# A Deep Learning Approach to Camera Pose Estimation

Federico Izzo
*DISI, University of Trento*
Trento, Italy
federico.izzo@studenti.unitn.it

Francesco Bozzo
*DISI, University of Trento*
Trento, Italy
francesco.bozzo@studenti.unitn.it

***Abstract*—The task of camera pose estimation aims to find the position of the camera in an image within a given environment. While different geometric approaches have already been studied in the literature, the aim of this project is to study and improve the performances of deep learning models for the camera pose estimation problem. In this work, we analyze models for both relative camera pose estimation (MeNet) and absolute camera pose estimation (PoseNet, MapNet). Moreover, we propose a pipeline for the generation of a ground truth dataset based on structure from motion techniques (COLMAP). Finally, we (1) show how the proposed framework has been used to build a dataset of the second floor of the Povo 1 building in the University of Trento, (2) train an absolute pose estimation model with PyTorch, (3) and deploy it through a web dashboard using FastAPI. The deep learning approach could give interesting results in combination with geometric methods, especially for: relocation after lost tracking, closed-loop detection, better dealing with moving objects in the scene. The deep learning SOTA for this field is less accurate than geometric approaches but, at the same time, mantains a more capacity of generalization reducing the computational cost.**

***Index Terms*—camera pose estimation, COLMAP, deep learning, vision**

## I. Introduction

The *camera pose*, referenced also with *camera extrinsics*, can be expressed as a combination of two components:

1) a tuple of three elements that identifies the absolute coordinates $x, y$ and $z$ in a reference space:

$$x_c = (x, y, z) \quad x, y, z \in \mathbb{R} \tag{1}$$

2) a quaternion of four elements that identifies the rotation of the camera:

$$q_c = (qw, qx, qy, qz) \quad qw, qx, qy, qz \in \mathbb{R} \tag{2}$$

Consequentially, the pose is referred as $p_c = (x_c, q_c)$.

It is important to notice that this is not the only available representation of a pose: other methods are based also on rotation matrices and Euler angles. It is worth specifying that even if Euler angles are the most straightforward and efficient in terms of memory consumption, they suffer from of the Gimbal lock problem. Even if rotation matrices guarantee a good representation, they are more memory expensive (9 values) than quaternions (only 4 values): for this reason the latter form is preferred here.

Given an image $I_c$ captured by a camera $C$, an absolute pose estimator $E$ tries to predict the $3D$ pose orientation and location of $C$ in world coordinates, defined for some arbitrary reference $3D$ model. The *absolute pose estimation (APE)* problem can be formally defined as the problem of estimating a function $E$ taking as input an image $I_c$ captured by a camera $C$ and as output its respective pose:

$$E(I_c) = (x_c, q_c) \tag{3}$$

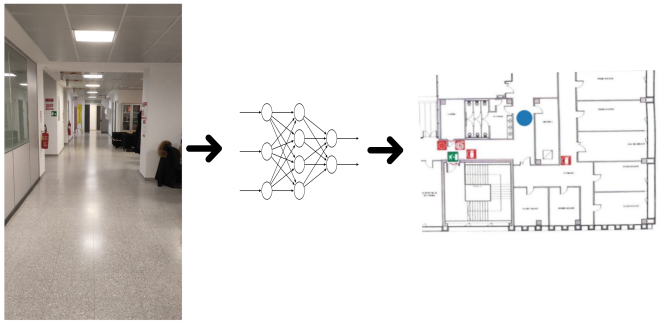

Fig. 1. APE model developed

Apart from APE, a popular task is also *relative pose estimation (RPE)*. In this kind of approach the estimator takes two images $I_c^1$ and $I_c^2$ captured by $C$ and aims to predict the relative pose between them. In this case, the formulation of the function $E$ described in Equation (3) is a little different, since it receives in input two images:

$$E(I_c^1, I_c^2) = (x_c^{rel^2}, q_c^{rel}) \tag{4}$$

where $x_c^{rel}$ is defined as the absolute pose with *coordinates reference system* in $I_c^1$ or, in an equivalent way, as the translation vector from $I_c^1$ to $I_C^2$.

With this work, we show how it is possible to build a deep learning model which is able to learn the function $E$ using a data-driven approach.

In both scenarios the estimator can be viewed as a model who describes an environment that can be questioned about the pose of an element within it.

## II. Related works

In the literature there are many deep learning approaches used to perform RPE and APE: here we focus on MeNet for the first and PoseNet [1] and MapNet [2] for the latter.

