Estimating 3D Camera Pose from 2D Pedestrian Trajectories

Yan Xu
Carnegie Mellon University
yxu2@andrew.cmu.edu

Vivek Roy
Carnegie Mellon University
vroy@andrew.cmu.edu

Kris Kitani
Carnegie Mellon University
kkitani@cs.cmu.edu

Abstract

We consider the task of re-calibrating the 3D pose of a static surveillance camera, whose pose may change due to external forces, such as birds, wind, falling objects or earthquakes. Conventionally, camera pose estimation can be solved with a PnP (Perspective-n-Point) method using 2D-to-3D feature correspondences, when 3D points are known. However, 3D point annotations are not always available or practical to obtain in real-world applications. We propose an alternative strategy for extracting 3D information to solve for camera pose by using pedestrian trajectories. We observe that 2D pedestrian trajectories indirectly contain useful 3D information that can be used for inferring camera pose. To leverage this information, we propose a data-driven approach by training a neural network (NN) regressor to model a direct mapping from 2D pedestrian trajectories projected on the image plane to 3D camera pose. We demonstrate that our regressor trained only on synthetic data can be directly applied to real data, thus eliminating the need to label any real data. We evaluate our method across six different scenes from the Town Centre Street and DUKEMTMC datasets. Our method achieves an improvement of $\sim 50\%$ on both position and orientation prediction accuracy when compared to other SOTA methods.

1. Introduction

Our task is to re-evaluate the 3D pose of a stationary single-view camera continually. The camera pose changes over time because of external forces, including winds, earthquakes, and other factors. For instance, a bird jumping on a calibrated security camera could cause a change in the looking angle of this camera. Thus, the camera pose needs to be re-estimated. PnP [15] methods seem to be a good solution to this problem except that they require 2D-to-3D correspondences. Annotating 3D points is expensive, especially when continuous camera pose estimation is required. Therefore in this task, we assume that the 3D point annotations are unavailable and solve the problem with only 2D information inputs. We find that 2D pedestrian trajectories contain useful information for estimating 3D camera pose.

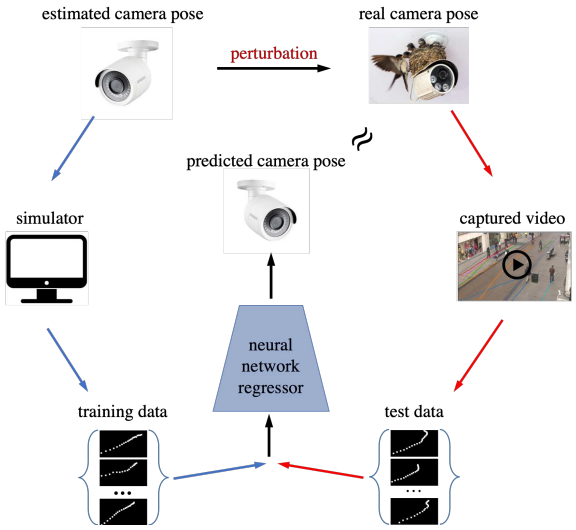


Figure 1: The structure of our proposed method. The training pipeline is connected with blue arrows, the test pipeline is connected with red arrows. The training data is synthetic data, the test data is extracted from real videos.

According to the study from Carey [10] on 815 pedestrians across ages, genders, fitness *etc*., pedestrians walk in a roughly constant speed of about $1.4m/s$. This statistical information can be leveraged for estimating camera pose. If we assume that a pedestrian is walking at a strictly constant speed, the points on the 3D trajectory of this pedestrian should be equally spaced which can serve as a geometric constraint for solving camera pose. Formally, we define the real camera pose as $\mathcal{P}^*$, an estimated camera pose as $\tilde{\mathcal{P}}$, and the pedestrian walking speed as $\boldsymbol{V}$. We use $\tilde{\mathcal{P}}$ to project the observed 2D pedestrian trajectory $\boldsymbol{\mathcal{T}}$ to the 3D ground. As a result, when $\tilde{\mathcal{P}}$ is a close to $\mathcal{P}^*$, points on the projected 3D trajectory should be equally spaced by $\boldsymbol{V}$. Thus, we can build a mapping from a 2D pedestrian trajectory to the 3D camera pose: $\mathcal{P}^* = \boldsymbol{f}(\boldsymbol{\mathcal{T}}; \boldsymbol{V})$. However, the assumption that the pedestrian walks at a constant speed of $\boldsymbol{V}$ is too strong. Instead, the walking speed of a pedestrian is unknown and unlikely to be constant over time. Nevertheless,

Carey [10] shows that the statistical average of pedestrian speeds is generally constant ($\sim 1.4m/s$) over time. We thus propose a method to leverage the statistical information and solve the problem of camera pose estimation in a data-driven manner, by training a neural network regressor to learn an end-to-end mapping from a 2D trajectory to the 3D camera pose.

The structure of our proposed approach is presented in Figure 1. The input of our network is a 2D trajectory, consisting of points whose 2D coordinates represent the pedestrian position in the image. The output of our network is the 3D camera pose, including position and orientation. As shown in Figure 1, our network is trained only with synthetic data. For most data-driven methods, training data collection is costly. We thus bypass this process by training our network with synthetic pedestrian trajectories. Specifically, given a rough estimate of the camera pose and a human motion model, we can generate synthetic trajectories for training. First, we sample a set of camera poses $\{\boldsymbol{P_i}\}$ around an estimation $\boldsymbol{P^o}$. Our hypothesis is that the perturbation the real camera pose $\boldsymbol{P^*}$ is close to a camera pose in $\{\boldsymbol{P_i}\}$. In the next step, we generate synthetic trajectories for each sampled camera pose $\boldsymbol{P_i}$ with the human motion model and use the data to train our NN regressor. At test time, we extract real pedestrian trajectories from videos and feed them into the trained regressor. We use the average of the predictions from all test trajectories as the estimated camera pose $\tilde{\boldsymbol{P}} \approx \boldsymbol{P^*}$. To verify the effectiveness, we evaluate our approach across six real camera settings from *Town Centre Street* [3] and *DUKEMTMC* [39] datasets. Compared to baseline methods using an image as input, our approach reduces the prediction error by $\sim 50\%$ for both translation and rotation while exponentially decreasing training time cost. We summarize our contributions as follows.

- We propose an approach to regress the 3D pose of a stationary camera from only 2D pedestrian trajectories. Our approach could serve as a potential alternative solution for camera pose estimation, especially when 3D information is unavailable.
- We demonstrate with experimental results that our NN regressor trained only on synthetic data can be directly applied to real data, therefore saving on the cost of collecting training data and has good generalizability.

2. Related Work

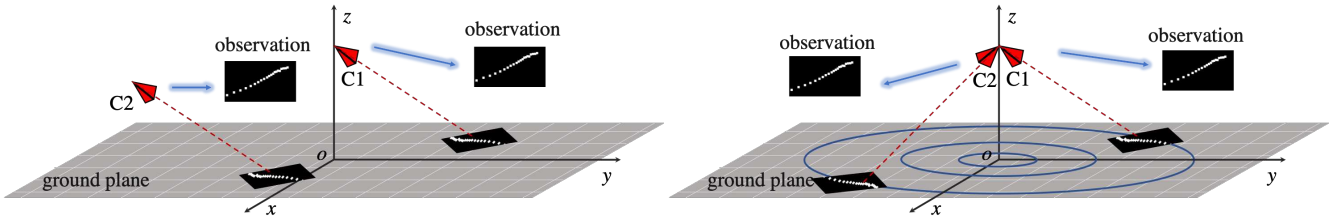
Geometric Methods are the conventional solution for camera pose estimation. When 3D information is available, PnP methods [15] are the preferred choice. These methods [26, 50, 22, 8, 29] solve for the camera pose by minimizing the reprojection error of 2D-to-3D feature correspondences, usually inside a RANSAC [14] loop to eliminate noisy data. When 3D information is not available, methods have been proposed to employ constraints from geometric relationships, such as parallelism and orthogonality [23, 46, 49, 47, 9, 13], for estimating camera pose. Geometric methods are usually accurate. However, 2D-to-3D correspondences are not readily available in real applications, and geometric shapes are usually missing in many scenes. To deal with this, we attempt to use a ubiquitous existing 2D information, pedestrian trajectories, to solve camera pose.

Learning-based Methods for camera pose estimation have shown encouraging results in recent years. One class of learning-based methods regress camera pose end-to-end from an image or a video clip. A representation of these methods is PoseNet [20], which regresses the 6-DoF camera pose directly from a monocular image. Since PoseNet, many methods have been proposed. These methods either utilize different network architectures [19, 48, 35, 33] or leverage temporal information from the input [45, 12] or introduce geometric constraints [19, 25, 6] to improve performance. The second class of learning-based methods put a learnable neural network module into a structural pipeline, establishing correspondences between 2D pixels and 3D points, then solve camera pose by solving the RANSAC PnP problem. These methods include DSAC/DSAC++ [4, 5], BTBRF [34], InLoc [43] *etc*. Some other works treat camera pose estimation as a multi-task learning [37, 27] or a metric learning [2] problem. Generally, all these methods do not require annotated 2D-to-3D correspondences. However, training data for these methods is still expensive to collect. To avoid such costs, Chen *et al*. [11] propose to use synthetic training images for Sports Camera Calibration. We leverage a similar idea but use low-dimensional synthetic pedestrian trajectories as training data instead of high-dimensional images.

Camera Pose Estimation Using Pedestrians have been well investigated in the last decade. Some methods [38, 1] simultaneously solve tracking and camera pose estimation using pedestrian trajectories by assuming a constant velocity of the moving object. Other methods [31, 32, 24, 18, 28] leverage the prior knowledge about the distribution of relative human heights to perform camera pose estimation. A third class of methods [16, 44, 17, 7, 31] make use of head and foot location information to calculate the vanishing points and line to further estimate camera pose. Our work also leverages the geometric information in pedestrian trajectory to estimate the 3D camera pose. On contrary to these methods, we chose to solve the problem in a learning-based manner. Our intuition is to leverage the good expressive ability of neural networks to learn a mapping from input to camera pose while improving noise-robustness.

3. Approach

As mentioned in Section 1, the 3D pose of a static camera can be estimated from 2D pedestrian trajectories, under



(a) Position ambiguity.

(b) Orientation ambiguity.

Figure 2: Location and orientation ambiguities when estimating 3D camera pose with only pedestrian trajectories. In each subfigure, *C1* is the real camera pose, *C2* represents an alternative fake camera pose that has the same observation as *C1*.

the assumption that the pedestrian is moving at a constant speed. Formally, consider a pedestrian moving with a constant speed $\boldsymbol{V}$ on the ground plane $\boldsymbol{\pi_g}$ being observed by a camera. We aim to build a mapping from a 2D projection of the trajectory $\boldsymbol{\mathcal{T}}$ to the 3D camera pose $\boldsymbol{\mathcal{P}}$,using a neural network approximation function $\boldsymbol{f(\cdot;\theta)}$, where $\boldsymbol{\theta}$ are learnable network parameters:

$$\boldsymbol{\mathcal{P} = f(\mathcal{T}(V);\theta)} \tag{1}$$

In this section, we first make necessary explanations on the definition of camera pose $\boldsymbol{\mathcal{P}}$ and the evaluation metrics. We then describe the proposed neural network architecture and give the loss which we use to supervise the learning. Finally, we explain synthetic training data generation.

3.1. Evaluation Metrics

The 3D pose of a camera $\boldsymbol{C}$ is conventionally defined as the location and orientation of the camera in a world coordinate system. The location of $\boldsymbol{C}$ is often specified by a vector $\boldsymbol{t} \in \mathbb{R}^3$ with respect to the world origin. The orientation of $\boldsymbol{C}$ can be described with several representations, such as a rotation matrix, a set of Euler angles, or a quaternion. Among these representations, the quaternion $\boldsymbol{q} \in \mathbb{R}^4$ is most commonly used to describe the camera orientation. A tuple then represents the overall 3D camera pose:

$$\boldsymbol{\mathcal{P} = (t, q)} \tag{2}$$

Most deep pose regressors directly predict the camera pose $\boldsymbol{\mathcal{P}}$ as a 7-dimensional vector. However, if the pedestrian trajectory is the only information input for estimating the camera pose, then there will be ambiguities in the predicted camera pose, as shown in Figure 2. From Figure 2, we observe that the 2D trajectory can only decide 1-dimensional location (height) and 2-dimensional orientation (pitch and roll angle) of the camera. In the implementation, we focus on the relative position between the camera and the scene and only predict the height of the camera $\boldsymbol{Z_c}$: $\boldsymbol{t = [Z_c]}$. For the estimation of camera orientation, we still predict the quaternion $\mathbf{q}$, but we fix the yaw angle at training and test. (Note that: Other deep pose regressors also have the ambiguity problem. However, since they use images as input and it is unlikely to get the same images at different places, so the ambiguity problem does not affect them much. However, as a cost, lighting and seasonal changes will have dramatic impacts on those deep pose regressors.)

The performance of a pose estimator is estimated by the location prediction error $\boldsymbol{t_{err}}$ and orientation prediction error $\boldsymbol{r_{err}}$. $\boldsymbol{t_{err}}$ is defined as the Euclidean distance between the real location $\boldsymbol{t^*}$ and the predicted location $\boldsymbol{t}$:

$$\boldsymbol{t_{err} = \|t^* - t\|_2} \tag{3}$$

$\boldsymbol{r_{err}}$ is measured by the angle between the real orientation $\boldsymbol{q^*}$ and predicted orientation $\boldsymbol{q}$ [42]:

$$\boldsymbol{r_{err} = 2\cos^{-1}(|\langle q^*, q\rangle|)} \tag{4}$$

3.2. Network Architecture

The architecture of our proposed NN regressor is presented in Figure 3. The input of our network is a 2D pedestrian trajectory $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{2\times N}$, where $\boldsymbol{N}$ is the length of the trajectory, and 2 is the dimension of the pixel coordinate of each point on the image plane. The output of our network is the camera pose $\boldsymbol{\mathcal{P}}$ as given in Equation 2.

Considering the input of our network is a trajectory with variable length, we first use an RNN Feature Encoder (FE) to encode the input trajectory $\boldsymbol{\mathcal{T}}$ to feature $\boldsymbol{u}$ with a fixed dimension. After the FE module, we concatenate a Joint-feature Extractor (JE) to learn a common feature $\boldsymbol{v}$ that helps predict both location and orientation. Finally, the camera location and orientation are separately predicted from $\boldsymbol{v}$ with a Location Branch (LB) and an Orientation Branch (OB).

FE is a bi-directional LSTM [41] with a 64-dimensional hidden layer whose input is $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{2\times N}$. It has two hidden layers, and connects the output of the two hidden layers of opposite directions as a final output $\boldsymbol{u} \in \mathbb{R}^{128}$. In the implementation, we found that bi-directional LSTM significantly outperforms single-directional LSTM in terms of the prediction accuracy of camera pose.

JE is a multi-layer perceptron (MLP) consisting of three fully connected layers. The sizes of the three layers are

128×256, 256×1024, 1024×512 respectively. The input is the feature $\boldsymbol{u} \in \mathbb{R}^{128}$ from FE, and the output is the joint-feature $\boldsymbol{v} \in \mathbb{R}^{512}$.

LB and OB are two branches taking $\boldsymbol{v}$ from JE as input. LB is a 3-layer MLP with the sizes 512×256, 256×128, 128×1. OB is another 3-layer MLP with the sizes 512×256, 256×128, 128×4. The output of LB is camera location $\boldsymbol{t} \in \mathbb{R}^1$, and the output of OB is a quaternion $\boldsymbol{q} \in \mathbb{R}^4$ representing camera orientation. We found that separately predicting $\boldsymbol{t}$ and $\boldsymbol{q}$ leads to higher prediction accuracy than predicting them together with one fully-connected layer. Our intuition is that since $\boldsymbol{t}$ and $\boldsymbol{q}$ are measured with different scales (*meter* vs *radian*), it is better to predict them with separate branches.

All the fully-connected layers except for the last layers of LB and OB are concatenated with a non-linear layer. The activation function is ReLU.

3.3. Loss

The regression function $\boldsymbol{f}(\cdot;\boldsymbol{\theta})$ from a trajectory $\mathcal{T}$ to the camera pose $\mathcal{P}$ is presented in Equation 1. We aim to solve for an optimal $\boldsymbol{\theta} = \boldsymbol{\theta}^*$ such that the difference between the predicted camera pose $\mathcal{P}$ and the real camera pose $\mathcal{P}^*$ is minimized. The loss function for supervising training is given in Equation 5. It consists of two parts: the Location Loss (L Loss) and the Orientation Loss (O Loss).

$$J(\mathcal{T}) = \underbrace{\|t^* - t\|_2}_{Location Loss} + \alpha \cdot \underbrace{\left\| q* - \frac{q}{\|q\|_2} \right\|_2}_{Orientation Loss} \tag{5}$$

in which, $\boldsymbol{t}^*$ and $\boldsymbol{q}^*$ represent the real camera pose, and $\boldsymbol{\alpha}$ is an adjustable scale factor used to balance the prediction accuracies of location and orientation. The first term describes the location loss, which is the Euclidean distance between the predicted location $\boldsymbol{t}$ and the real location $\boldsymbol{t}^*$. The second term is the orientation loss, measuring the difference between the predicted quaternion $\boldsymbol{q}$ and the real quaternion $\boldsymbol{q}^*$. Euclidean distance is used to approximate the spherical distance for the orientation loss, which has been proved effective by Kendall *et al*. [20].

The parameter $\boldsymbol{\alpha}$ in PoseNet needs to be carefully tuned for different scenes. However, we found that the value of $\boldsymbol{\alpha}$ does not have much impact on the performance of our regressor. We've trained networks across different camera pose settings using $\boldsymbol{\alpha} \in [\mathbf{1}, \mathbf{10000}]$, and the difference in the prediction accuracies is negligible. We guess it is because our input is a 2D trajectory, which contains less redundant information than a colorful image. It is easier for the network to learn the difference between location and orientation, even without tuning $\boldsymbol{\alpha}$. Therefore, we set $\boldsymbol{\alpha} = \mathbf{1}$ in all our experiments.

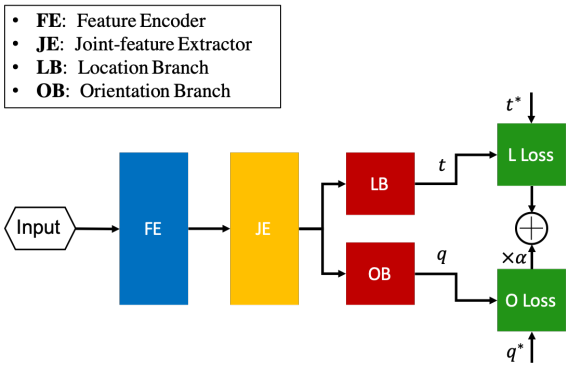


Figure 3: Architecture of our proposed NN regressor.

3.4. Synthetic Training Data

To save the cost of training data collection and annotation, we propose to train our NN regressor with synthetic data and directly apply to real data. To generate synthetic training data, we consider the following scenario: 1) A non-accurate estimation of the camera pose is available. As an example, the estimation can be from past measurements. 2) The intrinsic parameters are known, which is reasonable since the intrinsic parameters will not change once the camera is produced. In such a scenario, we can generate synthetic data for training.

Training with Synthetic Data When a non-accurate estimation of the camera pose is available, we can hypothesize that the real camera pose is in a reasonably close range to the estimated one. Let the estimated camera pose be $\mathcal{P}^o$:

$$\begin{aligned} \mathcal{P}^o &= (t^o, \theta^o) \\ t^o &= (t^o_x, t^o_y, t^o_z) \\ \theta^o &= (\theta^o_x, \theta^o_y, \theta^o_z) \end{aligned} \tag{6}$$

in which $\boldsymbol{t}^o$ is the estimated location, and $\boldsymbol{\theta}^o$ is the estimated orientation. We then sample a set of camera poses $\{\mathcal{P}_i\}$ around $\mathcal{P}^o$. In implementation, we use the ground truth camera pose given in the dataset as $\mathcal{P}^o$, around which we sample $\{\mathcal{P}_i\}$ and make sure $\mathcal{P}^o \notin \{\mathcal{P}_i\}$. In real application, $\mathcal{P}^o$ can be obtained from a past measurement.

For sampling location, we keep $[t^o_x, t^o_y]$ fixed and sample camera height from $[\mathbf{min}\,(\mathbf{0}, t^o_z - \mathbf{3}m), t^o_z + \mathbf{3}m]$. For sampling orientation, since we use the x-conventional Euler angles in the experiments, $\boldsymbol{\theta}^o_x$ represents the yaw angle. We keep $\boldsymbol{\theta}^o_x$ fixed and sample other two angles from $[\tilde{\theta} - \mathbf{15}°, \tilde{\theta} + \mathbf{15}°]$, in which $\tilde{\theta} \in \{\tilde{\theta}^o_y, \tilde{\theta}^o_z\}$. We make uniform samplings for both location and orientation and set the sampling steps to be $\mathbf{0.4}m$ and $\mathbf{2}°$ respectively.

Once we have $\{\mathcal{P}_i\}$, we can generate synthetic trajectories. Considering that pedestrians generally walks with an constant average speed $V \approx \mathbf{1.4}m/s$ [10], we assume V

follows a Gaussian distribution:

$$V \sim \mathcal{N}(\bar{V}, \sigma^2) \tag{7}$$

in which $\bar{V} = 1.4m/s$ and $\sigma = 0.1m/s$. For each camera pose $\mathcal{P}_i$, we randomly sample 10 pedestrian speeds $\{V_j\}, j \in [1, 10], j \in \mathbb{R}^{\mathbb{N}}$. By doing this, we can guarantee that $p(V_j \in [\bar{V} - 3\sigma, \bar{V} + 3\sigma]) = 99.74\%$. Namely, the sampled pedestrian speeds will fall into the interval of $[1.1m/s, 1.7m/s]$ with a probability of 99.74%, which is reasonable in real-life scenarios. For each combination of camera pose and pedestrian speed $\{\mathcal{P}_i, V_j\}$, we generate 1 synthetic 2D trajectory $\mathcal{T}_{ij}$ with the camera projection model and a human motion model:

$$\mathcal{T}_{ij} = g(\mathcal{P}_i, V_j) \tag{8}$$

in which $g(\cdot)$ is the trajectory generation function, and $\mathcal{T}_{ij} \in \mathbb{R}^{2\times N}$, $N \in [11, 31]$ is the length of $\mathcal{T}_{ij}$.

At training, we generate a synthetic dataset $\{\mathcal{T}_{ij}; \mathcal{P}_i\}$ to train our NN regressor. Figure 4 presents a comparison between synthetic pedestrian trajectories and real pedestrian trajectories. We can see that the synthetic data are almost indistinguishable from the real data.

Test on Real Data At test time, we extract pedestrian trajectories $\{\mathcal{T}_k\}, k \in [1, K]$, from videos captured by the camera, feed each $\mathcal{T}_k$ to a trained regressor $f(\cdot; \theta^*)$ to get a camera pose prediction $\tilde{\mathcal{P}}_k$. Finally, we leverage the prior knowledge on the statistical average of pedestrian speeds and take the mean of the predictions from all the trajectories as the final estimation of the real camera pose $\mathcal{P}^*$:

$$\tilde{\mathcal{P}} = \frac{1}{K}\sum_{k=1}^{K}\tilde{\mathcal{P}}_k = \frac{1}{K}\sum_{k=1}^{K} f(\mathcal{T}_k; \theta^*) \approx \mathcal{P}^* \tag{9}$$

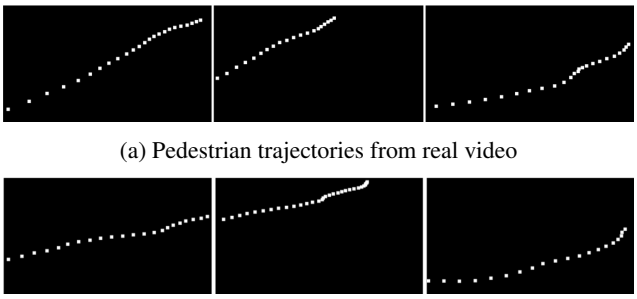
4. Experiments

In this section, we evaluate our approach on real camera settings and present the results and analysis. Section 4.1 introduces the datasets and necessary implementation details. Section 4.2 compares our approach with learning-based baseline methods. Section 4.3 compares our approach with a geometric baseline method. Section 4.4 gives a qualitative result of our approach. Section 4.5 provides the ablation study results.

4.1. Datasets and Training Details

The six real scenes that we used to evaluate our approach are from Town Centre Street [3] and DUKEMTMC [40]. These two datasets contain a large number of pedestrians moving at roughly constant speeds. Figure 5 shows the real pedestrian trajectories we use for testing.

Town Centre Street (TCS) [3] is an outdoor dataset containing a video sequence of a busy street with up to thirty pedestrians visible at a time. The video is 5 minutes long at 25 fps and 1080p resolution. Camera intrinsic and extrinsic parameters and bounding boxes for each pedestrian at each frame are available.



(a) Pedestrian trajectories from real video

(b) Synthetic pedestrian trajectories

Figure 4: Pedestrian trajectories extracted from real videos and synthetic trajectories generated from our simulator.

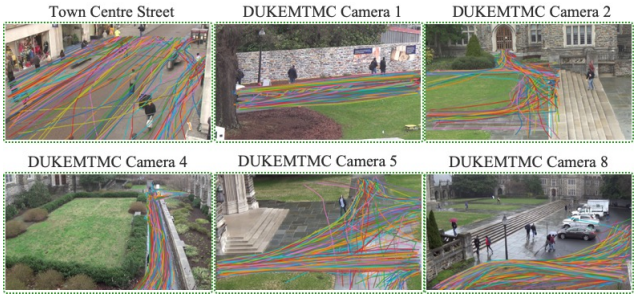


Figure 5: Real pedestrian trajectories of for test. Only the trajectories on the same ground plane are used.

DUKEMTMC (DM) [39] is a manually annotated and calibrated multi-camera dataset recorded outdoors on the Duke University campus with 8 synchronized cameras. It consists of 8 static cameras $\times$ 85 minutes of 1080p 60 fps video for a total of more than 2,000,000 manually annotated frames and more than 2,000 identities. The camera extrinsic and intrinsic parameters are provided. The ground plane coordinates of all the identities are accurately labeled. We pick 5 of the most densely populated scenes from the total of 8 scenes to evaluate our method.

The model is implemented using PyTorch [36] and optimized using ADAM [21] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and an initial learning rate of 0.01. We train on one NVIDIA Titan X (Pascal) GPU with a batch size of 1024 and each round consisting of 50 epochs.

4.2. Compare with Learning-based method

We compare our method with other learning-based pose estimators in this experiment. Current learning-based pose estimators can be generally divided into two classes. The first class of methods regress the camera pose end-to-end from an image or video clip, such as PoseNet [20] and its variants. The second class of methods embed a learn-

Method		TCS	DM1	DM2	DM4	DM5	DM8	Average
PoseNet [20]	$\alpha = 1$	$0.27m, 4.99°$	$0.74m, 5.08°$	$0.56m, 4.92°$	$0.26m, 5.26°$	$0.90m, 4.99°$	$0.35m, 4.45°$	$0.51m, 4.95°$
	10	$0.23m, 5.13°$	$0.42m, 4.89°$	$0.48m, 5.06°$	$0.22m, 5.11°$	$1.03m, 4.71°$	$0.24m, 4.97°$	$0.44m, 4.98°$
	50	$0.29m, 4.96°$	$0.78m, 5.00°$	$0.57m, 5.03°$	$0.23m, 5.22°$	$0.87m, 4.57°$	$0.32m, 5.04°$	$0.51m, 4.82°$
	100	$0.26m, 4.85°$	$0.43m, 4.99°$	$0.53m, 5.14°$	$0.25m, 5.06°$	$1.09m, 3.81°$	$0.29m, 5.05°$	$0.48m, 4.97°$
	500	$0.22m, 4.41°$	$0.51m, 5.01°$	$0.58m, 2.49°$	$0.19m, 4.99°$	$1.18m, 2.02°$	$0.27m, 4.81°$	$0.49m, 3.96°$
	1000	$0.34m, 4.60°$	$0.63m, 4.68°$	$0.71m, 2.82°$	$0.18m, 4.85°$	$0.92m, 1.87°$	$0.32m, 5.20°$	$0.52m, 4.00°$
	2000	$0.26m, 4.62°$	$0.66m, 4.70°$	$0.49m, 2.28°$	$0.33m, 3.76°$	$1.12m, 1.70°$	$0.23m, 4.69°$	$0.52m, 3.63°$
	5000	$0.26m, 4.34°$	$0.96m, 5.10°$	$0.60m, 1.57°$	$0.29m, 3.46°$	$0.86m, 1.86°$	$0.36m, 4.97°$	$0.56m, 3.55°$
Best fine-tuned result		$0.22m, 4.41°$	$0.42m, 4.89°$	$0.49m, 2.28°$	$0.18m, 4.85°$	$0.86m, 1.86°$	$0.23m, 4.69°$	$\mathbf{0.40m, 3.83°}$
Ours		$\mathbf{0.21m, 2.16°}$	$\mathbf{0.31m, 2.02°}$	$\mathbf{0.16m, 1.29°}$	$\mathbf{0.18m, 0.76°}$	$\mathbf{0.22m, 1.88°}$	$\mathbf{0.26m, 3.73°}$	$\mathbf{0.22m, 1.97°}$

Table 1: Location and orientation prediction errors of our approach and the PoseNet [20] baselines with scale factor α of different values. "TCS" denotes Town Center Street, "DMi" denotes DUKEMTMC scene-i. *(The denotations will be the same for all following tables unless explicitly explained.)* Our approach outperforms the best results of all baseline models.

Methods	TCS	DM1	DM2
DSAC++ [5]	$12.99m, 123.97°$	$23.57m, 99.55°$	$44.87m, 118.10°$
Ours	$\mathbf{0.21m, 2.16°}$	$\mathbf{0.31m, 2.02°}$	$\mathbf{0.16m, 1.29°}$
	DM4	DM5	DM8
	$37.24m, 128.01°$	$35.46m, 104.08°$	$38.63m, 116.35°$
	$\mathbf{0.18m, 0.76°}$	$\mathbf{0.22m, 1.88°}$	$\mathbf{0.26m, 3.73°}$

Table 2: Location and orientation prediction errors of DSAC++ [5] and our method on six real camera settings. DSAC++ fails on this task, while our method performs well.

Methods	TCS	DM1	DM2	DM4	DM5	DM8
VP-1	$86.22°$	$70.48°$	$46.11°$	$45.12°$	$32.81°$	$38.10°$
VP-2	$84.28°$	$44.64°$	$55.37°$	$64.04°$	$28.26°$	$41.63°$
VP-3	$89.22°$	$52.68°$	$38.37°$	$36.54°$	$19.65°$	$18.77°$
Ours	$\mathbf{2.16°}$	$\mathbf{2.02°}$	$\mathbf{1.29°}$	$\mathbf{0.76°}$	$\mathbf{1.88°}$	$\mathbf{3.73°}$

Table 3: Orientation prediction errors of the vanishing-point calibration baseline [17] and our approach. Three different sets of orthogonal vanishing points are used, 'VP-i' denotes the i-th set. The output of baseline changes with different sets of VP. Our approach outputs stable accurate results.

able DNN module into a structural pipeline to solve camera pose, such as DSAC/DSAC++ [4, 5]. We use PoseNet and DSAC++ as the learning-based method baselines in this experiment. Note that PoseNet and DSAC++ are used for solving the relocalization tasks, and their input data are images captured from different view angles. However, in our task, the camera view angle does not change over time. The observations at different time are identical except for the positions of the pedestrians. To eliminate the impact of the static background on the prediction of the baseline methods, we set the input of the pose estimators to be a black and white image containing only a single pedestrian trajectory, as shown in Figure 4.

Table 1 gives a quantitative comparison of the predicted camera location error and orientation error between our approach and the PoseNet [20] baselines with fine-tuning the scale factor $\boldsymbol{\alpha}$. We observe that our proposed method almost *doubles the prediction accuracy* on both location and orientation comparing to the best result of the fine-tuned baseline models. Our model does not require fine-tuning $\boldsymbol{\alpha}$, saving the training cost. Results show that 1) 3D camera pose can be estimated from only 2D pedestrian trajectories using learning-based methods and 2) our model trained on synthetic data can be directly applied to real test data.

Table 2 gives the quantitative comparison between DSAC++ [5] and our method. We observe that SOTA DSAC++ fails on this task, while our method achieves low prediction errors. We believe that DSAC++ fails because it relies on an image as input, which in our case is a mostly black image with sparse trajectory locations marked with white. The first module of DSAC++, which is an FCN [30], maps from a 2D image to 3D points. The black areas of different location inside an image or on the same locations of different images would have the same pixel values, zeros, but different depths. However, the FCN is trained to map the pixels with the same zero values to 3D points with different depths, which leads to the collapse of DSAC++.

4.3. Compare with Geometric method

We compare our approach with traditional geometric methods in this section. Geometric methods usually requires 2D-3D feature correspondences or objects with known geometric patterns, neither of which is available in our task. However, Huang *et al.* [17] proposed a method, using a pedestrian trajectory to find the three orthogonal vanishing points (VP) which can be applied for camera orientation (or rotation) estimation. This method also leverage pedestrian trajectory information and can be applied in our

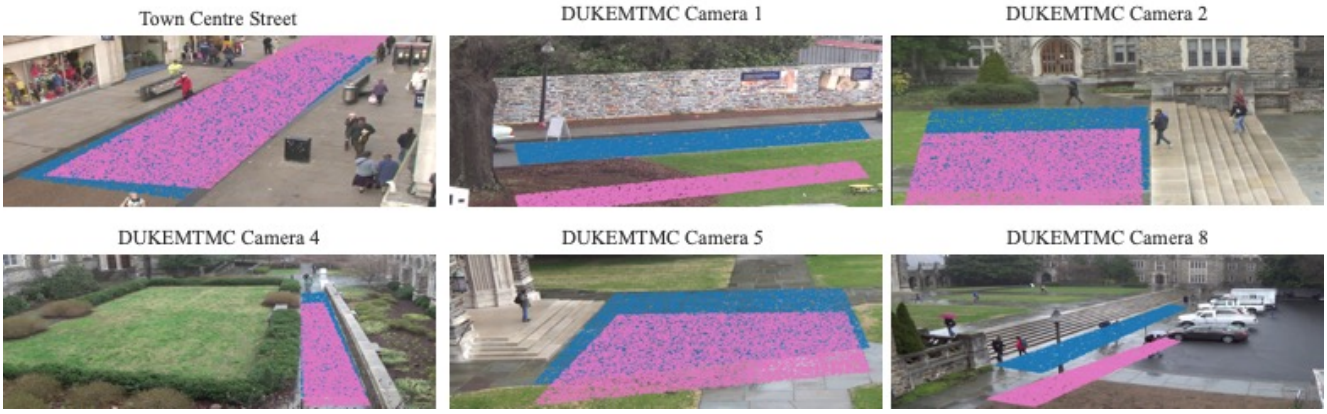


Figure 6: Visualization of the real ground plane and the re-projected ground plane. The blue dotted plane in each image represents the real position of the ground plane, while the pink dotted plane represents the ground plane projected with the camera pose $\tilde{\mathcal{P}}$ predicted by our NN regressor. Our approach works reasonably well across all scenes.

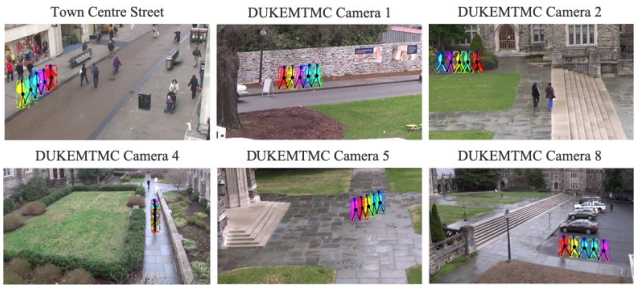


Figure 7: The detected head and feet positions used in the vanishing-point calibration baseline.

experiment scenarios. We compare our approach with the VP calibration method in this experiment.

Table 3 gives the rotation prediction errors of the VP calibration baseline [17] and our approach. One set of orthogonal vanishing points can be detected from each walking pedestrian. We use 3 different identities (or three different sets of orthogonal vanishing points) in this experiment. We observe that the results computed by VP calibration baseline are inaccurate and variate with different sets of vanishing points, while our approach achieves accurate and stable results. Our understanding is that, the VP baseline calculates a rotation between the camera coordinate system and the 3D coordinate defined by the three detected vanishing points. With different sets of vanishing points, the results will be different. On the contrary, our approach estimates the rotation between the camera coordinate system and the world coordinate system, and this rotation is unique and implicitly embedded in the training data. Figure 7 shows the detected head and feet positions used in the VP baseline.

4.4. Qualitative Result

To obtain an intuitive understanding of the camera pose prediction error, we annotate the ground plane in each scene, project it to the 3D world with real camera pose $\mathcal{P}^*$, then re-project it back to the 2D image with our estimated camera pose $\tilde{\mathcal{P}}$. Figure 6 shows the result. We observe that the re-projected ground planes generally overlap well with the real ground planes. Note that, the re-projected ground plane not overlapping well with the ground truth does not means the prediction error is large, the visualization result also impact by the camera looking angle. However, in general, our approach works reasonably well across all scenes.

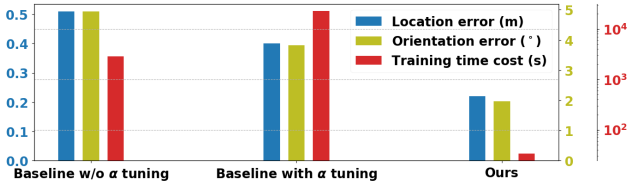


Figure 8: Efficiency comparison between our approach and baselines. The left 3 bars are the result of baseline with $\alpha = 1$, the middle 3 bars are the result of baseline with α fine-tuning, the right 3 bars are the result of our method.

4.5. Ablation Study

4.5.1. Analysis of Efficiency

Figure 8 shows the location and orientation prediction errors vs the training time costs of our approach and the learning-based baselines. For the baseline without fine-tuning α, we set $\alpha = 1$ as what we did for training our model. We observe that, if we do not fine-tune α, the baseline model will have high prediction errors on both location and orientation. However, if we fine-tune α, the training time cost will go up by almost an order. Comparing with the baseline model with fine-tuning α, our model reduces $\mathbf{50\%}$ of both the location and orientation prediction error. Meanwhile, the training time cost of our model is exponentially less even than the baseline model without fine-tuning.

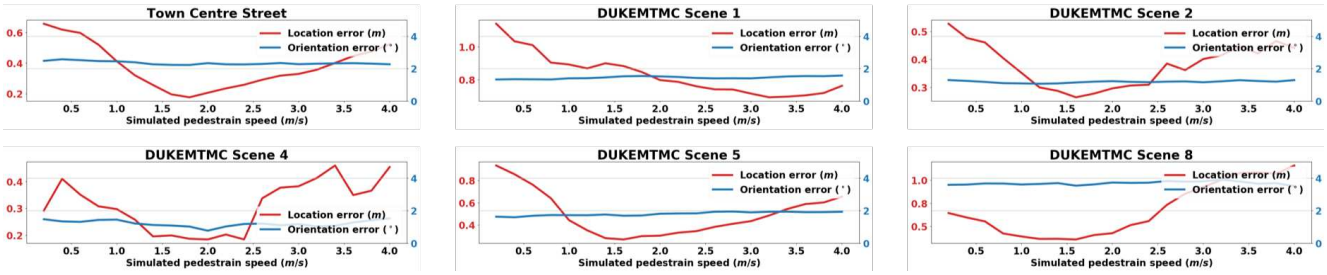
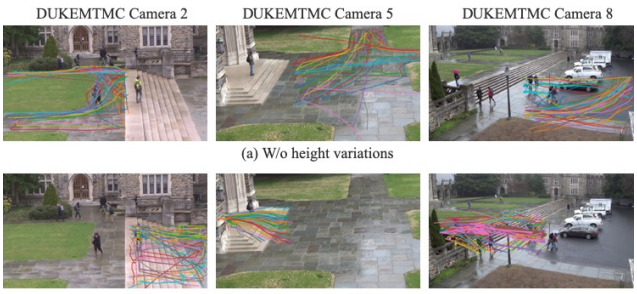


Figure 9: Location and orientation prediction errors of our approach over different synthetic speeds. x-axis measures the synthetic speed, y-axis measures the prediction error. When the synthetic speed is close to human speed, the error is small.



(a) W/o height variations

(b) W height variations

Figure 10: Test trajectories with and without height variations used to test the same trained model.

4.5.2. Analysis of Trajectory Height Variation

For a better understanding of how the height variations in the trajectories impact the performance, we test the same trained model on both trajectories that have height variations and trajectories that do not. We conduct this experiment on DM2, DM5, and DM8, since the stairs in the three scenes will lead to significant trajectory height variations. We generate two test sets for each scene and make sure the two sets have the same size. One set contains trajectories passing through stairs, and another set contains trajectories on the ground plane. The same trained model is tested on both sets. A visualized comparison of the two test datasets is presented in Figure 10, and the result is given in Table 4. We observe that height variation will damage the prediction accuracy, especially the location prediction accuracy. Our explanation is, the stairs will impact the absolute distance of two adjacent points (step length) on the image plane. which will lead to high location (or depth) prediction error.

4.6. Analysis of Synthetic Pedestrian Speed

Since we use synthetic data to train our regressor, the distribution similarity between the synthetic trajectories and the real trajectories will, to a large extent, determine the performance of our approach. To measure the impact of synthetic speed, we conduct the following experiment. For each real test scene, we generate multiple training datasets, with the synthetic speed uniformly sampled from $[\mathbf{0.2}m/s, \mathbf{4.0}m/s]$. The sampling step is $\mathbf{0.2}m/s$. Then for each training dataset, we train a separate regressor and test the trained regressor with the same real test dataset. We conduct this experiment on all the six scenes and present in Figure 9 the result. We have the following observations:

Test Data	DM2	DM5	DM8
w/	$0.22m, 1.31°$	$0.32m, 1.78°$	$0.38m, 3.49°$
w/o	$\mathbf{0.17m}, \mathbf{1.12°}$	$\mathbf{0.21m}, \mathbf{2.07°}$	$\mathbf{0.16m}, \mathbf{2.95°}$

Table 4: Position and orientation prediction errors when trajectories are w/ or w/o height variations. The result shows trajectory height variations impact the performance.

- The synthetic speed has a significant impact on the location prediction accuracy while does not affect the orientation prediction accuracy that much.
- The location error plots for all the six datasets shape as a "bowl" over the synthetic speed. We guess that the synthetic speed, which leads to the smallest location prediction error is the real average pedestrian speed. For 5 out of 6 datasets, the optimal synthetic speed is close to $1.4m/s$, which accords with Carey [10].
- The optimal synthetic speed for DM1 is abnormally around $3.5m/s$, which is almost as fast as the bicycle speed. We guess that the given FPS parameter of DM1 is incorrect. If we eliminate the result for DM1, the prediction errors will become $0.21m$ and $1.96°$.

5. Conclusion

In this paper, we have proposed a learning-based approach to end-to-end regress the 3D camera pose from 2D pedestrian trajectories. Experiments on six real scenes has demonstrated that our approach can achieve high camera pose prediction accuracy across a variety of real-life camera settings. We also verified with experiments that our proposed NN regressor could be trained on synthetic data only and directly applied to real test data.

Acknowledgement: This work was sponsored in part by IARPA (D17PC00340).

References

[1] N. Anjum. Camera localization in distributed networks using trajectory estimation. *Journal of Electrical and Computer Engineering*, 2011:13, 2011. 2

[2] V. Balntas, S. Li, and V. Prisacariu. Relocnet: Continuous metric learning relocalisation using neural nets. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 751–767, 2018. 2

[3] B. Benfold and I. D. Reid. Guiding visual surveillance by tracking human attention. In *BMVC*, volume 2, page 7, 2009. 2, 5

[4] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother. Dsac-differentiable ransac for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6684–6692, 2017. 2, 6

[5] E. Brachmann and C. Rother. Learning less is more-6d camera localization via 3d surface regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4654–4662, 2018. 2, 6

[6] S. Brahmbhatt, J. Gu, K. Kim, J. Hays, and J. Kautz. Geometry-aware learning of maps for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2018. 2

[7] G. M. Brouwers, M. H. Zwemer, R. G. Wijnhoven, et al. Automatic calibration of stationary surveillance cameras in the wild. In *European Conference on Computer Vision*, pages 743–759. Springer, 2016. 2

[8] D. Campbell, L. Petersson, L. Kneip, and H. Li. Globally-optimal inlier set maximisation for simultaneous camera pose and feature correspondence. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–10, 2017. 2

[9] X. Cao and H. Foroosh. Camera calibration using symmetric objects. *IEEE Transactions on Image Processing*, 15(11):3614–3619, 2006. 2

[10] N. Carey. Establishing pedestrian walking speeds. *Portland State University*, 2005. 1, 2, 4, 8

[11] J. Chen and J. J. Little. Sports camera calibration via synthetic data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. 2

[12] R. Clark, S. Wang, A. Markham, N. Trigoni, and H. Wen. Vidloc: A deep spatio-temporal model for 6-dof video-clip relocalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, 2017. 2

[13] C. Colombo, D. Comanducci, and A. Del Bimbo. Camera calibration with two arbitrary coaxial circles. In *European Conference on Computer Vision*, pages 265–276. Springer, 2006. 2

[14] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 2

[15] J. A. Grunert. Das pothenotische problem in erweiterter gestalt nebst bber seine anwendungen in der geodasie. *Grunerts Archiv fur Mathematik und Physik*, pages 238–248, 1841. 1, 2

[16] J. Guan, F. Deboeverie, M. Slembrouck, D. Van Haerenborgh, D. Van Cauwelaert, P. Veelaert, and W. Philips. Extrinsic calibration of camera networks based on pedestrians. *Sensors*, 16(5):654, 2016. 2

[17] S. Huang, X. Ying, J. Rong, Z. Shang, and H. Zha. Camera calibration from periodic motion of a pedestrian. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3025–3033, 2016. 2, 6, 7

[18] I. Junejo and H. Foroosh. Robust auto-calibration from pedestrians. In *null*, page 92. IEEE, 2006. 2

[19] A. Kendall, R. Cipolla, et al. Geometric loss functions for camera pose regression with deep learning. In *Proc. CVPR*, volume 3, page 8, 2017. 2

[20] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015. 2, 4, 5, 6

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[22] L. Kneip, H. Li, and Y. Seo. Upnp: An optimal o (n) solution to the absolute pose problem with universal applicability. In *European Conference on Computer Vision*, pages 127–142. Springer, 2014. 2

[23] N. Krahnstoever and P. R. Mendonca. Bayesian autocalibration for surveillance. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1858–1865. IEEE, 2005. 2

[24] N. Krahnstoever and P. R. Mendonça. Autocalibration from tracks of walking people. In *in Proc. British Machine Vision Conference (BMVC*. Citeseer, 2006. 2

[25] Z. Laskar, I. Melekhov, S. Kalia, and J. Kannala. Camera relocalization by computing pairwise relative poses using convolutional neural network. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 929–938, 2017. 2

[26] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009. 2

[27] Y. Lin, Z. Liu, J. Huang, C. Wang, G. Du, J. Bai, S. Lian, and B. Huang. Deep global-relative networks for end-to-end 6-dof visual localization and odometry. *arXiv preprint arXiv:1812.07869*, 2018. 2

[28] J. Liu, R. T. Collins, and Y. Liu. Surveillance camera autocalibration based on pedestrian height distributions. In *British Machine Vision Conference (BMVC)*, volume 2, 2011. 2

[29] L. Liu, H. Li, and Y. Dai. Efficient global 2d-3d matching for camera localization in a large-scale 3d map. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2372–2381, 2017. 2

[30] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 6

[31] F. Lv, T. Zhao, and R. Nevatia. Self-calibration of a camera from video of a walking human. In *Pattern Recognition,*

2002. Proceedings. 16th International Conference on, volume 1, pages 562–567. IEEE, 2002. 2

[32] F. Lv, T. Zhao, R. Nevatia, et al. Camera calibration from video of a walking human. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (9):1513–1518, 2006. 2

[33] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu. Image-based localization using hourglass networks. *arXiv preprint arXiv:1703.07971*, 2017. 2

[34] L. Meng, J. Chen, F. Tung, J. J. Little, J. Valentin, and C. W. de Silva. Backtracking regression forests for accurate camera relocalization. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6886–6893. IEEE, 2017. 2

[35] T. Naseer and W. Burgard. Deep regression for monocular camera-based 6-dof global localization in outdoor environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1525–1530. IEEE, 2017. 2

[36] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 5

[37] N. Radwan, A. Valada, and W. Burgard. Vlocnet++: Deep multitask learning for semantic visual localization and odometry. *IEEE Robotics and Automation Letters*, 3(4):4407–4414, 2018. 2

[38] A. Rahimi, B. Dunagan, and T. Darrell. Simultaneous calibration and tracking with a network of non-overlapping sensors. In *null*, pages 187–194. IEEE, 2004. 2

[39] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. In *European Conference on Computer Vision workshop on Benchmarking Multi-Target Tracking*, 2016. 2, 5

[40] E. Ristani and C. Tomasi. Tracking multiple people online and in real time. In *Asian Conference on Computer Vision*, pages 444–459. Springer, 2014. 5

[41] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. 3

[42] Y. Shavit and R. Ferens. Introduction to camera pose estimation with deep learning. *arXiv preprint arXiv:1907.05272*, 2019. 3

[43] H. Taira, M. Okutomi, T. Sattler, M. Cimpoi, M. Pollefeys, J. Sivic, T. Pajdla, and A. Torii. Inloc: Indoor visual localization with dense matching and view synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7199–7209, 2018. 2

[44] Z. Tang, Y.-S. Lin, K.-H. Lee, J.-N. Hwang, J.-H. Chuang, and Z. Fang. Camera self-calibration from tracking of moving persons. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 265–270. IEEE, 2016. 2

[45] F. Walch, C. Hazirbas, L. Leal-Taixe, T. Sattler, S. Hilsenbeck, and D. Cremers. Image-based localization using lstms for structured feature correlation. In *Int. Conf. Comput. Vis.(ICCV)*, pages 627–637, 2017. 2

[46] K.-Y. Wong, P. R. S. Mendonca, and R. Cipolla. Camera calibration from surfaces of revolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):147–161, 2003. 2

[47] K.-Y. K. Wong, G. Zhang, and Z. Chen. A stratified approach for camera calibration using spheres. *IEEE Transactions on Image Processing*, 20(2):305–316, 2010. 2

[48] J. Wu, L. Ma, and X. Hu. Delving deeper into convolutional neural networks for camera relocalization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5644–5651. IEEE, 2017. 2

[49] H. Zhang, K. W. Kwan-yee, and G. Zhang. Camera calibration from images of spheres. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):499–502, 2007. 2

[50] Y. Zheng, Y. Kuang, S. Sugimoto, K. Astrom, and M. Okutomi. Revisiting the pnp problem: A fast, general and optimal solution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2344–2351, 2013. 2