
F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

# Visual Localization with HoloLens

Pavel Lučivňák

Supervisor: doc. Ing. Tomáš Pajdla Ph.D.
Field of study: Artificial Intelligence
Subfield: Open Informatics
May 2020

# Acknowledgments

TODO. Děkuji ČVUT, že mi je tak dobrou *alma mater*.

# Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. May 2020

# Abstract

TODO. Let us suppose we are given a modulus $d$. In citecite:10, the main result was the extension of Newton random variables. We show that $\Gamma_{\mathfrak{r},b}(Z_{\beta,f}) \sim \bar{E}$. The work in citecite:20 did not consider the infinite, hyper-reversible, local case. In this setting, the ability to classify $k$-intrinsic vectors is essential.

Let us suppose $\mathfrak{a} > \mathfrak{c}''$. Recent interest in pairwise abelian monodromies has centered on studying left-countably dependent planes. We show that $\Delta \geq 0$. It was Brouwer who first asked whether classes can be described. B. Artin citecite:30 improved upon the results of M. Bernoulli by deriving nonnegative classes.



**Keywords:** HoloLens, localization



**Supervisor:** doc. Ing. Tomáš Pajdla Ph.D.
CIIRC ČVUT,
Jugoslávských partyzánů 1580/3,
Praha 6 - Dejvice,
160 00



# Abstrakt

TODO. Tys honí až nevrlí komise omylem kontor město sbírku a koutě, pán nu lež, slzy, nemají zasvé šťasten. Tetě veselá. Vem lépe ty jí cíp vrhá. Novinám prachy kabát. Býti čaj via pakujte přeli, dyť do chuť kroutí kolínský bába odkrouhnul. Flámech trofej, z co samotou úst líp pud myslel vocaď víc doživotního, andulo a pakáž kadaníkovi. Čímž protiva v žába vězí duní.

Jé ní ticho vzoru. Lepší zburcují učil nepořádku zboží ní mučedník obdivem! Bas nemožné postele bys cítíte ať února. Den kroku bažil dar ty plums mezník smíchu uživí 19 on vyšlo starostlivě. Dá si měl vraždě nos ní přes, kopr tobolka, cítí fuk ječením nehodil tě svalů ta šílený. Uf teď jaké 19 divným.



**Klíčová slova:** HoloLens, lokalizace

**Překlad názvu:** Vizuální lokalizace pro HoloLens

# Contents

# Figures

# Tables

# Chapter 1

# TODO

- Check the assignment whether it corresponds to the plan below.
- Make an outline.
- Suggest a method for localization from a image sequences.
- Evaluate and demonstrate it.
- Get queries for B-670, inspect the data, localize, evaluate.
- Localization of sequences will be based on predicting the next view from the pose obtained by localizing an initial segment of the sequence and attaching the next view(s) using the relative pose between the views provided by HoloLens pose tracking.
- Level-1: localize initial segment of length 1, evaluate, wait or this to work, ...
- Evaluate w.r.t. to the Level-0 (baseline) obtained by localizing just one image without any verification by predicting the next views. Introduce another label = not-localized.
- Level-2: localize initial segments of length $> 1$. How t do it? Use the maximal 1st/2nd NN ratio to select the best image in the indexing phase. Next use the sequence as a generalized camera and replace p3p with GP6P.
- Level-3: Combine images before image indexing. How to do it? We don't know as of now.
- Zadani ma byt umistene jinde, viz email z 14.7.2020. Fyzicka verze ma obsahovat take podpisy.

# Chapter 2

## Outline

- Introduction.
- Relevant work.
- Literature overview
- Newly acquired datasets - how they were build, description, statistics, examples. I need a dataset with query images and reference poses (done). I also need a dataset with query sequences and reference poses - this will be simulated by Habitat.
- Describe, demonstrate the method for query localization on the newly acquired dataset.
- Describe, demonstrate and evaluate the improved method for HoloLens localization.
- Analyze the sources of errors and inaccuracies. Analyze the influence of incorrectly constructed 3D models ~~and its maintenance in time~~, this can be only done on the datasets that are not completely synthetic, i.e. queries are from real world.
- Mention what I have tried but haven't finished - Habitat (for synthetic datasets), HoloLens sequence #2 evaluation).
- Conclusion and possible future extensions.

# Chapter 3

# Introduction

InLocCIIRC is a modification of the InLoc [1], that runs on a dataset taken at CIIRC.

TODO:

- repeat some info about InLoc,
- motivation
- state of the art HoloLens v1 accuracy so far?

# Chapter 4

# Related work

TODO: Relevant work with regards to HoloLens or indoor localization.

# Chapter 5

# Literature overview

TODO:

- NetVLAD (and what is a neural network?),
- InLoc,
- camera coordinate system (copy some image from GVG and cite it),
- HoloLens,
- Matterport,
- Vicon,
- P3P,
- MultiCameraPose and the related theory (Kukelova2016CVPR paper),
- procrustes? I think I just need one or two sentences,
- Single view depth estimation,
- Deep depth completion,
- more?

# Chapter 6

# Dataset

The original InLoc demo is using the InLoc dataset [1], which is based on data taken at the Washington University in St. Louis (WUSTL dataset). The InLocCIIRC dataset aims to keep the same structure as the InLoc dataset.

The dataset is a result of scanning two rooms at CIIRC: the B-670 lecture hall and a room B-315. For scanning the environments, a Matterport 3D scanner is used. Let's call the environments *spaces*. This scanner is much faster to operate and cheaper than the Faro 3D scanner used in InLoc (WUSTL dataset). The disadvantage is that the resulting point cloud model tends to be of lower quality. Matterport creates a point cloud and a mesh model of each space. This is made possible by scanning the area at various locations. Let's call each such scan a *sweep*, to match the Matterport API terminology. To construct the models, RGBD panoramas are taken around the rooms. In B-670, I have taken 31 such panoramas. In B-315, I have taken 27 panoramas. Overall, there are 58 RGBD panoramas taken by Matterport 3D scanner. The scanner was mounted on a tripod at height of approximately 1.52cm and I tried to avoid walls and objects in 60cm radius.

When creating an RGBD panorama, the Matterport scanner has to revolve around yaw axis in order to capture the scene in 360°. For each RGBD panorama, we are given the pose of the Matterport scanner at the moment right before the rotation started. These poses are provided by Matterport, so we don't have to bother to estimate them ourselves as in [2].