APE deep learning models rely mostly on *transfer learning*: the idea is to use SOTA vision models to extract features from images and use them to estimate camera extrinsics. The PoseNet model has been the first to be developed following this idea. The starting network for the knowledge transfer was a GoogLeNet [3], where the softmax classification layer is replaced with a sequence of fully connected layers. Even if the obtained results are decent, the model lacks of generalization when applied to unseen scenes.

In order to solve this problem, other techniques have been developed, which can be classified in:

- *end-to-end* approaches;
- *hybrid* approaches.

Most of the end-to-end proposed models are based on the PoseNet architecture, with the addition of some components, such as *encoder/decoder blocks*, *linear layers*, and *LSTM blocks*. The most successful model on this category is MapNet and related variants MapNet+ and MapNet+PGO [2].

Hybrid approaches instead try to focus on different support tasks with the goal of helping the final pose prediction. Those techniques rely on unsupervised learning, 3D objects reconstruction and other data extracted with external tools: for this reason those methods are under the scope of out work.

## III. Dataset generation

### A. Tested approaches

The deep learning approaches explained in this document are *supervised learning* techniques that require a labeled dataset. Several paths were tested in order to generate this kind of dataset:

- *IMU sensors*: usage of gyroscope and accelerometer sensors of a smartphone to estimate the position of the camera during a video given a fixed origin point.
- *digital video*: usage of free online 3 dimensional datasets in which video can be recorder in a digital way.
- *motion capture system*: usage of a motion capture system that estimates the camera position following some tracking objects attached to the subject.
- *structure from motion techniques*: techniques that compute a sparse and dense reconstruction from a sequence of images.

The main problem encountered with IMU sensors was the high noise presence during acquisitions, the final signal was very dirty, and the resolution was not acceptable for the dataset generation. A possible solution could have been the usage of a well calibrated hardware used in other kind of contexts.

Most of the 3 dimensional acquisitions available online for free are acquired with *depth sensors* or *LIDAR sensors*, for this reason although the camera pose estimation would not have presented any errors the images would have been at low quality.

The motion capture system is able to follows the position of the tracked objects with extremely precision, the main problematic remains the association of poses to video captured from the camera held by the tracked subject. Other difficulties involved the calibration of the tool.

The techniques of structure from motion were invented with the goal of generate structures for which a huge amount of photos is available. The overall idea is to feed the algorithm with data in order to extract feature and build a recomposition of the environment. A step required in order to obtain a result is the estimation of the pose of images. These intermediate requirement have been exploited by us to generate a labeled dataset.

### B. Pipeline

The implemented pipeline require a video captured by any camera, it is not required any calibration of the sensor. It is composed by several steps:

1) video split: the captured video is split into many frames;
2) structure from motion: images obtained from the previous step are fed into a structure motion tool called *COLMAP*;
3) cross validation dataset: positions obtained during the camera estimation of the reconstruction process are split into three batches: train, validation, test.

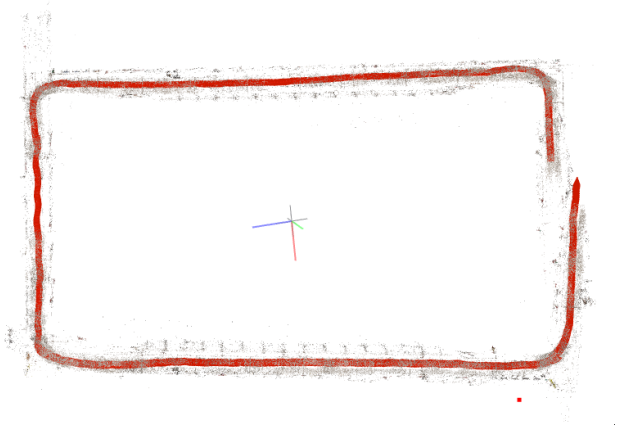

Fig. 2. Trajectory computed by COLMAP

In Figure 2 is presented the trajectory obtained with the structure from motion techinque through COLMAP. The process involes a feature extraction phase.

### C. COLMAP reconstruction

COLMAP [4] is a tool that allows to build a 3D reconstruction (model) of an environment. There exists two types of models, *sparse* and *dense* and both of them are composed by:

- a set of points $P$ which are features of the environment (dots on Figure 3);
- a set of poses $\mathcal{V}$ associated witht the images used for the reconstruction (red squares on Figure 3);

- a Coordinate Reference system.

