A Deep Learning Approach to Camera Pose Estimation

Federico Izzo
DISI, University of Trento
Trento, Italy
federico.izzo@studenti.unitn.it
229316

Francesco Bozzo
DISI, University of Trento
Trento, Italy
francesco.bozzo@studenti.unitn.it
229312

***Abstract*—The task of camera pose estimation aims to find the position of the camera that captured an image within a given environment. While different geometric approaches have already been studied in the literature, the aim of this project is to analyze 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 labeled 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 deep learning model with PyTorch, and (3) 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. Even if the state-of-the-art (SOTA) deep learning approaches for this field is less accurate than geometric ones, they ensure more generalization capabilities at a reduced computational cost.**

***Index Terms*—camera pose estimation, PoseNet, MapNet, deep learning, computer vision**

I. Introduction

The *camera pose*, known also as *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 for poses: 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. Also, although 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 in this work.

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 system in the 3D model. Formally speaking, the *absolute pose estimation (APE)* problem can be 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:

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

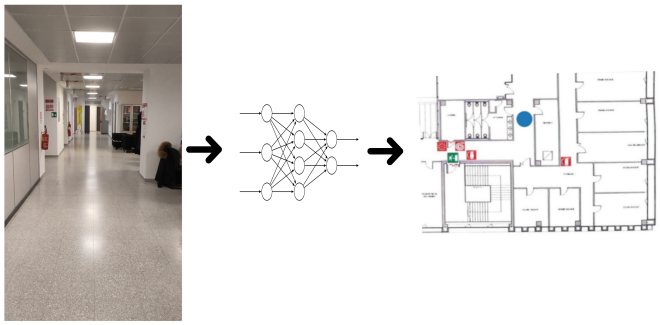


Fig. 1: How an APE deep learning model works.

Apart from APE, another 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 as input two images:

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

where (x_c^{rel}, q_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 which 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 for performing RPE and APE: here we focus on MeNet [1] for the first and PoseNet [2] and MapNet [3] for the latter, since they can achieve good results without requiring huge computational costs.

APE deep learning models rely mostly on *transfer learning*: the idea is to use state-of-the-art pre-trained 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. In this case, the pre-trained network used for the knowledge transfer is GoogLeNet [4], where the softmax classification layer is replaced with a sequence of fully connected layers. Even if the obtained results are decent, the model (1) lacks of generalization when applied to unseen scenes and (2) fails in learning complex environments.

In order to solve these problems, 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, with its related variants MapNet+ and MapNet+PGO [3].

On the other hand, 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, such methods are not considered in this work.

III. Dataset generation

A. Tested approaches

The deep learning approaches applied in this work are *supervised learning* techniques that require a labeled dataset. Many attempts were made in order to generate a high-quality labeled dataset for the deep learning models:

- *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. The process of combining multiple noisy information sources to obtain a good estimation can be achieved through Kalman filters;
- *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 that digitally reconstruct environments from a sequence of images with point clouds.

The main problem encountered with IMU sensors is the high presence of noise during data acquisition: this causes the processed signal to be very dirty, so much that the precision is not acceptable for our intentions. A possible solution should imply exploiting some well calibrated hardware in a controlled environment.

Most of the online-available 3D acquisitions are acquired with *depth sensors* or *LIDAR sensors*. For this reason, although the camera pose estimation would have been very precise, we could not reproduce this acquisition system in the University of Trento. In addition to that, only few datasets are freely available, and they usually lack of good documentation regarding how they were captured.

The motion capture system is able to follow the position of the tracked objects with extreme precision. In this case the main problems are related to the association of poses to video captured from the camera held by the tracked subject and difficulties involved in the system calibration.

The techniques of structure from motion are used to generate 3D models in case many photos are available. The overall idea is to feed the algorithm with these images in order to extract features and build a recomposition of the environment through a point clouds. During the reconstruction process, structure from motion tools also compute camera poses in an arbitrary reference system: this intermediate requirement have been exploited by us to generate a labeled dataset, as presented in Figure 2.