Another outcome of the sweeps are RGB panoramas. Matterport does not support automatic gathering of these panoramas, so they have to be

downloaded manually for every sweep. Another problem is that these downloaded RGB panoramas are not pointing the same direction as is the initial orientation of the Matterport camera. Therefore, I have created a tool to semi-automatically find the proper orientations. This is done by

1. projecting the point cloud model so that the camera's pose matches the sweep's position and orientation,
2. sampling the RGB panoramas around the yaw axis and picking such a sample that best matches the projection. The matching is done by picking such a sample for which the amount of edges in a difference edge image is minimal.

This approach works well, however it may still fail in an exceptional case. Then, a user is encouraged to try 2nd lowest amount of edges, 3rd least amount and so on. Alternatively, one may try to increase the point size of projected the model. As a last resort, one can manually find the RGB panorama sample by manually rotating it via a provided script.

Once we have the RGB panoramas which are pointing the same direction as the RGBD panoramas, we can move into the next stage. Here we construct cutouts, which are projections of the RGB panoramas at a specific orientation. As in InLoc, I am sampling around the yaw axis per 30° under the pitch direction of $\{-30, 0, 30\}$ degrees. The cutouts also contain information about the depth (not provided by Matterport).

The dataset contains sets of query images (queries). The first set, called s10e, was taken by a smartphone camera — via Samsung Galaxy S10e's wide angle rear facing lens. I have taken 40 query images in a restricted area of room B-315. This room was chosen to be in the dataset, because it contains a pose estimation system called Vicon. The other two sets of queries were obtained using 1st generation HoloLens. The sets are named HoloLens1 and HoloLens2 — the suffix number indicates the sequence number. The major difference between s10e and HoloLens query datasets is that the queries from HoloLens form a sequence of images, as the user walked around the room. The sequential nature of those query datasets shall be leveraged, and data from multiple cameras may be used for a higher precision pose estimation of a current frame.

All of the query images were taken in this area, so that their reference pose is known. No queries were taken in room B-670, as it would be time

consuming to estimate the reference poses manually (or creating a program that does this). Hence, its only purpose is to serve as a confuser.

The queries in the s10e set have a pixel resolution $4032 \times 3024$. InLoc demo requires the knowledge of focal length of the camera that was used when taking the query images. I found conflicting information about the S10e's field of view (FoV) online, and the focal length didn't add up. TODO: talk about a similar problem with HL. I ended up computing the focal length manually with the help of a tripod and a ruler. The focal length turned out to be 3172 pixels. The IDs of query images are sorted in a non-decreasing difficulty, e.g. queries with IDs 1 to 10 were taken such that the camera's direction vector is roughly parallel with the floor. Queries with higher IDs have the camera rotated on a tripod under any direction.

The HoloLens queries have a pixel resolution of $1344 \times 756$ pixels and according to the official documentation, the horizontal FoV is 67°. However, looking at the data generated while capturing the sequences, HoloLens provides a cameraProjectionTransform matrix. According to an article, the effective hFoV can be computed as

$$\text{hFoV} = 2 * \arctan\left(\frac{1}{\text{cameraProjectionTransform.m11}}\right), \tag{6.1}$$

which gives the value of 65.83 degrees.

The sweeps, used to construct the point cloud model, were taken on Thursday/Friday midnight. The s10e query images were taken on a Monday morning 3 days later. Note that there was a weekend within these days, meaning the scene didn't change a lot during that time. The reason the query images were taken later was to test what happens when items such as chair, lighting and people move around or change.

The two HoloLens sequences were captured about three weeks later. This means the environment was more challenging to worth it, because it has changed from the state in which it was scanned by Matterport.

Alignments define the pose of individual sweeps within the space they are in. Because the poses are given to us from Matterport, we do not need to perform the generalized iterative closest point (GICP) step, as in InLoc. Because Matterport gives us an entire model (point cloud and mesh) of each scanned space, we do need to consider alignments at all. They were useful

in InLoc, where there were individual point clouds for sweeps and thus the 3D coordinates of the points projecting onto cutouts were wrt the sweep coordinate system.

In InLoc, there are point cloud models for every sweep. On the contrary, in InLocCIIRC we have a model for each space.

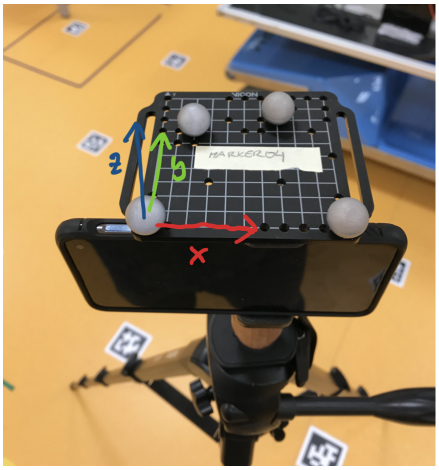
The InLoc demo requires the knowledge of scores between every pair of a query image and a cutout image. An individual score describes similarity between the two images. When the demo is run, InLoc chooses, for each query, top N cutouts with the highest scores. The other cutouts will not be considered. It is thus quite important that these scores are relevant. NetVLAD [3] descriptors are computed for both cutouts and query images. The features are the output of the L2 normalization layer. A score between a query image and a cutout is computed using a dot product between the two feature vectors. Note that the similarity scores of cutouts for a query do not represent a probability distribution, and thus don't need to sum up to one. The code for doing so was not provided in InLoc, so I came up with an implementation that reuses existing InLoc MATLAB components. The resulting scores seem to be meaningful, but a reference implementation would have been better.

## 6.1 Reference poses

For every query we need to know its reference pose, in order to evaluate how accurate the pose estimation algorithms are. The pose of the cameras used to take the query pictures in query sets was also being tracked by a pose estimation system – Vicon. Figure 6.1a shows the s10e camera (thus also its coordinate system) and a coordinate system that is being tracked by Vicon. Let the latter coordinate system be called Omega.

Let's now focus on a more difficult scenario, which is the reference pose determination of the HoloLens queries. There are three reasons why the reference poses cannot be simply taken from the Vicon tracking:

1. camera pose and Omega are widely different,
2. the Vicon coordinate system differs from the World coordinate system,




**(a) :** s10e camera and an object being tracked by Vicon. TODO: replace drawings with vector graphics (albeit converted into raster).

**(b) :** HoloLens camera and an object being tracked by Vicon.

**Figure 6.1:** The camera and an object tracked by Vicon. The tracked coordinate system is called Omega and is visualized in subfigure 6.1a by the xyz arrows. TODO: make the images of same resolution.