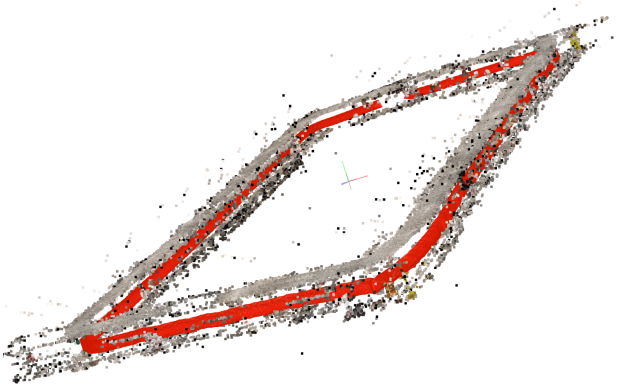

Fig. 3. Features extracted by COLMAP

The set of poses are represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and for each node there is an association with a subset of environment features $\mathcal{V} = \{R\} \quad R \subset P$. Edges $\mathcal{E}$ connect following poses.

The algorithm that creates a sparse model extracts features from each given image and build the set $P$ called *bag of words*. Once this is done, thanks to a convolution on each image, the next step is to created associations between node of the graph. The convolution enables to map only subsection of the whole image, this allows to estimate the movement based on the position of the matrix used for the convolution within the image grid. The bag of words is consulted at every step. Once associations are done the features and the poses are composed in order to create the sparse model.

The algorithm for the dense model is very similar to the one used for the sparse. Tha main differences is how the association are created, it is not used a convolution but a pointwise method. This allows to be more accurate and create a more definite cloud of points increasing the computational time.

The settings used for sparse dataset generation of Povo 1 used a sequence of images with Poisson mesher.

### D. Coordinate reference system alignment

The dataset generated by COLMAP has a *coordinate reference system (CRS)* choosen arbitrarily during the reconstruction. Origin and axes of the system may not concide with the real world CRS. In order to be able to place a prediction on a map it is required to rotate and translate until an alignment is reached.

This task was accomplished through the Euclidean or Rigid transformation, it invoves a rotation $R$, a translation $t$ and at least three points for both the CRSs that represents the same locations $A = \{(x_1^A, y_1^A), (x_2^A, y_2^A), (x_3^A, y_3^A)\}$, $B = \{(x_1^B, y_1^B), (x_2^B, y_2^B), (x_3^B, y_3^B)\}$. In Equation (5) is presented the equation of the rigid transform.

$$R \times A + t = B \tag{5}$$

Matrix $R$ and vector $t$ are obtained using *Singular value decomposition (SVD)*, it takes a matrix $E$ and return 3 other matrices, such that:

$$\begin{aligned} [U, S, V] &= SVD(E) \\ E &= USV^T \end{aligned} \tag{6}$$

The first step needed to extract the matrix $R$ is the alignment on the same origin of the datasets centroids. This is done subtracting to each coordinate of each point the centroid of the dataset. Once this is done it is possible to ignore the component of the translation $t$ and compute the rotation $R$:

$$\begin{aligned} H &= (A - \text{centroid}_A)(B - \text{centroid}_B)^T \\ [U, S, V] &= SVD(H) \\ R &= VU^T \end{aligned} \tag{7}$$

Finally it is possible to use the Equation (5) to obtain the translation vector $t$:

$$\begin{aligned} R \times A + T &= B \\ R \times \text{centroid}_A + T &= \text{centroid}_B \\ t &= \text{centroid}_B - R \times \text{centroid}_A \end{aligned} \tag{8}$$

## IV. Models

In this work we take in consideration some SOTA models for camera pose estimation, also adding small modifications to make them fit better to our use case scenario. In particular, we present:

- MeNet (add citation) for RPE;
- PoseNet [1] and MapNet [2] for APE.

### A. MeNet

The first model we would like to analyze is the MeNet model (Figure 4), which is specifically targeted for RPE. The input of the network consists in a stack of two images: the goal is to estimate the relative pose of the second image with respect to the first one. As presented in Figure 4, the MeNet is composed by nine deep convolutional layers followed by a pair of linear layers. While the first part of the network works as a feature extraction mechanism, the last layers are used to combine the extracted features.

In order to train the model, we use a loss function that consists in the weighted composition of two Mean Square Errors (MSE) computed separately on positions and quaternions:

$$Loss(w) = \frac{1}{N} \sum_{i=1}^{N} \left\| P^i - \hat{P}^i \right\|_2^2 + \alpha \left\| Q^i - \hat{Q}^i \right\|_2^2 \tag{9}$$

where $P$, $\hat{P}$, $Q$, $\hat{Q}$, and $\alpha$ are the ground truth position vector, the estimated position vector, the ground truth quaternions, the estimated quaternions, and the weight for balancing the displacement error and the rotation angle error.

