dataset, so there were almost no correspondences that would act as confusers.

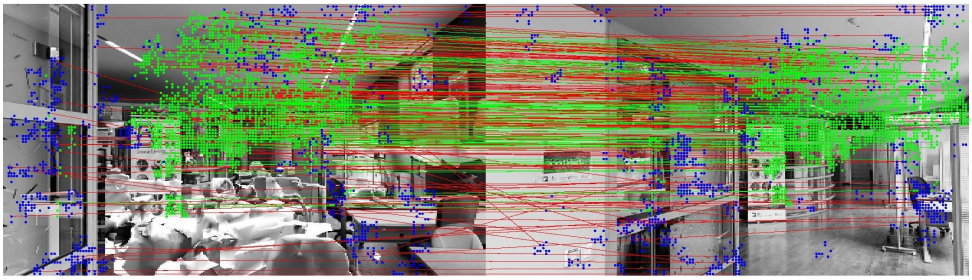


Figure 5.5: Matches between a query and a cutout with the matches in the area of the dynamic objects filtered out. It shows the first two images from Figure 5.8 with their correspondences.

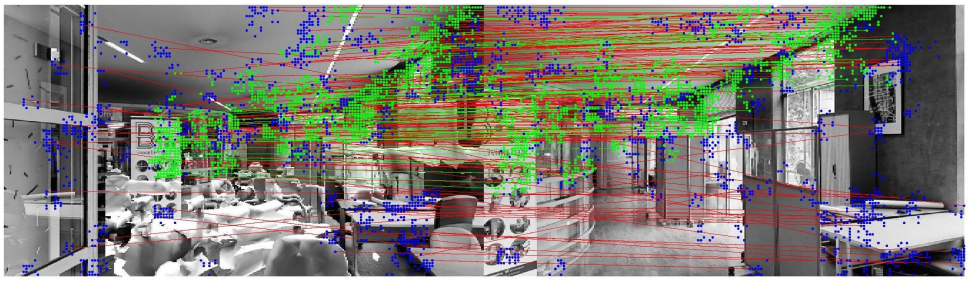


Figure 5.6: Matches between a query and a cutout without filtering. It shows the first two images from Figure 5.9 with their correspondences.

Figure 5.7 shows the filtering process. The left image is a query image. The right image is the retrieved database image from which we estimated the query pose.The blue dots are the features with correspondences shown with red lines. The green dots and lines are their subset and represent the inliers and their correspondences.

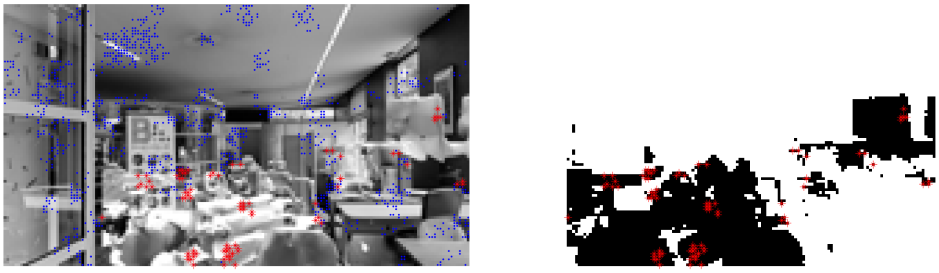


Figure 5.7: The left picture shows a resized query with the query correspondences of the matches between a query and a database image. The blue correspondences are the ones used for pose estimation. The red correspondences are filtered out because they are on dynamic objects. The right picture shows a mask for the query and also the correspondences that were filtered out.

Figure 5.8 and Figure 5.9 show the results of the localization for the same query. The first image shows a query image. The second image is a synthesized image from the retrieved pose. The third image is their comparison. The last image is the best cutout from the dataset.

Figure 5.8: Result of localization with filtering of matches. The error in translation is 0.34 meters and in rotation 1°.

Figure 5.9: Result of localization without filtering of matches. The error in translation is 0.24 meters and in rotation 2°.