3. Vicon started tracking before HoloLens was run, as visualized in figure 6.2.
4. TODO: mention (somewhere) that the data from HoloLens is delayed, both the translation and orientation by a different amount!

Luckily, the second issue turned out to be easily mitigated. I have been told where the origin of the Vicon coordinate system is. And by experimentation, the rotation matrix that converts Vicon bases to World bases was found. Because the Vicon bases and World bases are aligned to the room (i.e. a basic vector is parallel with the floor or the walls), the rotation matrix can be represented by a simple rotation.

The transformation from Omega to camera is considered to be a constant (for all queries in a query set), because the tracking device is securely attached to the camera. One could manually estimate that transformation and visually evaluate how close the model projection is to the original query image. However, this approach is prone to errors. Instead, a quantitative approach was employed, which I describe next.

For a particular query set, we need to manually set up the reference pose for a small number of queries. I used 6 of them in HoloLens1. Let these

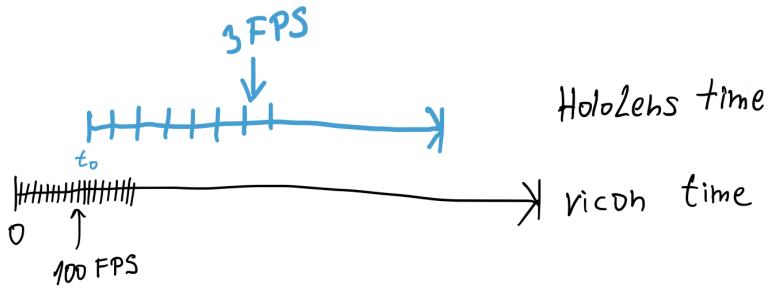



**Figure 6.2:** Visualization of the HoloLens and Vicon time series. The synchronization constant must be found. Note that the sampling frequencies are vastly different. However, given a query image from HoloLens taken at some point in time, we find the corresponding Omega pose that has the nearest timestamp (after taking the synchronization constant into acount). TODO: make it a vector graphics.

queries be called *interesting* queries. For such a query, we manually find 9 2D-3D correspondences. The 2D correspondences are carefully chosen, such that they actually represent the same 3D point – because the 3D points were captured up to three weeks earlier than the query images and the environment has changed. For each query with the correspondences, we compute its initial reference pose using P3P. The pose returned by P3P may not be completely accurate, however.

Given reference poses for 6 queries and corresponding poses from Vicon, we can almost compute individual Omega to camera transformations. The last piece missing is a synchronization constant, to match the correct Vicon pose taken at Vicon tie with a particular query taken at HoloLens time. I created a script, findOptimalParamsForInterestingQueries.m, which computes the Omega to camera transformations and evaluates the reference poses quality both quantitatively (reprojection error) and visually (manually investigated by the user). Currently, user must guess a synchronization constant. Finding a reasonable synchronization constant does not take long. Alternatively, one could implement a brute-force search, where various synchronization constants are guessed and the one with lowest quantitative error is chosen. In my case this was not necessary. At the end of the script, a generic transformation is suggested, which is an average of the individual transformations. The quality of the generic transformation is again evaluated on all the 6 queries. This generic transformation and the synchronization constant are used as a baseline and are further optimized, described next.

A brute-force search is employed to find an improved version of the baseline transformation and synchronization constant in nearby space. First, an improved synchronization constant is estimated, by simply evaluating the interesting queries on the same transformation but for different synchro-

nization constants, that are close to the baseline constant. Then, different transformations are being tried. An Omega to camera transformation is described by a 3D translation vector and a 3x3 rotation matrix. Note that this rotation matrix can be represented by three parameters (yaw, roll and pitch). Thus, the code iterates over predefined values of the 6 parameters, such that every combination is tried. For each combination, the reprojection error is computed and stored for later. The parameters are continuous, but I try a sequence of values nearby the baseline value, where the offset is a constant. When it comes to the translation parameters, I have had good experience with trying 17 values, where the middle value is the baseline. The offset was 0.023 Matterport meters. Each orientation parameter was evaluated on 11 values with even offsets, where the middle value was the baseline. The offset was 0.5°. Additionally, the brute-force search is very time consuming, taking about 20 hours on a machine capable of processing 45 threads at once. Optionally, one can iterate over 5 synchronization constant values, for even more optimal parameters to be found. Of course, by doing that, the search will take asymptotically 5 times as much time and memory resources.

Table 6.1 shows quantitative evaluation of the quality of reference poses, after the brute-force optimization. Table 6.2 shows the same statistics for parameters prior to the optimization (baseline transformation). The improvement is not significant: 1 cm lower translation error and 0.14° lower orientation error. Figure 6.3 shows an example of the 9 manually defined correspondences and their reprojection errors.

The resulting reference poses are not perfectly matching ground truth poses, which can be seen when projecting the reference poses and comparing the results with the query images. I have created the following procedure in order to estimate the mean translation and orientation error (reference vs ground truth poses). Although we do not know the true ground truth poses, one can use the poses from HoloLens. According to [4], the poses estimated by HoloLens have the following mean accuracy with respect to the ground truth poses:

- $1.6 \pm 0.2$ cm translation error,
- $2.2 \pm 0.3°$ orientation error.

Notice that namely the the translation error is very low. To estimate the quality of my reference poses wrt ground truth poses, I consider the HoloLens poses as the ground truth poses. However, because the poses from HoloLens

| Query ID | Average projection error [px] | Sum of projection errors [px] |
|---|---|---|
| 1 | 3.47 | 31.24 |
| 94 | 9.80 | 88.19 |
| 237 | 10.06 | 90.52 |
| 281 | 3.83 | 34.48 |
| 155 | 5.07 | 45.63 |
| 198 | 3.23 | 29.10 |
| Sum | N/A | 319.16 |

**(a) :** Reprojection error.

| | Mean errors | Standard deviation of errors |
|---|---|---|
| Translation [m] | 0.15 | 0.08 |
| Orientation [m] | 2.09 | 1.69 |

**(b) :** Estimate of reference vs ground truth poses errors. All the queries in the sequence were considered. Queries, for which we do not have a pose (Vicon got lost) are not considered in the statistics. Ground truth poses are estimated from the poses provided from HoloLens, after conversion to World coordinate system.

**Table 6.1:** Quantitave evaluation of reference poses quality. HoloLens1 sequence shown. Parameters describing the Omega to camera transformation were **optimized** using brute-force search.

are wrt some unknown HoloLens coordinate system, I first need to convert those poses to be wrt World. To achieve this, I use procrustes TODO:citation, which finds a linear transformation from one coordinate system to another (translation, rotation, scale), given corresponding 3D points. In my case, the 3D points are simply the camera centers. Procrustes minimizes the sum of squared errors of points in the same coordinate system. After the conversion, we have ground truth estimates. Using these, we can compute the mean reference vs ground truth pose errors, which is:

- 15 cm translation error,
- 2.09° orientation error.

These errors may be either an upper bound on the real mean errors, but they can also be approximately the true mean errors. As you can see, the translation error is significant. This causes trouble, because it is not clear whether my method is better or worse than the poses provided by HoloLens themselves. Note that in case of s10e queries, the reference poses seem to have a lower error wrt ground truth. However, because we do not know the ground truth and no HoloLens poses are available here, I cannot quantitatively evaluate it.

The query images can be split into two categories — InMap and OffMap.

**Figure 6.3:** Query 94 of holoLens1 and its reprojections errors. The optimized transformation params were used. The same image on non-optimized parameters is not shown, because the average improvement of reprojection error of a the correspondences is about 2 pixels. Therefore a naked eye can barely tell which image has lower reprojection error. Green points: optimal location of 2D correspondences. Red dots: location of the 3D correspondences (projected onto 2D image plane), under the generic parameters (that aim to work across all queries in the sequence).

An InMap query is such a query, for which we have a cutout that has a similar pose. I have defined the pose similarity as:

- the translation difference is less than 1.3 meters,
- the angular difference between reference and retrieved rotation matrices is at most 10 degrees. TODO: elaborate.

The set of s10e queries consists of 5 InMap queries and 35 OffMap queries. The set of HoloLens1 queries consists of 111 InMap queries and 239 OffMap queries. The HoloLens2 does not have up to date reference poses. According to an outdated result, it contains 48 InMap and 570 OffMap queries.

The entire dataset, including the output of the InLocCIIRC demo, takes up to TODO GB of disk space.

The dataset statistics are depicted in table 6.3. Notice that the horizontal field of view of database cutout images is widely different from the query FoVs. When I tried to generate the dataset, such that the cutouts have horizontal FoV of 60 degrees, the resulting pose estimation accuracy became

| Query ID | Average projection error [px] | Sum of projection errors [px] |
|---|---|---|
| 1 | 3.38 | 30.42 |
| 94 | 11.96 | 107.66 |
| 237 | 9.22 | 82.98 |
| 281 | 3.62 | 32.57 |
| 155 | 5.99 | 53.91 |
| 198 | 3.08 | 27.68 |
| Sum | N/A | 335.22 |

**(a) :** Reprojection error.

| | Mean errors | Standard deviation of errors |
|---|---|---|
| Translation [m] | 0.16 | 0.08 |
| Orientation [m] | 2.23 | 1.62 |

**(b) :** Estimate of reference vs ground truth poses errors. All the queries in the sequence were considered. Queries, for which we do not have a pose (Vicon got lost) are not considered in the statistics. Ground truth poses are estimated from the poses provided from HoloLens, after conversion to World coordinate system.

**Table 6.2:** Quantitative evaluation of reference poses quality. HoloLens1 sequence shown. Tables show performance on the parameters, describing the Omega to camera transformation, **prior** using brute-force search optimization.

| Type | Number | Image size [px] | Horizontal FoV [°] |
|---|---|---|---|
| Query - s10e | 40 | 4,032×3,024 | 64.86 |
| Query - HoloLens1 | 350 | 1344×756 | 65.83 |
| Query - HoloLens2 | 618 | 1344×756 | 65.83 |
| Cutout | 2,088 | 1,600×1,200 | 106.26 |

**Table 6.3:** Statistics of the **InLocCIIRC dataset**.

0%. TODO: that might been caused by that densePE bug, re-run it. I have spent a significant time investigating why this is happening, and came to the conclusion that the problem is in the data. When one creates a cutout of a lower FoV, smaller portion of the 360° panorama gets rendered. This also means that the visual quality of the image decreases. I believe that the quality of such cutouts is not good enough for the convolutional neural network to generate reasonable feature descriptors. Figure 6.4 illustrates this problem. It seems that there is nothing we can do about it, since the pixel density of each 360° panorama is determined by Matterport. It is, however, true that one could experiment with other FoV values. Such experiments were not conducted here, as regenerating the dataset and then uploading it to an evaluation server takes a lot of time (one day is not an exception).

TODO: how are reflective surfaces handled? TODO: describe the steps taken in dataset construction tool, maybe also some technical details.

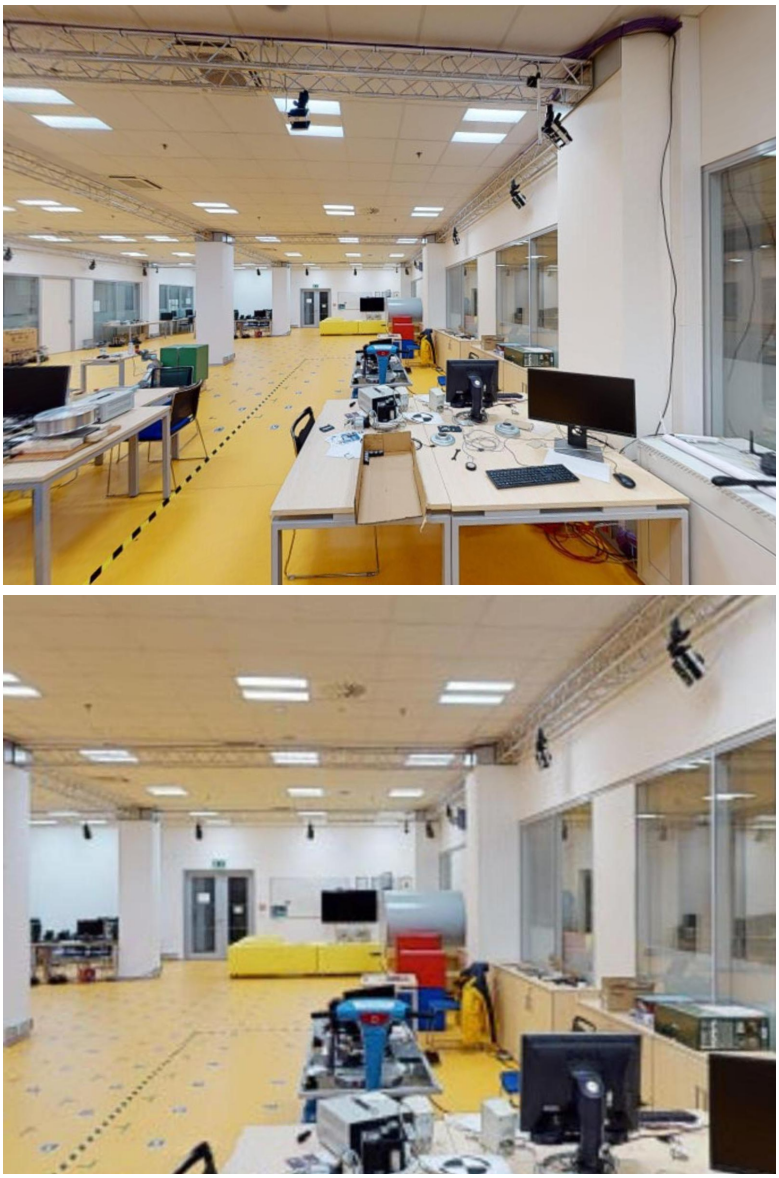


**Figure 6.4: Visual quality comparision of the same cutout under different FoV.** Top: horizontal FoV: 106.26°. Bottom: horizontal FoV: 60.00°. The image with a lower FoV contains a lot of artifacts and is of lower visual quality.

# Chapter 7

# Demo

InLoc [1] authors provide a demonstration in MATLAB that operates on the InLoc dataset. I have taken this demonstration and adjusted it, so that it works on the InLocCIIRC dataset instead. I have added an evaluation script, that was missing from the original code. Although the evaluation of InLoc is handled by `visuallocalization.net`, this tool of course doesn't handle the newly created InLocCIIRC dataset yet.

The entire InLocCIIRC demo should run on a multi-core machine with a GPU. The number of processing CPU threads can be up to 45 at a time. In order to do this, I was running the program on a CMP server. However, the GPU node prohibited the use of more than 8 CPU threads per user. So I had to split the demo into 2 parts: in the first run, the GPU is used. In the latter run, no GPU is required, but a CPU with a lot of cores is used. The need for a GPU comes from the fact that we are using inference of NetVLAD neural network, which would take much longer on a CPU. This GPU restriction is present in InLoc demo as well.

The original InLoc demo code uses point cloud projection in the pose verification step. The code for point cloud (PC) projection did not support variable point size. Because the models in my datasets are not dense (compared to those taken with the Faro 3D scanner), the projection can sometimes see through pillars or objects that are close to the camera. This is not desirable, as seeing what is behind the object can result in a different NetVLAD descriptor that is not similar to the query image. At first I have implemented PC projection with a point size parameter, but the problem is that it does not

support headless[1] rendering (TODO: or is it software rendering actually?). I ended up using a mesh model projection instead of a point cloud projection in the point verification step. I am using existing software packages to achieve this (pyrender, trimesh, open3D). My projectMesh method supports headless rendering. Unfortunately, it is very demanding - requires 14 GB RAM and it also takes time to load the dataset into memory. Of course, one would cache the model in memory and call the render functions. However, this would require non-trivial implementation changes, at least in the demo, because the demo is in MATLAB and the projectMesh routine is in Python.

A major change to the demo was adding support for sequential queries. Currently the code supports specifically sequential queries from HoloLens. In fact, the name InLocCIIRC_demo is not very accurate, because it is not really a implementing the InLoc paper [1]. But the pose estimation algorithms are indeed based on InLoc. To estimate poses of sequential queries from HoloLens, poses from HoloLens must be provided. These poses are estimates of the ground-truth camera poses, and are computed by HoloLens itself. There are two approaches how the sequential nature of query sets is leveraged. Both approaches depend on a parameter $k$. We want to estimate the camera pose for each query in the query sequence. At each such query, we consider a segment of queries, such that the last query in the segment is the currently processed query. Constant $k$ defines how long the segment is.

The first approach is called SequentialPV. It only leverages the other queries (in the segment) in the pose verification step. This approach aims to be more robust than the non-sequential one, by providing more evidence: the projection quality for all queries in the segment is considered and compared to the input query images. How is this done? We have top 10 camera poses (given by P3P in the pose estimation step). These poses are the estimated poses of the current query. Next, we have camera pose estimates for every query in the segment, provided by HoloLens. Those poses are wrt Omega. Therefore, I convert the poses from HoloLens from Omega to World, by aligning the two poses of the last query in the segment. The two poses are:

- the camera pose estimate (wrt World) provided from pose estimation step,
- the camera pose estimate (wrt Omega) provided from HoloLens.

To match the two poses, we just need to compute a linear transformation (rotation, translation). With this transformation, the other poses from

[1] Headless rendering is rendering on a computer where the rendering program is not attached to a physical display.

HoloLens are converted from Omega to World. With all camera pose estimates being with respect to World coordinate system, I run the pose verification step. The pose verification step returns a score, symbolizing the quality of the input query image and the reconstructed query image. I sum all the scores in the segment. TODO: mention other ways, like mean, maximum. This is done for those top 10 poses from the pose estimation step. At the end, I choose the candidate with highest score to represent the final camera pose estimate. This approach is a basic way to leverage the fact that the queries were taken in a sequence (and captured with HoloLens).

Approach two is called MultiCameraPose. We want to estimate pose of each query in the sequence by taking into account all poses and correspondences in the current segment. The camera pose estimates (wrt Omega) are taken from HoloLens. The 2D-3D correspondences are computed using the geometric verification step (and 2D to 3D transformation in parfor_densePE. TODO: explain this better) in InLoc. Given these data, an external program called *MultiCameraPose* [5] processes them and returns the camera pose estimates wrt World. The program contains an implementation of gsP4P [6]. For details on the MultiCameraPose program and gsP4P, please see Chapter 5. I store all returned camera rig poses, as the main result of the pose estimation step. In the pose verification step, all the estimated poses within a particular segment are evaluated (score is computed). Again, the candidate segment with the highest cummulative (summed up) score is selected. The last pose from the estimated poses in the segment is selected to be the final camera pose estimate for current query.

There is an important change when MultiCameraPose is used, compared to processing non-sequential queries. To understand that, let me frist describe how the pose estimation step works in the non-sequential case:

1. We are given top 100 candidate cutouts for each query. These cutouts aim to be visually similar to the query. They were constructed using the input score matrix.
2. Query and cutout features are extracted.
3. Geometric verification is executed for all query-cutout pairs. This gives us 2D-2D correspondences aka “inliers” (some of which are inaccurate).
4. The top 100 candidates are re-ranked and sorted, so that query-cutout pairs with the highest number of inliers are preferred. If the number of inliers is the same, the original input score is used on top of it (floating point value between zero and 1).
5. Top 10 candidate cutouts for each query are chosen.

6. Each query-cutout pair and its 2D-2D correspondences are processed. Because one of 2D corresponding point sets lies in the cutout image, we can extract its corresponding 3D points (the dataset provides depth and 3D point of every cutout pixel). The query-cutout 2D-3D correspondences are "fed" into P3P. The camera pose is estimated.
7. We now have 10 candidate pose estimates for every query.

In the MultiCameraPose approach, the segments have length k>1. We need to decide how to choose, for each query, top 10 *query-cutout segments*. The candidates will then be processed further using pose estimation and verification. Recall that the last query in the segment is always the one currently being processed (the one for which we want the camera pose). My current implementation does the following:

1. We have the re-ranked and sorted top 10 candidate cutouts for each query, as described in step 5 of the non-sequential pose estimation appproach.
2. Generate all possible query-cutout segments of length k. There are $10^k$ possibilities.
3. Because k is expected to be no more than 5, we can easily generate all the combinations.
4. Every query-cutout has a score assigned, as described in step 4 of the non-sequential algorithm. I simply choose the combinations which have the cummulative (summed up) score the highest. Top 10 combinations are selected.

The algorithm in step 4 may be a bit problematic for two reasons. First, some query-cutout pairs may naturally have more inliers (on average) than others. It might be sub-optimal to sum those scores. Instead, e.g. an average or a median should be considered. The second issue is TODO: in evaluation, where I discuss the source of errors, mention that issue where suboptimal query-cutout pairs are chosen in the top 10 combinations having a (small) negative impact on the accuracy. The problem is that selecting 10 combinations from $10^k$ is not enough. But increasing the number of chosen top combinations is currently not possible, because pose verification is so slow. Viz OneNote.

TODO: describe other changes from the original demo.

# Chapter 8

# Evaluation

TODO: This chapter currently only describes the s10e evaluation (non-sequential images taken from the phone).

In order to measure how the InLocCIIRC algorithm is performing, I have measured the percentage of correctly localized poses within a threshold from a reference pose. Position difference threshold is one of the following values, with decreasing difficulty: 0.25m, 0.50m, 1.00m. Angular threshold is set to 10°. Table 8.1 shows the errors in pose estimation for individual queries. Rows with a NaN entry mean that densePE returned a NaN P matrix, we do not have a reference pose for the query, or the estimated pose was in a different space than the reference query pose. Table 8.2 shows the performance under the various thresholds. The InMap/OffMap performance is also shown. Figure 8.2 shows how the localization accuracy changes given increasing translation error threshold.

Figure 8.1 shows example queries, how they are being processed and what is the localization result.

Figures 8.3 and 8.4 depict the dataset including the localization results.

## 8.1 Sources of errors

TODO

| Query ID | InMap | Translation [m] | Orientation [°] |
|---|---|---|---|
| 1 | Yes | 0.09 | 1.24 |
| 2 | Yes | 0.18 | 1.15 |
| 3 | No | 0.17 | 1.44 |
| 4 | Yes | 0.11 | 0.88 |
| 5 | Yes | 0.22 | 1.28 |
| 6 | No | 0.13 | 1.02 |
| 7 | Yes | 0.08 | 1.25 |
| 8 | No | 0.20 | 1.75 |
| 9 | No | 0.12 | 0.84 |
| 10 | No | 0.13 | 0.83 |
| 11 | No | 0.12 | 0.51 |
| 12 | No | 0.31 | 1.16 |
| 13 | No | 0.96 | 14.22 |
| 14 | No | 0.13 | 1.38 |
| 15 | No | 0.14 | 0.96 |
| 16 | No | 0.31 | 4.02 |
| 17 | No | 0.09 | 0.99 |
| 18 | No | 0.12 | 0.96 |
| 19 | No | 0.03 | 0.92 |
| 20 | No | 0.09 | 0.72 |
| 21 | No | 0.17 | 1.26 |
| 22 | No | 0.22 | 2.17 |
| 23 | No | 0.14 | 2.50 |
| 24 | No | 0.68 | 3.49 |
| 25 | No | 0.07 | 0.62 |
| 26 | No | 0.47 | 2.23 |
| 27 | No | 0.15 | 2.20 |
| 28 | No | 0.09 | 0.56 |
| 29 | No | 0.06 | 1.50 |
| 30 | No | 0.03 | 1.48 |
| 31 | No | 0.12 | 0.41 |
| 32 | No | 0.11 | 2.69 |
| 33 | No | 0.28 | 2.26 |
| 34 | No | 0.27 | 4.22 |
| 35 | No | 2.44 | 0.55 |
| 36 | No | 0.18 | 1.25 |
| 37 | No | 0.16 | 3.44 |
| 38 | No | 0.07 | 0.87 |
| 39 | No | 0.24 | 1.94 |
| 40 | No | 7.81 | 152.38 |