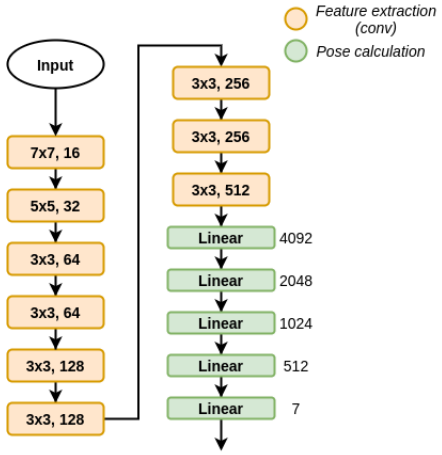


Fig. 4. The architecture of the MeNet model.

### B. PoseNet

Since the results given by RPE deep learning solutions are not very promising, from now on we are going to consider models strictly developed for APE. In this sense, the first model we present is the PoseNet model [1]. Just like the MeNet model, the PoseNet is made up by two components:

- feature extraction through a sequence of convolutional layers. This component has been named internally also as *backend*;
- pose regression on the extracted features using linear layers.

This model architecture is actually pretty convenient, since it can use pre-trained deep convolutional models, through the transfer learning approach. In most of the cases, this kind of models are trained on the ImageNet dataset [5], which counts almost 3.2 million real-world images. Some examples of SOTA backend models that have been considered are: GoogLeNet [3], ResNet [6], and EfficientNet-B7 [7]. Table I shows the accuracy over the k=(1, 5) top predictions for the used backends on the ImageNet dataset.

TABLE I
BACKENDS PERFORMANCE IN IMAGENET

| Model | Acc@1 | Acc@5 |
|---|---|---|
| GoogLeNet | 69.778 | 89.530 |
| ResNet-18 | 69.758 | 89.078 |
| ResNet-34 | 73.314 | 91.420 |
| ResNet-50 | 76.130 | 92.862 |
| ResNet-152 | 78.312 | 94.046 |
| EfficientNet-B7 | **84.122** | **96.908** |

Since ImageNet pre-trained models are used for classification purposes, we can just remove the last layers, and use only the feature extraction-related ones. In this way we are introducing in the PoseNet a feature extraction mechanism on real-world images with minimum effort: training by scratch these models would require a huge amount of computational power.

Also in this model we adopt the weighted loss described in Equation (9), also used in the MeNet model.

### C. MapNet

The MapNet model for APE represents an evolution of the PoseNet model: in fact, the model architecture remains actually the same. On the contrary, the main difference between the PoseNet is the loss function used to train the model. In this case, the errors in the prediction of absolute poses are not the only ones which are penalized: also errors in the relative poses are taken in consideration.

The size of the last linear block depends on the dimension of the map that we would like to introduce.

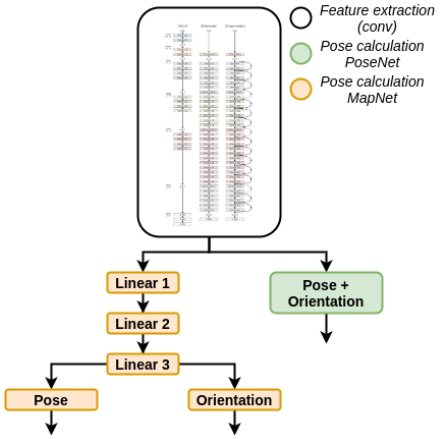


Fig. 5. The architecture of the PoseNet and MapNet models.

## V. RESULTS

### A. PoseNet

Several pretrained models can be used as features extractor in the PoseNet structure. In Table IV are presented the most powerful ones for features extraction tested on the same final linear encoder. The overall trend is similar, this highlights that the extracted features are enough for the task independently of the backbone used.

TABLE II
POSENET BACKEND COMPARISON

| Model | Position error | Rotation error | Total parameters | Trainable parameters |
|---|---|---|---|---|
| GoogLeNet | 0.781 | 0.119 | - | - |
| ResNet-18 | 0.635 | 0.288 | 11,180,103 | 3,591 |
| ResNet-34 | 0.632 | 0.223 | 21,288,263 | 3,591 |
| ResNet-50 | 0.707 | 0.191 | 23,522,375 | 14,343 |
| ResNet-152 | **0.594** | **0.139** | 58,158,151 | 14,343 |
| EfficientNet-B7 | 0.817 | 0.132 | 63,804,887 | 17,927 |