Chapter 6

Conclusion

In this thesis, we not only created three new datasets, but also provided a completely automatic implementation for creating new localization datasets from the Matterpak bundle provided by Matterport and the data from the Matterport API [28]. There is also the possibility of having dynamic objects in the dataset, in which case the only manual step is the annotation of objects. In the future, this step could also be automated with the use of neural networks such as YOLACT [38].

The datasets contain RGBD database images and RGBD query images with known poses. In the thesis, the depth of the queries was ignored and they were localized based on RGB information from the image.

We proposed a new pipeline with filtering of dynamic objects included. It showed improvements in localization. We tested it on a dataset where the dynamic objects did not have any texture. It needs to be tested on a new dataset, where the dynamic objects resemble the ones already present in the scene. In that case, the pipeline with filtering should perform the same, while the pipeline without filtering is expected to perform worse.

We already started working on a new method for comparing query images with synthesized images in the last step of InLoc (*pose verification*). This could lead to even further improvements to the pipeline from this work. The results of both works are expected to be published at a conference.

The thesis is submitted together with all the necessary codes that are also

publicly available on Github [8]. The size of the generated datasets is too large to be submitted. Only the base data are uploaded. It includes the Matterpak bundle, panoramic images and their poses. These data should be sufficient to replicate the datasets using the Github codes.

Bibliography

[1] A. Torii, J. Sivic, T. Pajdla, and M. Okutomi, "Visual place recognition with repetitive structures," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 883–890.

[2] H. Taira, M. Okutomi, T. Sattler, M. Cimpoi, M. Pollefeys, J. Sivic, T. Pajdla, and A. Torii, "InLoc: Indoor Visual Localization with Dense Matching and View Synthesis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 4, pp. 1293–1307, 2019.

[3] "SPRING: Socially Pertinent Robots in Gerontological Healthcare," https://spring-h2020.eu/, accessed: 2022-5-14.

[4] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5297–5307.

[5] H. Taira, I. Rocco, J. Sedlar, M. Okutomi, J. Sivic, T. Pajdla, T. Sattler, and A. Torii, "Is this the right place? Geometric-semantic pose verification for indoor visual localization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4373–4383.

[6] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4938–4947.

[7] P. Lucivnak, "Visual localization with hololens," Master's thesis, CIIRC, CTU in PRague, 2020.

[8] "VisualLocalizationInDynamicEnvironments," https://github.com/dubenma/VisualLocalizationInDynamicEnvironments, accessed: 2022-5-20.

[9] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building rome in a day," *Communications of the ACM*, vol. 54, no. 10, pp. 105–112, 2011.

[10] A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5974–5983.

[11] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon, "Scene coordinate regression forests for camera relocalization in rgb-d images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2930–2937.

[12] F. Walch, C. Hazirbas, L. Leal-Taixe, T. Sattler, S. Hilsenbeck, and D. Cremers, "Image-based localization using lstms for structured feature correlation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 627–637.

[13] T. Sattler, M. Havlena, F. Radenovic, K. Schindler, and M. Pollefeys, "Hyperpoints and fine vocabularies for large-scale location recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2102–2110.

[14] T. Sattler, B. Leibe, and L. Kobbelt, "Efficient & effective prioritized matching for large-scale image-based localization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 9, pp. 1744–1756, 2016.

[15] F. Camposeco, T. Sattler, A. Cohen, A. Geiger, and M. Pollefeys, "Toroidal constraints for two-point localization under high outlier ratios," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4545–4553.

[16] S. Cao and N. Snavely, "Minimal scene descriptions from structure from motion models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 461–468.

[17] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, H. Chen, R. Vedantham, R. Grzeszczuk, and B. Girod, "Residual enhanced visual vectors for on-device image matching," in *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. IEEE, 2011, pp. 850–854.

[18] T. Sattler, M. Havlena, K. Schindler, and M. Pollefeys, "Large-scale location recognition and the geometric burstiness problem," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1582–1590.

[19] A. Torii, R. Arandjelovic, J. Sivic, M. Okutomi, and T. Pajdla, "24/7 Place Recognition by View Synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[20] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for fine-grained visual recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1449–1457.

[21] R. Arandjelović and A. Zisserman, "Three things everyone should know to improve object retrieval," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 2911–2918.