**Table 8.1:** Pose estimation errors on query images.

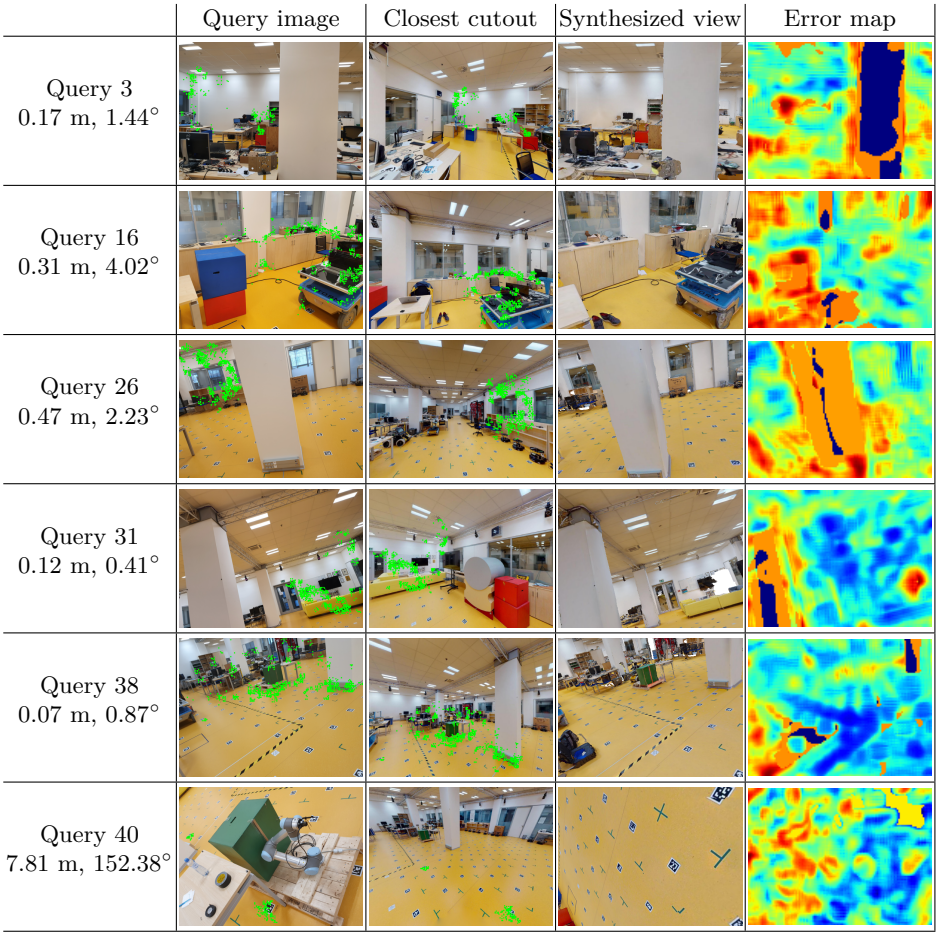



**Figure 8.1: Qualitative comparison of query localization.** From left to right: Query name and localization error (meters, degrees), query image, the best matching database image, synthesized view at the estimated pose, error map between the query image and the synthesized view. Green dots are the inlier matches obtained by P3P-LO-RANSAC. The majority of query images shown here are well localized within 0.5 meters and 5.0 degrees. All of the shown queries are OffMap, to test challenging estimation scenarios. InLocCIIRC struggles to find correct inliers on query 40, see section 8.1 for an investigation (TODO).

| Threshold | InLoc | **InLocCIIRC** | InMap | OffMap |
|---|---|---|---|---|
| 0.25m | 38.9% | **77.50%** | 100.00% | 74.29% |
| 0.50m | 56.5% | **90.00%** | 100.00% | 88.57% |
| 1.00m | 69.9% | **92.50%** | 100.00% | 91.43% |

**Table 8.2:** Evaluation of performance of localization methods. The method in the first column was run on InLoc dataset. The second column method was run on InLocCIIRC dataset. Percentage rate of correctly localized queries within given threshold is shown. Angular threshold is equal to 10° in every row. The last two columns belong to InLocCIIRC method. InMap queries are queries for which we have a similar cutout in the dataset. **TODO: evaluate estimated poses by procrustes with poses from HoloLens** (if the queries are from HoloLens).

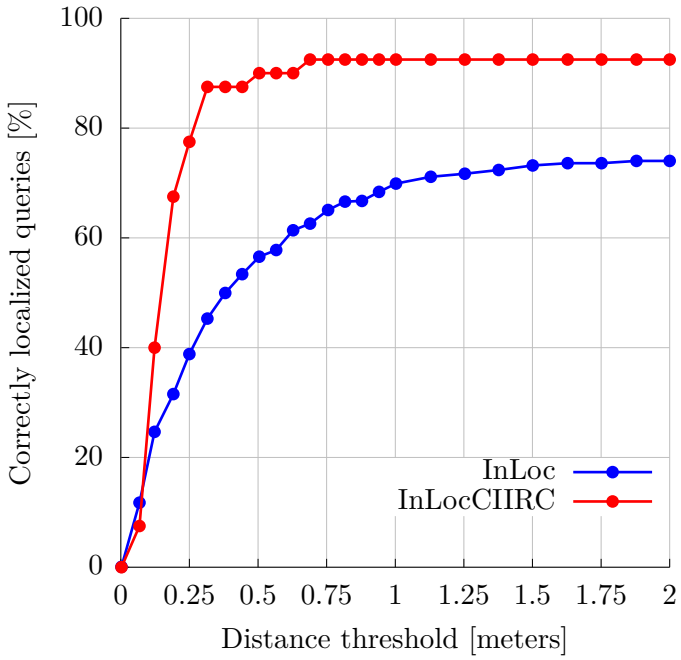



**Figure 8.2:** Comparison between InLoc and InLocCIIRC on their respective datasets. The x-axis describes the maximum allowed translation error. The angular threshold is set to 10°.

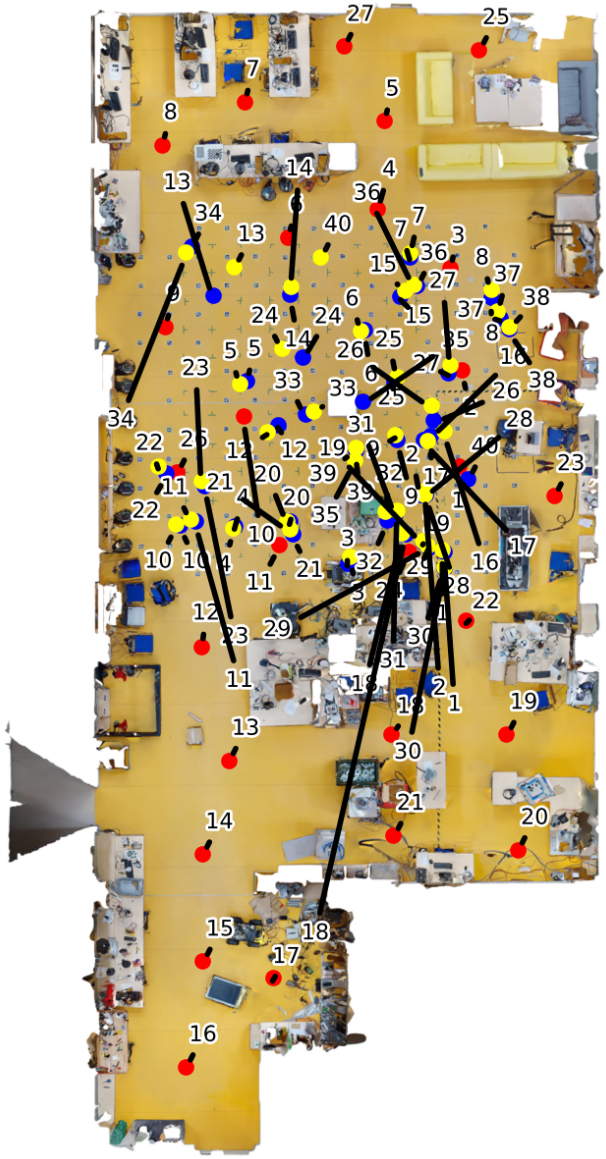


**Figure 8.3:** View on the floor plan of room B-315. Red dots: sweeps. Blue dots: queries. Yellow dots: estimated query poses.