TABLE III
POSENET LOSSES COMPARISON

| Loss | Position Error | Rotation Error |
|---|---|---|
| SmoothedL1Loss | **0.594** | **0.139** |
| L1Loss | 0.906 | 0.226 |
| MSE | NaN | NaN |
| weighted_custom | NaN | NaN |

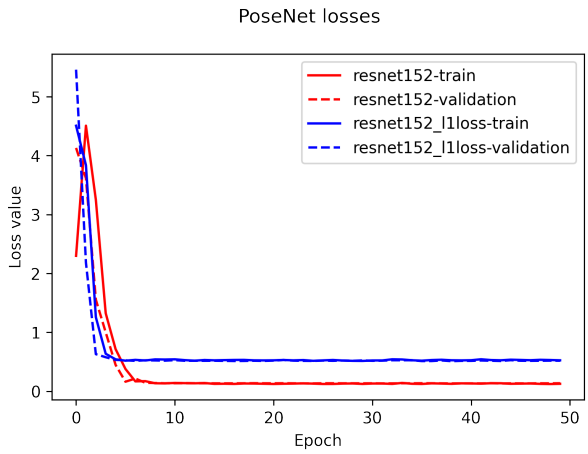


Fig. 6. PoseNet (ResNet-152) losses

### B. MapNet

Several pretrained models can be used as features extractor in the MapNet structure. In Table IV are presented the most powerful ones for features extraction tested on the same final linear encoder. The overall trend is similar, this highlights that the extracted features are enough for the task independently of the backbone used.

TABLE IV
MAPNET BACKEND COMPARISON

| Model | Position error | Rotation error | Total parameters | Trainable parameters |
|---|---|---|---|---|
| GoogLeNet | 0.225 | 0.0876 | - | - |
| ResNet-18 | 0.202 | 0.0658 | 14,853,703 | 3,677,191 |
| ResNet-34 | **0.187** | **0.0757** | 24,961,863 | 3,677,191 |
| ResNet-50 | 0.220 | 0.0969 | 30,330,951 | 6,822,919 |
| ResNet-152 | 0.233 | 0.0869 | 64,966,727 | 3,677,191 |
| EfficientNet-B7 | 0.210 | 0.0848 | 71,658,455 | 7,871,495 |

Another point that emerges is the importance of the final encoder, it works similarly of the *bag of words* used by structure from motion (link al paper). Extracted features are mapped in a space that is used later as a comparison tool for new images for which the pose is asked. For this reason the final encoder was modified from the original one (link al paper) in order to increase the latent space in which data can be stored.

### C. Comparison

### D. Dashboard

A dashboard was developed with the aim to easily allow users to interact with model inference through a web-server. In Figure 11 is presented the *UI* where red zones are not walkable areas.

## VI. MATERIALS

Every material used in the project have been uploaded respectively:

- the datasets have been uploaded on the Google Drive folder;
- the code is available in the GitHub repository.

The project has been developed in Python 3, using common data science libraries, such as numpy, pandas, PyTorch, matplotlib, scipy, and many others.

### A. Repository organization

The repository follows the structure:

- `camera-pose-estimation/`
  - `model/` contains everything related to the deep learning part of the project. It also includes the code used for implementing the web server under `webserver.py` and `static/`.
  - `tools/` contains scripts used for the dataset generation pipeline.
- `config_parser/`: Python package written by us that allows to create configuration files, with the idea of improving reproducibility in our experiments. Each configuration file can be subdivided in sections: for each section you can define variables with the sintax `label=value`, where `value` is a parsable JSON object (boolean, int, float, list, object).
- `notebooks/` contains some Python Jupyter Notebooks that have been used for data exploration, validation, and post-processing of the model predictions.

### B. Data organization

For each footage, a folder has been created:

- `imgs/` contains the video frames exported with ffmpeg;
- `processed_dataset/` contains the train, validation, and test datasets that can be reused during different trainings: this helps speeding up the loading procedure from ... minutes to ... seconds;
- `workspace/` contains the models generated by COLMAP;
- each of `train.csv`, `validation.csv`, and `test.csv` contains a table for specifying the pose for each image frame. This are the files generated with the `video_to_dataset.sh` script.

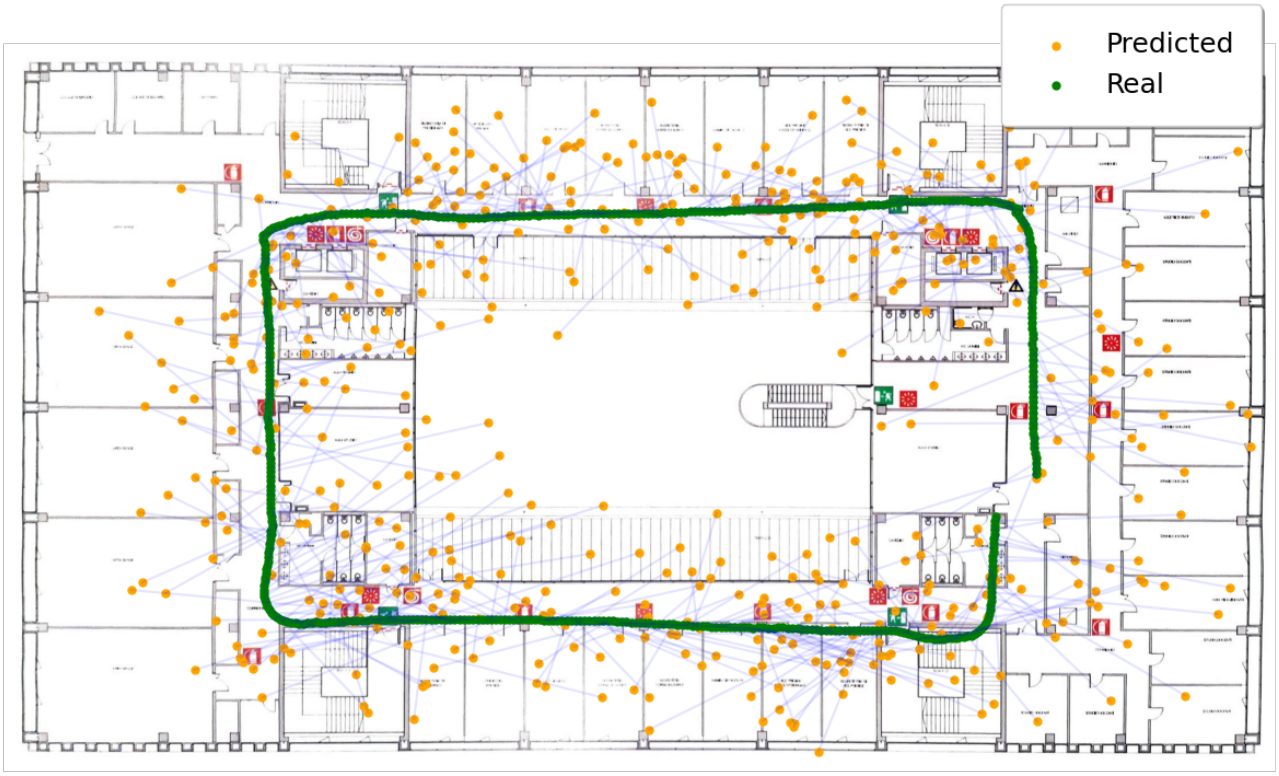


Fig. 7. Predicted trajectory PoseNet

## VII. Conclusion

## References

[1] A. Elmoogy, X. Dong, T. Lu, R. Westendorp, and K. Reddy, "Linear-posenet: A real-time camera pose estimation system using linear regression and principal component analysis," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, 2020, pp. 1–6.

[2] S. Brahmbhatt, J. Gu, K. Kim, J. Hays, and J. Kautz, "Mapnet: Geometry-aware learning of maps for camera localization," *CoRR*, vol. abs/1712.03342, 2017. [Online]. Available: http://arxiv.org/abs/1712.03342

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[4] A. Fisher, R. Cannizzaro, M. Cochrane, C. Nagahawatte, and J. L. Palmer, "Colmap: A memory-efficient occupancy grid mapping framework," *Robotics and Autonomous Systems*, vol. 142, p. 103755, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889021000403

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[7] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