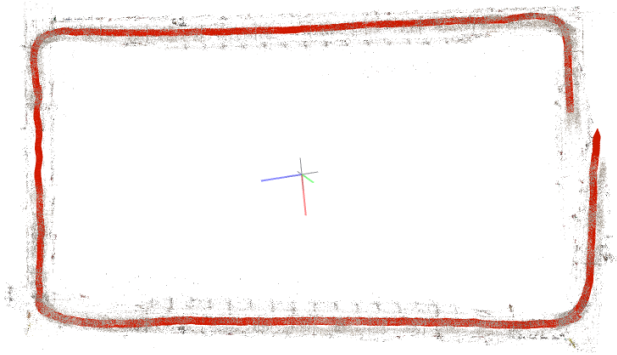


Fig. 2: Trajectory computed by a structure from motion algorithm (COLMAP) in the second floor of the Povo 1 building in the University of Trento. Note that these techniques may not detect successfully closed-loops.

B. Pipeline

In this section we present an automatic pipeline to generate a labeled camera pose estimation dataset using structure from motion techniques. The pipeline requires as input a video captured by any camera, even without sensor calibration. It is composed by several steps:

1) video split: the captured video is split into many frames using `ffmpeg`;
2) structure from motion: frames obtained from the previous step are fed into a structure motion tool that estimates camera poses. In our case we are using `COLMAP` [5];

3) dataset splitting: poses computed in the previous step are split into three subsets: train, validation, and test.

C. COLMAP reconstruction

COLMAP [5] is a tool that allows to build a 3D points clouds reconstruction model of an environment by using photos of it. Like the other structure from motion techniques, COLMAP also computes camera poses in an arbitrary reference system during the reconstruction process. It is possible to generate point clouds with two different levels of precision: *sparse* and *dense*, and both of them are composed by:

- a set of points P which represent the features extracted from the photos (points in Figure 3);
- a set of poses $\mathcal{V}$ associated with the images used for the reconstruction (red line in Figure 3);
- a *coordinate reference system* (CRS).

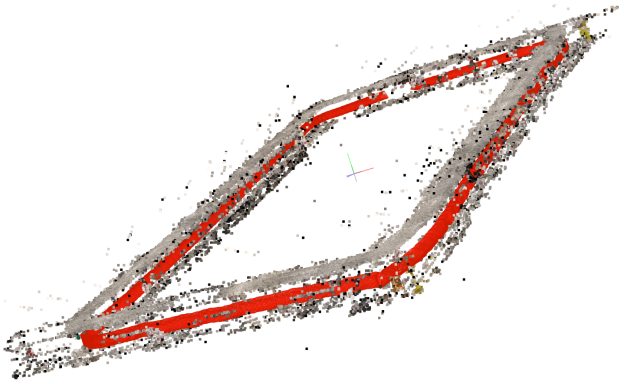


Fig. 3: Features extracted as a points clouds by COLMAP in the second floor of the Povo 1 building in the University of Trento. The red line represents the camera path in the video footage.

The set of poses is represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and for each node i there is an association with a subset of environment features $\mathcal{F}_i : \mathcal{F}_i \subset P$. Edges in $\mathcal{E}$ connect poses from images that use common features.

The sparse reconstruction algorithm first extracts features from each given image and builds the set P, called also as *bag of words*. Once this is done, thanks to a convolution on each image, it creates spacial associations between nodes in the graph comparing images with the bag of words. The convolution enables to map only subsections 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. Once associations are done, the features and the poses are composed in order to create the reconstructed model.

Moreover, the algorithm for the dense reconstruction is very similar to the one used for the sparse one. The main difference is about how the associations are created: in this case the algorithm uses point-wise associations and not convolutions. Even if it requires more computational resources, this approach allows to be more accurate and to create more definite point clouds.

Both algorithms can work on a sequence of unrelated photos of the same environment or a sorted sequence of frames: note that the reconstruction process is way more efficient in the second case.

D. Coordinate reference system alignment

The dataset generated by COLMAP has a *coordinate reference system (CRS)* chosen arbitrarily during the reconstruction. Origin and axes of the COLMAP model may not coincide (actually this is always the case) with the real-world CRS. In order to align them, three steps are required:

1) scale: it is necessary to scale the relative positions of the points with respect to the real-world CRS. This can be achieved measuring the distance between two known points in both COLMAP and real life;
2) translation: align the origins;
3) rotation: align the axes.

The last two steps can be grouped together by using the euclidean or rigid transformation. It involves a rotation R, a translation t and at least three points for both the CRSs that represent the same locations. Equation (5) describes the rigid transformation:

$$\begin{aligned} A &= \{(x_1^A, y_1^A), (x_2^A, y_2^A), (x_3^A, y_3^A)\} \\ B &= \{(x_1^B, y_1^B), (x_2^B, y_2^B), (x_3^B, y_3^B)\} \\ B &= R \times A + t \end{aligned} \tag{5}$$

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

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

To obtain the matrix R, the first step consists in aligning on the same origin the two datasets centroids. This is done by subtracting to each coordinate of each point the centroid of the respective dataset. After this, it is possible to ignore the translation component t and compute the rotation R:

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

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

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

IV. Models

In this work we take in consideration some state-of-the-art deep learning models for camera pose estimation, also with additional small modifications to make them fit better to our use case scenario. In particular, we present:

- MeNet [1] for RPE;
- PoseNet [2] and MapNet [3] for APE.

A. MeNet

The first model we would like to discuss is the MeNet model, that 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 shown in Figure 4, the MeNet is composed by nine deep convolutional layers followed by a sequence of linear layers. While the first part of the network covers the role of feature extraction, the last layers are used to combine the extracted features.

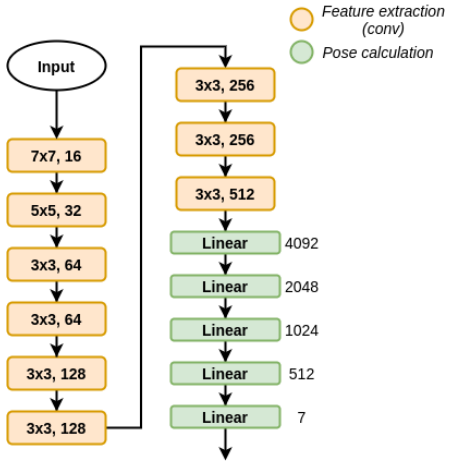


Fig. 4: MeNet model architecture.

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:

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

where P, $\hat{P}$, Q, $\hat{Q}$, and α 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.

B. PoseNet

Since the results given by RPE deep learning solutions are not very promising due to the lack of generalization and to cumulative errors in the estimations, from now on we are going to consider only models strictly developed for APE. In this sense, the first one we present is the PoseNet model [2]. As shown in Figure 5, 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 networks, through

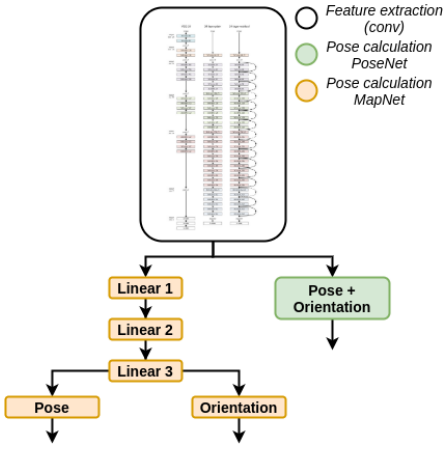


Fig. 5: PoseNet and MapNet model architecture.

the transfer learning approach. In most of the cases, this kind of models are trained on the ImageNet dataset [6], which counts almost 3.2 millions real-world images: this offer good generalization capabilities for the feature extraction task. It is important to notice that these models have been developed to work on the ImageNet dataset, which consists of RGB 224x224 pixels images: for this reason, any PoseNet input must have the same shape. Some examples of state-of-the-art backend models that have been considered are: GoogLeNet [4], ResNet [7], and EfficientNet-B7 [8]. 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	Parameters
GoogLeNet	69.778	89.530	$\sim$ 7,000,000
ResNet-18	69.758	89.078	11,180,103
ResNet-34	73.314	91.420	21,288,263
ResNet-50	76.130	92.862	23,522,375
ResNet-152	78.312	94.046	58,158,151
EfficientNet-B7	**84.122**	**96.908**	63,804,887

Since ImageNet pre-trained models are used for classification purposes, we need to remove the last classification-related layers, and use only the convolutional ones for feature extraction. This gives us the opportunity to insert in the PoseNet a feature extraction mechanism on real-world images with minimum effort: training by scratch such models would require a huge amount of computational power.

Moreover, to train the PoseNet 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, as shown in Figure 5. While even in this case the original model has a single linear layer, we introduce a sequence of them, each followed by the RELU activation

function. This enables the model to learn more complex scenarios fitting better the data distribution.

Another difference with respect to the PoseNet is the loss function used to train the model. In this case, the errors on the prediction of absolute poses are not the only ones which are penalized: in fact, also errors in the relative poses are taken in consideration. To be able to penalize also relative errors, during the training process the model receives as input a batch of `step` (s) sorted frames, that are separated by `skip` (k) frames in the original video. Equation (10) describes the MapNet loss as a mixture of absolute and relative errors regularized by the α hyperparameter:

$$L_{\mathcal{D}}(\Theta) = \sum_{i=1}^{|\mathcal{D}|} h([\hat{P}^i \hat{Q}^i], [P^i Q^i]) + \alpha \sum_{i,j=1, i \neq j}^{|\mathcal{D}|} h([\hat{P}^{ij} \hat{Q}^{ij}], [P^{ij} Q^{ij}]) \tag{10}$$

where $[P^{ij}Q^{ij}]$ is the relative camera pose between pose predictions $[P^iQ^i]$ and $[P^jQ^j]$ for images I^i and I^j. $h(\cdot)$ is a function to measure the distance between the predicted camera pose $\hat{P}$ and the ground truth camera pose P, defined as:

$$h([\hat{P}^i \hat{Q}^i], [P^i Q^i]) = \left\| \hat{P}^i - P^i \right\|_1 e^{-\beta} + \beta + \left\| \hat{Q}^i - Q^i \right\|_1 e^{-\gamma} + \gamma \tag{11}$$

where β and γ are the weights that balance the position loss and rotation loss. Both the parameters can be learned as well during the training procedure. (I^i, I^j) are image pairs within each tuple of s images sampled with a gap of k frames from $\mathcal{D}$.

V. Results

A. Dataset

To test the already discussed data generation pipeline and deep learning models, we mapped the second floor of the Povo 1 building in the University of Trento. We filmed a single vertical video with a smartphone (model: OPPO reno 4 CPH2091) at 2160x3840 pixels 30fps of the entire floor, avoiding moving objects and people. The frames have been extracted at 10fps, resulting in 1462 images for a total of 13.1 GB. COLMAP has been used to generate a sparse reconstruction map using the Poisson mesher, high quality, shared intrinsics, and simple_radial camera model. Due to COLMAP limitations, different runs were necessary, since the final reconstruction was not good enough, especially in terms of closed-loop detection: Figure 2 shows the best result that has been achieved.

Due to the models' backend implementation, input images of the model should be 224x224 pixels: for this reason, resizing and center cropping has been adopted. The final size of the processed dataset is 880.3 MB. Train, validation, and test datasets have equal size and have been generated according the $\mod 3$ operation on the frame index: in this way, adjacent frames are included in different datasets.

B. PoseNet

Since several pre-trained models can be used as the PoseNet backend, in Table IV we present a brief list of them with the respective mean absolute error in the test set: notice that all the PoseNets have been tested with a single final linear layer. Even if the overall trend is similar, it is possible to notice that in general bigger networks obtain better results: this is mainly due to the performance list showed on the ImageNet dataset in Table I. In this case, it means that more parameters make the difference, but causes also drawbacks in model time performances due to higher computational complexity.

TABLE II: PoseNet backend comparison

Model	Position error	Rotation error	Total parameters	Trainable parameters
GoogLeNet	0.781	**0.119**	$\sim$ 7,000,000	3,591
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

In addition, Table III shows different loss functions that have been tested for the Equation (9). Note that in case of $\alpha \neq 1$ and MSE, the training loss is not tractable (goes to infinity).

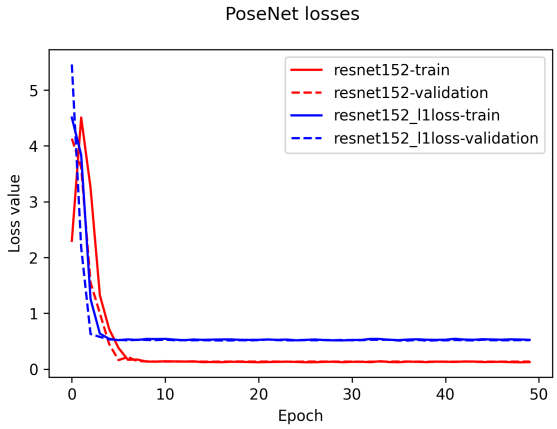


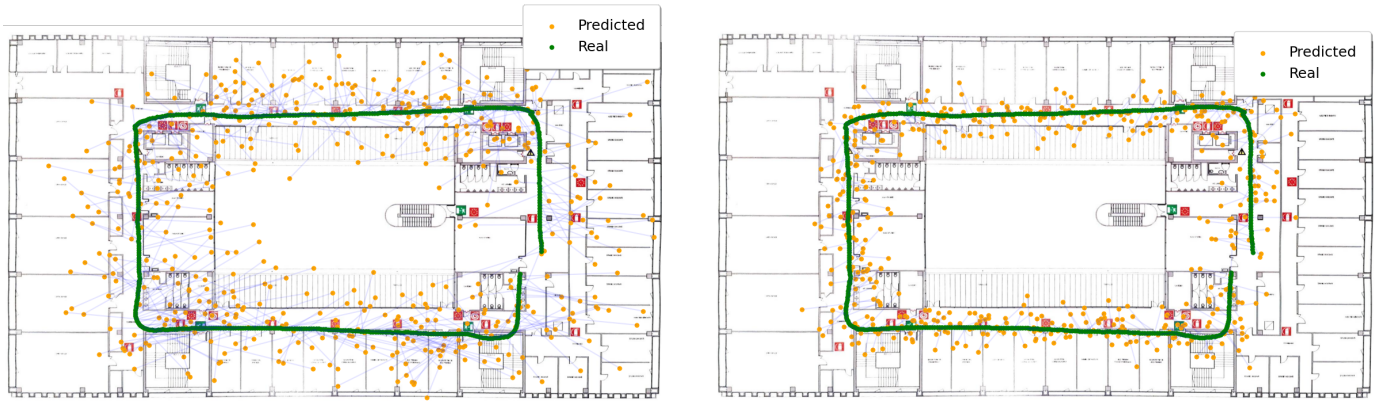
Fig. 6: PoseNet (ResNet-152) losses.

Figure 6 shows the SmoothedL1Loss and L1Loss loss curves for a ResNet-152 PoseNet. It is possible to notice from Table III that in both two cases, the model is converging without overfitting.

TABLE III: PoseNet losses comparison

Loss	Position Error	Rotation Error
SmoothedL1Loss	**0.594**	**0.139**
L1Loss	0.906	0.226
MSE	NaN	NaN
$\alpha \neq 1$	NaN	NaN

Moreover, Figure 7a shows a visual representation of the model performances. It is easy to notice that even if the predictions are almost in the same map zone with respect to



(a) Predicted trajectory with PoseNet (ResNet-152).

(b) Predicted trajectory MapNet (ResNet-34).

Fig. 7: Estimated trajectories.

their ground truth, they are still pretty sparse. The mean error between prediction and target is $\sim 2000cm$.

C. MapNet

To prevent sparsification in the model predictions, we propose to use the MapNet [3]. As for the PoseNet, also in this case several pre-trained models can be used as feature extractor: Table IV shows some candidates, each one used with three final linear layers. Also in this case the overall trend is similar: this highlights that the extracted features are good enough, independently of the used backbone. Another interesting difference with Table II is the fact that models with more parameters are not necessarily the ones performing better.

TABLE IV: MapNet backend comparison

Model	Position error	Rotation error	Total parameters	Trainable parameters
GoogLeNet	0.225	0.0876	$\sim$ 7,000,000	3,677,191
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 interesting point is the importance of the final linear encoder, represented in terms of size with the number of trainable parameters in Tables II and IV. It is clear that more parameters are useful to better fit complex environments, so for this reason three linear layers have been used in our implementation with respect to the original one [3].

Following the example of Figure 6 for the PoseNet model, Figure 8 shows a comparison on the ResNet-34 MapNet for the training and validation curves for different losses. Also in this case the model is converging without overfitting. The loss values can be viewed also on Table V.

Moreover, as can be seen in Figure 7b, the custom MapNet predictions follow more closely the ground truth path with respect to the ones described by the PoseNet in Figure 7a.

TABLE V: Mapnet losses comparison

Loss	Position Error	Rotation Error
L1Loss	0.227	0.042
SmoothedL1Loss	0.187	0.076
RMSE	**0.187**	**0.038**

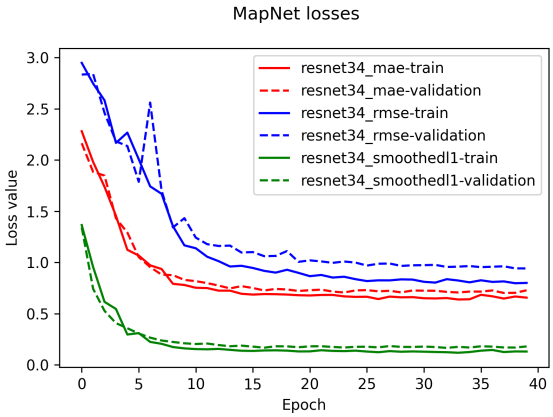
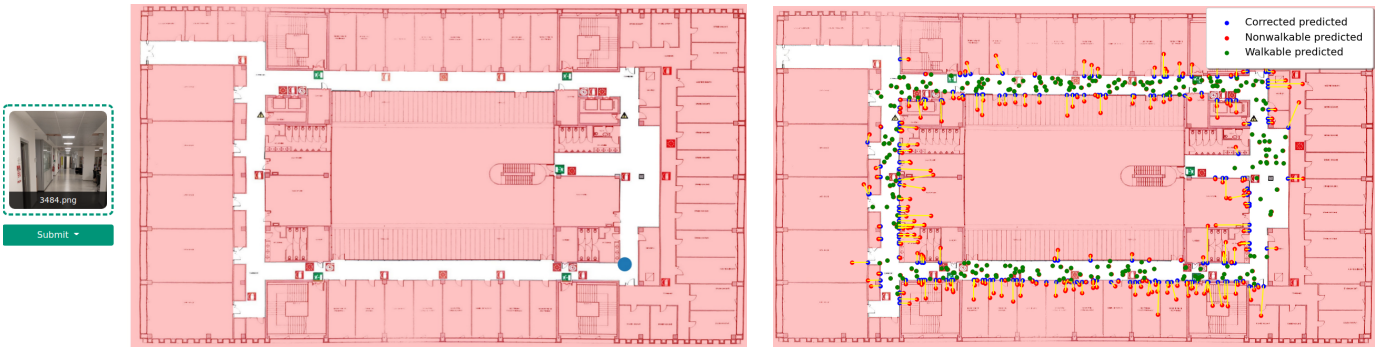


Fig. 8: MapNet (ResNet-34) losses.

Since this approach has been shown to still include some imperfections, we have also included a post-processing algorithm which tries to correct the model predictions. To be more specific, if the model prediction is not in a walkable spot in the map, then the output is going to be the nearest walkable point on the map according to the Euclidean distance criterion. The post-processing procedure allows to reduce the mean absolute error between predictions and targets from $153cm$ to $130cm$: the results can be viewed in Figure 9b.

D. Dashboard

To be able to deploy the model, a web-server has been developed with FastAPI: this interface aims to easily allow users to interact with the final model through a web-based



(a) Inference dashboard.

(b) Predictions post-processed on walkable areas.

Fig. 9: Non-walkable area maps.

Bootstrap dashboard. The dashboard can show the model predictions in three different ways:

- raw model output displayed through an alert;
- raw model output shown in the floor map;
- post-processed model output in a walkable zone in the floor map.

Figure 9a presents the *UI* for showing post-precessed model predictions: red zones are non-walkable areas.

VI. Materials

Every material used in the project have been uploaded. It is possible to find:

- the datasets on the Google Drive folder;
- the code on the GitHub repository.

The project has been developed in Python 3, using common data science libraries, such as numpy, pandas, PyTorch, matplotlib, scipy, aim 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).
- `docs/` contains the project report and presentation.
- `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;
- `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 frame. These files can be generated with the `video_to_dataset.sh` script.

For more information please refer to the GitHub repository.

VII. Conclusions

To summarize, the final results presented in this work are:

- the exploration of multiple dataset generation techniques;
- the COLMAP reconstruction of Povo 1 second floor;
- the development of relative and absolute pose estimation models;
- the fine-tuning of absolute pose estimation models;
- the post-processing of the model outputs;
- the model deployment using a FastAPI web-server.

But, even if the proposed results are promising, there are still many things to consider. Many of the future development of this work should concentrate on:

- trying MapNet+, which is able to learn also from unlabeled data through a semi-supervised approach;
- trying MapNet+PGO, which performs pose graph optimization (PGO) during inference time;
- modifying the model loss to penalize predictions which are in a non-walkable area;
- testing the models in an outdoor environment, with different weather conditions;
- automatizing the coordinate reference system alignment procedure;
- build a more robust indoor dataset to validate the full potential of the proposed deep learning approach;
- deploy the model to the end-user device to ensure privacy and better model performance.

REFERENCES

[1] X. Ruan, F. Wang, and J. Huang, "Relative pose estimation of visual slam based on convolutional neural networks," in *2019 Chinese Control Conference (CCC)*. IEEE, 2019, pp. 8827–8832.

[2] 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.

[3] 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

[4] 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

[5] 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

[6] 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.

[7] 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

[8] 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