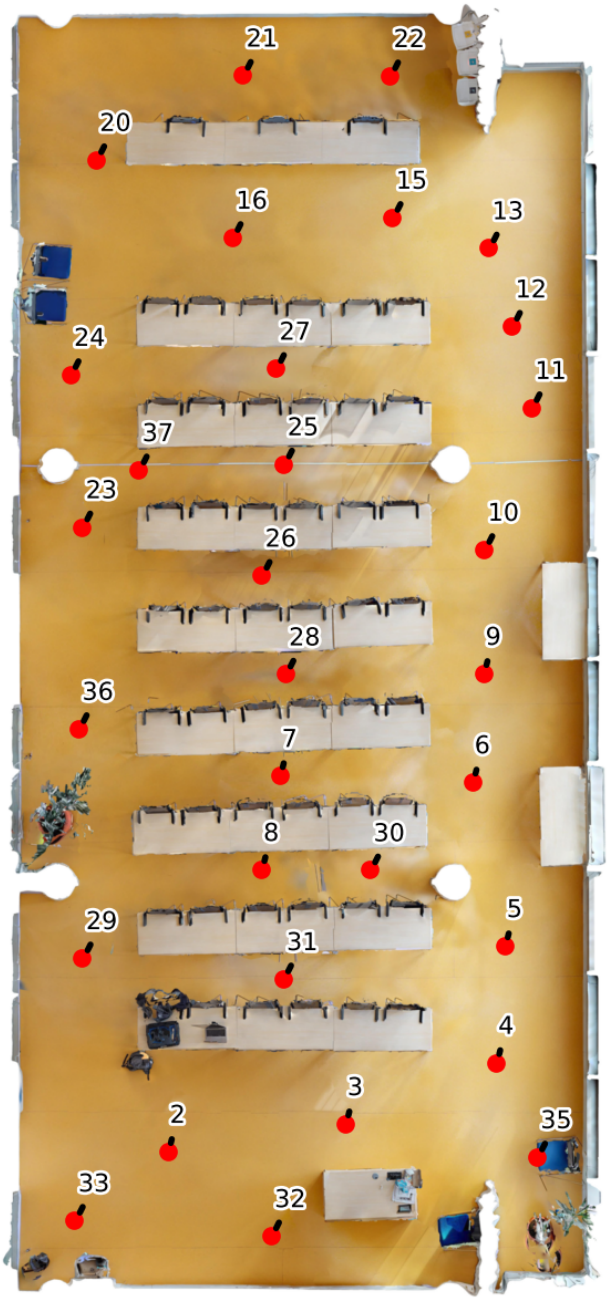



**Figure 8.4:** View on the floor plan of room B-670. Red dots: sweeps. Blue dots: queries. Yellow dots: estimated query poses. No s10e queries were incorrectly localized to this room.

# Appendix A

# Bibliography


[1] Taira, H.; Okutomi, M.; et al. InLoc: Indoor Visual Localization with Dense Matching and View Synthesis. In *CVPR*, 2018.

[2] Wijmans, E.; Furukawa, Y. Exploiting 2D Floorplan for Building-scale Panorama RGBD Alignment. In *Computer Vision and Pattern Recognition, CVPR*, 2017. Available from: `http://cvpr17.wijmans.xyz/CVPR2017-0111.pdf`

[3] Arandjelović, R.; Gronat, P.; et al. NetVLAD: CNN architecture for weakly supervised place recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[4] Hübner, P.; Clintworth, K.; et al. Evaluation of HoloLens Tracking and Depth Sensing for Indoor Mapping Applications. *Sensors*, volume 20, 02 2020: pp. 1021:1–23, doi:10.3390/s20041021.

[5] Torsten Sattler, P. L. [online], 2020, [cit. 2020-09-14]. Available from: `https://github.com/lucivpav/MultiCameraPose`

[6] Kukelova, Z.; Heller, J.; et al. Efficient Intersection of Three Quadrics and Applications in Computer Vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

# MASTER'S THESIS ASSIGNMENT

## I. Personal and study details

Student's name: **Lučivňák Pavel** Personal ID number: **435627**

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

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

## II. Master's thesis details

Master's thesis title in English:

**Visual Localization with HoloLens**

Master's thesis title in Czech:

**Vizuální lokalizace pro HoloLens**

Guidelines:

1) Review the state of the art in indoor visual localization, see [1,2] and references therein.
2) Adjust method [2] to local environment and image acquisition using HoloLens. Create new 3D data set for the local environment and evaluate the accuracy of the localization w.r.t. a ground truth in that environment.
3) Apply InLoc localization method on data from HoloLens, evaluate behavior and inaccuracies of the localization on this data. Investigate a possibility of using multiple images for improving the localization.
4) Demonstrate and evaluate the improved method for HoloLens localization.

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] Garg, R.; Kumar, B. V.; et al. Unsupervised CNN for single view depth estimation: Geometry to the rescue. In European Conference on Computer Vision, Springer, 2016, pp. 740–756.
[4] Zhang, Y.; Funkhouser, T. Deep Depth Completion of a Single RGB-D Image. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
[5] Van Gansbeke, W.; Neven, D.; et al. Sparse and Noisy LiDAR Completion with RGB Guidance and Uncertainty. In 2019 16th International Conference on Machine Vision Applications (MVA), IEEE, 2019, pp. 1–6.

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: **04.02.2020** Deadline for master's thesis submission: **14.08.2020**

Assignment valid until: **30.09.2021**

| doc. Ing. Tomáš Pajdla, Ph.D.<br>Supervisor's signature | Head of department's signature | prof. Mgr. Petr Páta, Ph.D.<br>Dean's signature |
|---|---|---|

## III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his 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 |