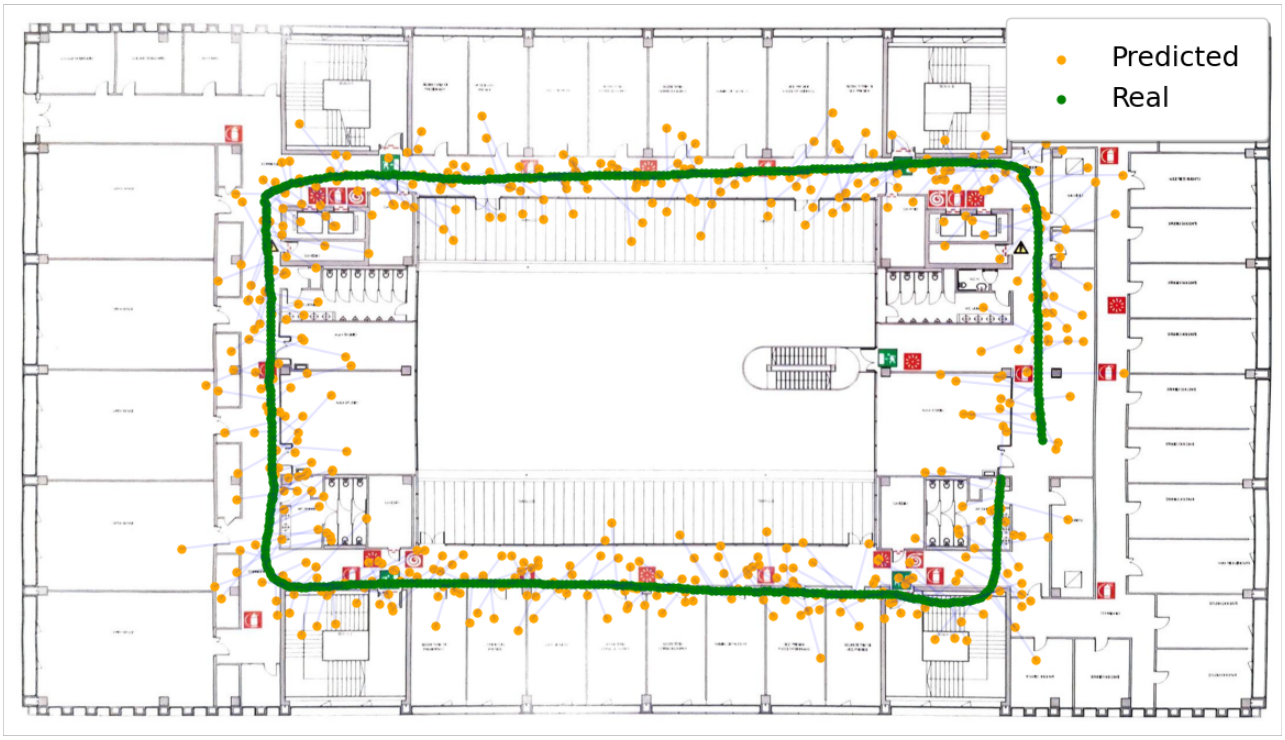


Fig. 8. Predicted trajectory MapNet

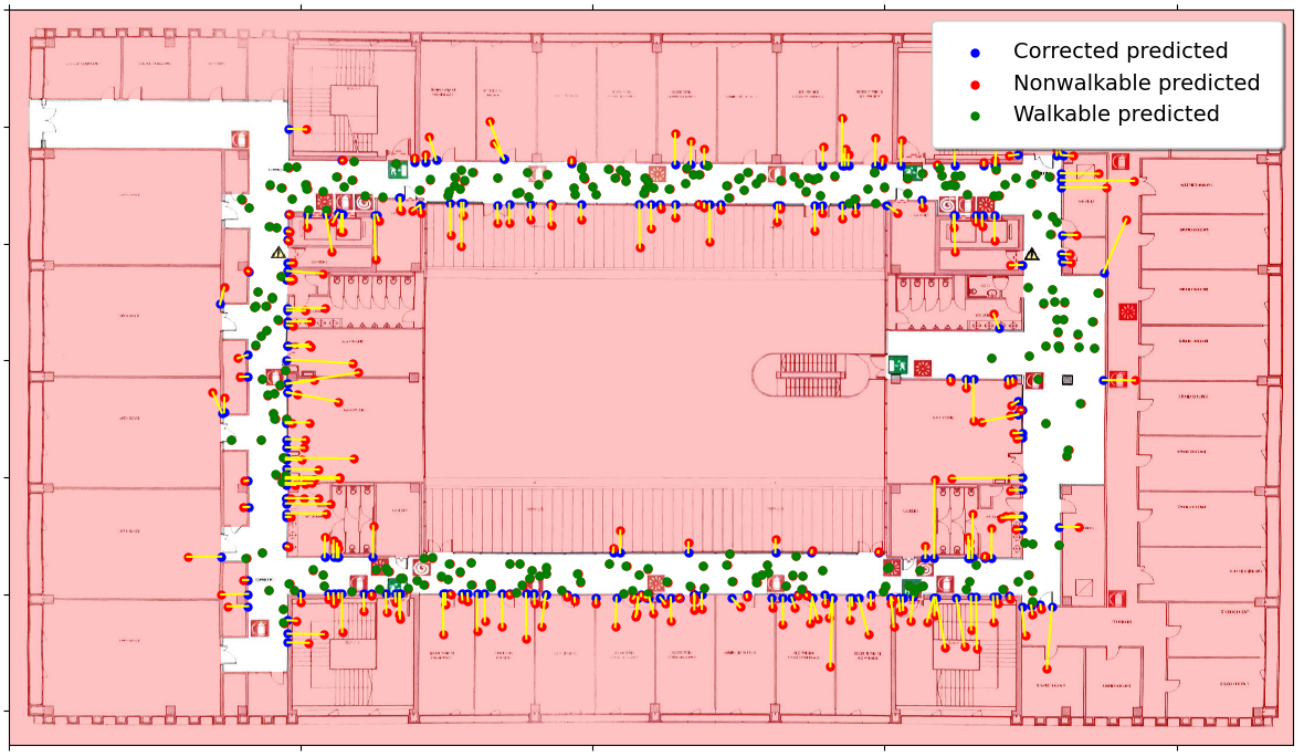


Fig. 9. Predictions post-processed on walkable areas

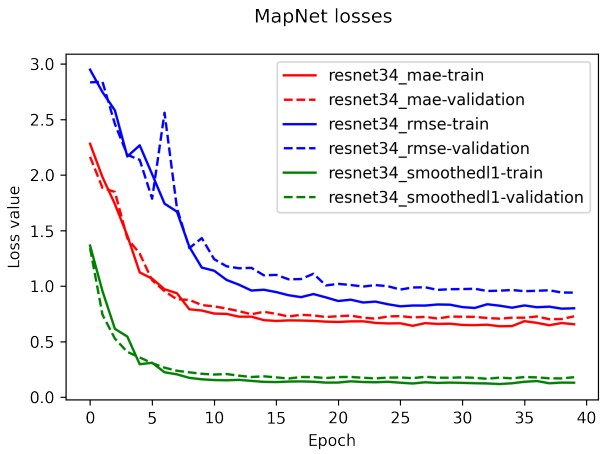


Fig. 10. MapNet (ResNet-34) losses

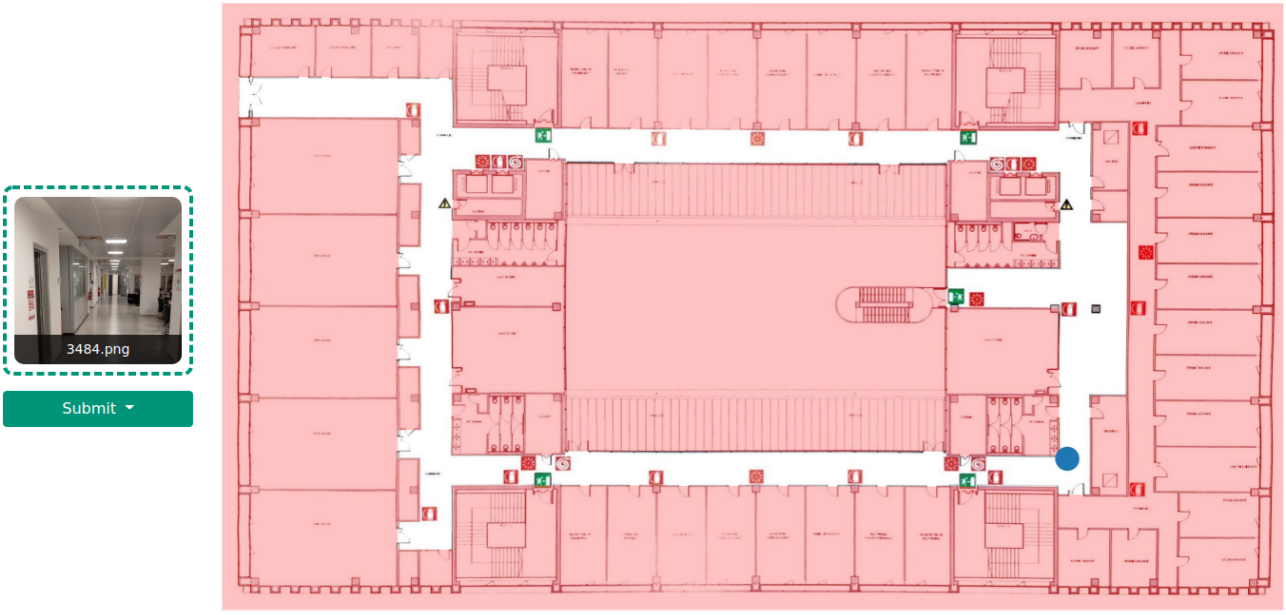


Fig. 11. Inference dashboard