[22] R. Arandjelovic and A. Zisserman, "All about vlad," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2013, pp. 1578–1585.

[23] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman, "Total recall: Automatic query expansion with a generative feature model for object retrieval," in *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[24] O. Chum, A. Mikulik, M. Perdoch, and J. Matas, "Total recall ii: Query expansion revisited," in *CVPR 2011*. IEEE, 2011, pp. 889–896.

[25] P. Gronat, G. Obozinski, J. Sivic, and T. Pajdla, "Learning and calibrating per-location classifiers for visual place recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 907–914.

[26] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, sep 2014. [Online]. Available: https://arxiv.org/abs/1409.1556v6

[27] "Mattreport Pro 3D Camera," https://matterport.com/cameras/pro2-3D-camera, accessed: 2022-5-20.

[28] "Matterport API," https://api.matterport.com/model/docs/reference, accessed: 2022-5-20.

[29] "cube2sphere 0.2.1," https://pypi.org/project/cube2sphere/, accessed: 2022-5-11.

[30] "Blender," https://www.blender.org/, accessed: 2022-5-16.

[31] "matterport_panoramas_download github repository," https://github.com/dubenma/matterport_panoramas_download, accessed: 2022-5-20.

[32] "Microsoft Hololens," https://www.microsoft.com/en-us/hololens, accessed: 2022-5-20.

[33] "Artwin: Industry & Construction 4.0 Solutions," https://artwin-project.eu/, accessed: 2022-5-14.

[34] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[35] "obj2gltf github repository," https://github.com/CesiumGS/obj2gltf, accessed: 2022-5-20.

[36] "habitat_ros_semantic_2 github repository," https://gitlab.com/ferbrjan/habitat_ros_semantic_2, accessed: 2022-5-20.

[37] "OpenCV," https://opencv.org/, accessed: 2022-5-20.

[38] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-Time Instance Segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[39] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[40] K. Lebeda, J. Matas, and O. Chum, "Fixing the locally optimized ransac–full experimental evaluation." Citeseer, 2012.

[41] C. Liu, J. Yuen, A. Torralba, J. Sivic, and W. T. Freeman, "Sift flow: Dense correspondence across different scenes," in *European conference on computer vision*. Springer, 2008, pp. 28–42.

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Dubeňová Martina** Personal ID number: **487221**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Visual Localization in Dynamic Environments

Master’s thesis title in Czech:

Vizuální lokalizace v dynamickém prostředí

Guidelines:

1) Review the state of the art in indoor visual localization, see [1,2,3,4,5] and references therein.
2) Adjust method [5] to home/medical environments. Create a new data set for home/hospital environments from Matterport scan and simulate segmentation of dynamic objects moving in the environment.
3) Investigate the influence of scene changes on the accuracy of localization of the method from 2) and suggest improvements providing robustness against moving objects.
4) Demonstrate and evaluate the improved method on the new data set.

Bibliography / sources:

[1] Arandjelović, R.; Gronat, P.; et al. NetVLAD: CNN architecture for weakly supervised place recognition. In IEEE Conference on Computer Vision and Pattern Recognition, 2016.
[2] Taira, H.; Okutomi, M.; et al. InLoc: Indoor Visual Localization with Dense Matching and View Synthesis. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 2018, ISSN 1063-6919, pp. 7199–7209, doi:10.1109/CVPR.2018.00752.
[3] H Taira, I Rocco, J Sedlar, M Okutomi, J Sivic, T Pajdla, T Sattler, A Torii. Is This The Right Place? Geometric-Semantic Pose Verification for Indoor Visual Localization CVPR 2019.
[4] P-E. Sarlin, D. DeTone, T. Malisiewicz, A. Rabinovich. SuperGlue: Learning Feature Matching with Graph Neural Networks. CVPR 2020.
[5] P. Lucivnak. Visual Localization with HoloLens. MSc Thesis, CIIRC CTU in Prague 2020.

Name and workplace of master’s thesis supervisor:

doc. Ing. Tomáš Pajdla, Ph.D. Applied Algebra and Geometry, CIIRC

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **01.10.2020** Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **30.09.2022**

doc. Ing. Tomáš Pajdla, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature