Sparse–View Localization via Online Neural 3D Regression

Supplementary Material

This supplementary material has five parts: Appendix A describes the initialization of the last layer's bias and the transformation of the database poses; Appendix B extends ON3R to dense matching using RoMa [16], introduces and benchmarks track-construction strategies, and reports results on ScanNet; Appendix C details the robust loss scaling and adaptive scheduling; Appendix D specifies architecture and training hyperparameters; and Appendix E provides additional visualizations (smoothness examples and 3D scene progression).

A. Initialization

As we train our network from scratch for every new query, it is beneficial to have a good initialization for the training to be fast and generalize across many scenes.

A.1. Initialization the Last Layer's Bias

Without a specific initialization, the model will predict world coordinates around the origin. This can be a poor starting point as it can lead to slow convergence or even degeneracies with 3D points appearing behind the database images. To solve this, the bias of the last layer is initialized with a simple heuristic. We sample S points on the principal axis in front of each camera to get $\{Y_{ij}\}_{i=1,j=1}^{S,K}$. We then form the median of these K rays giving $\{\tilde{Y}_i\}_{i=1}^{S}$ and then project down each point of $\{\tilde{Y}_i\}_{i=1}^{S}$ to each database image giving $\{\tilde{y}_{ij}\}_{i=1,j=1}^{S,K}$. Then, the sample that minimizes $e_i = \sum_{j=1}^{K} M_j \left\|\tilde{y}_{ij} - x_j^s\right\|_2 / M$ is chosen as the bias starting solution (recall that M_j and M denote the number active keypoints per database image and in total, respectively). Here, $x_j^s = \sum_{i=1}^{N} m_{ij} x_{ij} / M_j$ is the average keypoint in each database image. The starting solution is then refined with 100 Adam [24] steps.

We found that the location of the initial bias is not critical, provided it lies in front of most map images with some margin. To evaluate the bias sensitivity, we add Gaussian noise to it (before the steps in Appendix A.2) and measure the pose accuracy (see Tab. 7). Even if $\sigma = 4$ gives about the same accuracy as $\sigma = 0$, the runtime of ON3R becomes roughly 10% longer due to slower convergence.

σ [m]	0	4	8	16
$(\epsilon_R, \epsilon_t) \downarrow$	$(0.91, 0.51)$	$(0.95, 0.51)$	$(0.95, 0.56)$	$(0.98, 0.57)$

Table 7. Median rotation and translation error on Cambridge Landmarks for $K = 2$ with Gaussian noise added to the bias.

A.2. Transforming the Database Poses

Further normalizing the setup, a rigid transformation is first applied to all database cameras such that the coordinate-wise median of all camera centers is placed at the origin. Secondly, we normalize the scene scale to reduce sensitivity to the relative scale between the learned 3D structure and the camera configuration. In particular, the standard deviation of all network weights (and thus the initial 3D point cloud) may be too small or large relative to the distances between the camera centers and the bias point. To address this, we apply a similarity transform to the camera centers by uniformly scaling them around the bias point $b \in \mathbb{R}^3$. Let $s > 0$ denote the scale factor, chosen such that the median camera center (we drop cameras with few matches) attains a fixed distance to b. In homogeneous coordinates, this transformation can be written as

$$H = \begin{bmatrix} sI & (1-s)b \\ 0 & 1 \end{bmatrix}, \tag{9}$$

where I is the 3×3 identity matrix. To maintain consistency under the change of world coordinates, the camera poses are updated by right-multiplying with the inverse of the similarity transform

$$P_i' = P_i H^{-1}, \quad i = 1, \ldots, K \tag{10}$$

where P_i are world-to-camera transformations. The inverse transformation is given by

$$H^{-1} = \begin{bmatrix} \frac{1}{s} I & \left(1 - \frac{1}{s}\right) b \\ 0 & 1 \end{bmatrix}. \tag{11}$$

Carrying out the multiplication $P_i' = P_i H^{-1}$ yields an updated translation component while preserving the rotation (up to a global scale factor). In practice, we keep the rotation matrix fixed and update only the translation as

$$t' = st - (1-s)Rb, \tag{12}$$

which is equivalent up to a global scale of the camera coordinates and therefore preserves all image projections.

B. Dense Matches

In this section, we show that our method can be extended to dense matching methods. Specifically, we discuss how tracks from dense matches can be created and why this step is essential for our method. We use RoMa [16] as our dense matching method and demonstrate its performance on the

indoor dataset ScanNet [9]. This section illustrates the applicability of our method rather than aiming for state-of-the-art results. Therefore, we do not change any configuration details compared to our sparse matcher, which is naturally a suboptimal choice.

B.1. Creating Tracks from Dense Matches

RoMa is a dense image matching method that samples N matches from a dense warp map [16]. RoMa generates matches by warping pixel locations in one image to subpixel locations in another, meaning each correspondence involves one pixel- and one subpixel-level coordinate. As a result, in an asymmetric image setup like ours, not many tracks are usually obtained. Except for some efficiency purposes, this is not critical for our network, since 3D points observed in only one database image do not suffer from depth ambiguity along the projection ray, thanks to the network's spatial smoothness. In other words, if two query keypoints with correspondences in different database images are spatially close, their respective reprojection rays jointly constrain the 3D location, effectively resolving the ambiguity. However, this will not be the case in the bundle adjustment, since the optimization explicitly minimizes reprojection errors per point, and points observed in only one view remain unconstrained along their viewing rays. While each 3D point also has an observation in the query image, this constraint is less reliable because its residual depends on the estimated camera pose.

B.1.1. Greedy Merging of Nearby Keypoints

A naive approach for creating tracks is that if query keypoints are closer than some threshold, τ_q, then matches can be merged, giving a track. From the query keypoints $\{x^q_{ij}\}_{i=1,j=1}^{N,K}$, all pairwise distances can be formed $D^Q \in \mathbb{R}_+^{K\times K\times N\times N}$, where we only care about $D^Q_{ik}, i \neq k$ and note that $D^Q_{ik} = D^Q_{ki}$, and $D^Q_{:,:,j,l} = D^Q_{:,:,l,j}$.

Now, the property of two keypoints being *close* is not *transitive*, meaning that

$$D^Q_{ikjl} \leq \tau_q, D^Q_{klmn} \leq \tau_q \nRightarrow D^Q_{ijmn} \leq \tau_q. \tag{13}$$

Thus, there is some ambiguity when creating tracks: even if $(i,j) - (k,l)$ and $(k,l) - (m,n)$ should be paired, $(i,j) - (m,n)$ may not. Therefore, we solve the problem sequentially with a greedy approach. Given image pair 1 and 2, we do mutual nearest neighbor on the query distances with threshold $\tau_q = 5$px. If there is a match between (i,j) and (k,l) we merge the query keypoints and let the new location be

$$x^q_{ij} = x^q_{kl} := wx^q_{ij} + (1-w)x^q_{kl}, \quad i < k \tag{14}$$

where $w = \frac{\text{current track length}}{\text{current track length}+1}$ (when comparing image pair 1 and 2, the current track length will always be 1). Then, we continue from all the query keypoints already added (from image pairs 1 and 2 and this case) and compare with query keypoints from image pair 3. Then, we continue until $k = K$ is reached.

B.1.2. Including Geometric Consistency

The track construction strategy in Appendix B.1.1 only considers the 2D distance in the query image. However, for merging two tracks, we can also consider the geometric consistency between the keypoints in the reference images. For two matches (i,j) and (k,l) where $i < k$, we can measure this consistency using epipolar geometry based on the known poses of the reference images. The Sampson distance for the reference image keypoints is then

$$\mathcal{E}(x_{ij}, x_{kl}) = \frac{|x_{kl}^T F x_{ij}|}{\sqrt{\sum_{p=1}^2 (Fx_{ij})_p^2 + (F^T x_{kl})_p^2}}. \tag{15}$$

where F is the fundamental matrix computed from the reference poses/intrinsics, and $(Fx)_p$ denotes the kth element of the vector Fx. Similarly to the query keypoint distances D^Q, we can now form the reference image Sampson distances $D^R \in \mathbb{R}_+^{K\times K\times N\times N}$ with these errors.

To include this information, we simply perform mutual nearest neighbor matching based on the average of the query and Sampson distances (analogous to $\frac{D^Q + D^R}{2}$). However, for each query keypoint, the number of corresponding database keypoints equals the length of the track. Thus, we let the Sampson distance be the average distance between the putative match to be added and all database keypoints already in the track.

B.1.3. Benchmarking Dense Track Creation Methods

In the experiments presented in Tab. 8, each track creation method is tested on the MegaDepth 300-sample test set, as in the main paper. Since images have small overlap and to make runtimes faster, $N = 1000$ RoMa-matches are sampled for each image pair. We see that creating longer tracks is slightly advantageous, regardless of whether the greedy method or the Sampson error-based method is used. As a reference, we also include the results of our sparse matcher, LightGlue, used in the main paper.

B.2. Experiments on ScanNet

We test RoMa on the indoor dataset ScanNet using the greedy track creation method using the Sampson errors, and $N = 1000$ sampled correspondences. Exactly the same configuration settings as for our sparse matcher is used. The results are shown in Tab. 9. Among all methods, VGGT performs best, followed by Reloc3r and then ON3R.

This experiment demonstrates that, given dense matches, our pipeline improves upon transitive matching and motion averaging, which have previously been the most natural feature matching-based approaches to this problem. Never-

Method	$K=2$		$K=3$		$K=4$	
	$(\varepsilon_R, \varepsilon_t)\downarrow$	$(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)\uparrow$	$(\varepsilon_R, \varepsilon_t)\downarrow$	$(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)\uparrow$	$(\varepsilon_R, \varepsilon_t)\downarrow$	$(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)\uparrow$
Only two-view tracks	0.26, 0.33	14.3, 40.3, 74.3, 85.0	0.15, 0.26	15.0, 46.7, 88.7, 91.7	**0.14**, **0.21**	**16.3**, 56.7, **92.3**, **96.0**
Greedy merging	0.24, **0.30**	16.0, 41.3, **74.7**, **85.7**	0.15, 0.24	15.7, **51.7**, 89.0, 91.3	**0.14**, **0.21**	16.0, **59.7**, **92.3**, 95.7
↳ with Sampson error	**0.23**, 0.31	**16.3**, **42.0**, 74.0, 84.7	**0.14**, **0.23**	**18.0**, 51.3, **90.0**, **92.7**	0.15, **0.21**	14.7, 59.3, 91.7, 95.7
LightGlue	0.37, 0.45	7.7, 27.3, 71.0, 85.0	0.21, 0.30	9.3, 42.3, 85.3, 94.0	0.15, 0.21	13.3, 55.3, 94.0, 97.7

Table 8. **Results on MegaDepth for** $K = 2, 3, 4$. Metrics are median rotation/translation errors $(\varepsilon_R, \varepsilon_t)$ and recall $(\varepsilon_1$–$\varepsilon_4)$.

theless, feed-forward and pose-regression methods perform better in this indoor setting. To obtain better results with RoMa on ScanNet, the network architecture and configuration settings likely should be adapted to this new scenario. Specifically, the robust threshold used in the Cauchy loss strongly depends on the dataset.

C. Robust Loss Scaling

We use s instead of s^2 in the Cauchy loss (see Eq. (6)) so that the maximum of the derivative becomes 1, independent of s, attained at $r = s$. This follows from the fact that with $\rho(r) = s \ln\left(1 + \frac{r^2}{s^2}\right)$ we get

$$\rho'(r) = \frac{2rs}{r^2 + s^2}, \quad \rho''(r) = \frac{2s(s^2 - r^2)}{(r^2 + s^2)^2}. \tag{16}$$

We find the stationary point of $\rho'(r)$ by setting $\rho''(r) = 0$ and obtain $r^2 = s^2 \iff r = \pm s$ with only $r = s$ being interesting since $r \geq 0, s > 0$. This is a maximum since $\rho''(r) = \frac{2s(s^2-r^2)}{(r^2+s^2)^2} > 0$ for $r < s$ and < 0 for $r > s$. The value at the maximum is indeed 1 since $\rho'(s) = \frac{2s^2}{s^2+s^2} = 1$. This is beneficial if the value of s is changed during training since no unintentional scaling of the gradient step length is introduced.

C.1. Graduated Non-Convexity

The parameter s controls how robust the objective is (as s increases, so does the influence of large errors). Thus, it makes sense, as suggested by Brachmann et al. [3], to have s larger in the beginning when all errors are large and the outliers are hard to spot, and successively decrease it as the training progresses. Since we do not know the network convergence rate a priori, we adapt the Cauchy scale s online from the current reprojection errors. Every 10 epochs, we empirically set:

$$s = \alpha(w \cdot \text{median}\,\{r_k^c\} + (1 - w) \cdot \text{mean}\,\{r_k^c : r_k^c < \tau\}), \tag{17}$$

where $\tau = \frac{\sqrt{H^2+W^2}}{f}$ is the image diagonal in coordinates normalized by the focal length, and $\alpha = w = 0.7$. The training is stopped either when s is small enough ($s < \frac{7}{f}$), when no improvements are made, or when a fixed number of epochs is reached (500 epochs). For the depth loss, we set s to be $s_{\text{depth}} = \max(5s, 50/f)$ only to guide coarsely, whereas it is set to $1/f$ in the bundle adjustment, solely refining the pose on the inlier set.

D. Architecture Details

Here, we start by detailing hyperparameters and network details in ON3R. For a full enumeration of all choices, we refer to the code. We use the same hyperparameters for all datasets and tuple lengths. Our settings are the following:

- **Max epochs**: 500
- **Learning rate**: We start with 5e-3 and gradually lower to 1e-5 with step learning rate scheduler with $\gamma = 0.3$. We do a step after the first 50 epochs, and then check every 50th epoch if the mean recall for thresholds $[1, 2, 5, 10, 25, 50]$ has not increased, then a step is taken.
- **Early stopping**: We do early stopping if one of the following criterion are met
  - **No improvement**: The end learning rate has been reached.
  - **Residuals are low enough**: $s < \frac{7}{f}$ (f is the focal length).
  - **Max time hit**: The max epochs have been reached.
- **Optimizer**: Adam [24]
- **MLP details**
  - Seven layers
  - Input dim: 22 ($2(5{\cdot}2{+}1)$ where we use five frequencies of sinus-cosinus pairs for each x, y dim concatenated with the raw normalized coordinates)
  - **Hidden dim**: 512
  - **Block structure**: Linear (with bias), LayerNorm, GELU.
- **Matching details**: (for sparse method)
  - **Number of keypoints**: 2048
  - **Detector and descriptor**: SuperPoint [10]
  - **Matcher**: LightGlue [30] (with no point pruning or early stopping)
- **Robustness**
  - We use PoseLib's [25] robust absolute pose estimator with a 16 pixel threshold.
  - In our network loss, we start with $s = \frac{100}{f}$ for the reprojection loss and $s = \frac{500}{f}$. for the depth loss (f is the focal length)

Method	$K=2$		$K=3$		$K=4$	
	$(\varepsilon_R, \varepsilon_t)\downarrow$	$(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)\uparrow$	$(\varepsilon_R, \varepsilon_t)\downarrow$	$(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)\uparrow$	$(\varepsilon_R, \varepsilon_t)\downarrow$	$(\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4)\uparrow$
Transitive Matching	119.65, 2.89	4.0, 13.0, 24.0, 28.3	2.67, 0.11	16.7, 37.0, 65.7, 74.0	2.24, 0.10	18.0, 43.7, 70.3, 79.3
Motion Averaging	1.56, 0.11	20.3, 57.7, 83.3, 90.3	1.42, 0.11	17.7, 53.3, 80.0, 92.7	1.95, 0.36	9.3, 32.0, 67.3, 84.3
VGGT	**1.09**, **0.08**	**31.3**, **70.3**, **95.7**, **98.7**	**1.12**, **0.07**	**30.0**, **80.0**, **96.0**, **100.0**	**1.18**, **0.07**	**31.3**, **76.0**, **98.3**, 98.7
Reloc3r	1.41, **0.08**	19.3, 67.3, 94.7, 98.3	1.54, **0.07**	19.0, 62.0, 93.7, 98.7	1.60, 0.10	10.7, 56.3, 95.0, **99.0**
ACE	11.80, 1.25	0.0, 1.3, 13.3, 33.3	12.50, 0.84	0.0, 1.0, 13.7, 39.0	17.52, 1.07	0.3, 1.7, 11.3, 30.7
ON3R (with RoMa)	1.56, **0.08**	18.0, 57.3, 90.0, 96.0	1.58, **0.07**	26.3, 59.3, 92.0, 95.3	1.54, 0.07	25.0, 60.7, 92.3, 98.0

Table 9. **Results on ScanNet for** $K = 2, 3, 4$. Metrics are median rotation/translation errors $(\varepsilon_R, \varepsilon_t)$ and recall $(\varepsilon_1$–$\varepsilon_4)$.

- **Implementation**: The code is implemented in PyTorch [34] with GPU acceleration.

To get an initial idea of good hyperparameter settings, we used the HPO library NePS on the training set of MegaDepth for all tuple lengths separately [47]. Then, we picked the final setting configuration based on what worked for all tuple lengths and checked that this gave good results on the other datasets as well.

E. Further Visualizations

In this section, we provide additional visualizations.

E.1. Additional Smoothness Examples

In Figs. 5 and 6 we show two additional cases of how our network learns smoothness. The Fig. 5 example shows that even though the orientation of the database cameras differs a lot, the network still learns an accurate smooth representation of the full 3D scene. Interestingly, the network learns a smooth map from the query image pixels on the fountain to the buildings behind it. The example in Fig. 6 is much easier, but it demonstrates how accurate lines, details, and surfaces we obtain on the facade, even though we have no explicit supervision there.

E.2. 3D Scene Progression

The longer our network trains, the more accurate the regressed 3D model typically becomes. We visualize in Fig. 7 the progression of the 3D point cloud during the network epochs, after the training is finished, and after the bundle adjustment is completed. As can be seen, the scene is initially a random blob. After only 10 epochs, a rough shape of the gate starts to emerge. It is then increasingly refined until the network reaches 60 epochs and the stop criterion is met. Already then, we can see what type of object is represented. However, after the bundle adjustment, the representation is further refined, giving a realistic-looking outline.

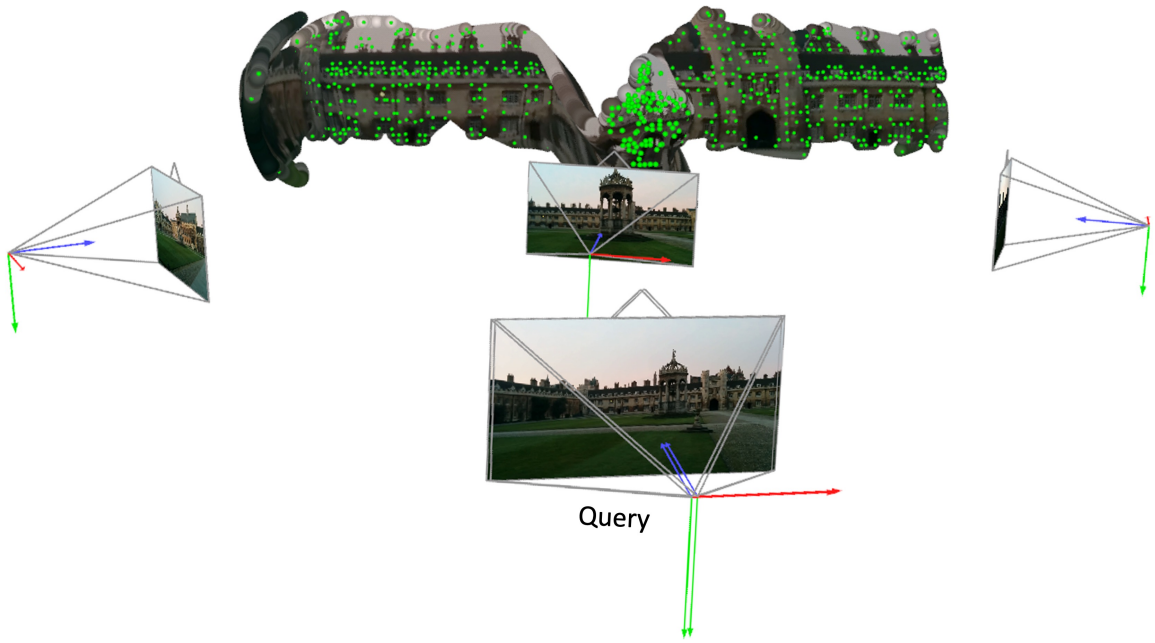

Figure 5. **Network smoothness on GreatCourt**. A dense reconstruction of GreatCourt from Cambridge Landmarks learned by our network. The green points are the only ones used during training, whereas the rest of the pixels (the concave hull of the query keypoints) lifted to 3D space are *given for free* following the network smoothness. The 3D points here have not been refined with bundle adjustment, so in practice, the locations of the query keypoints are even more precise.

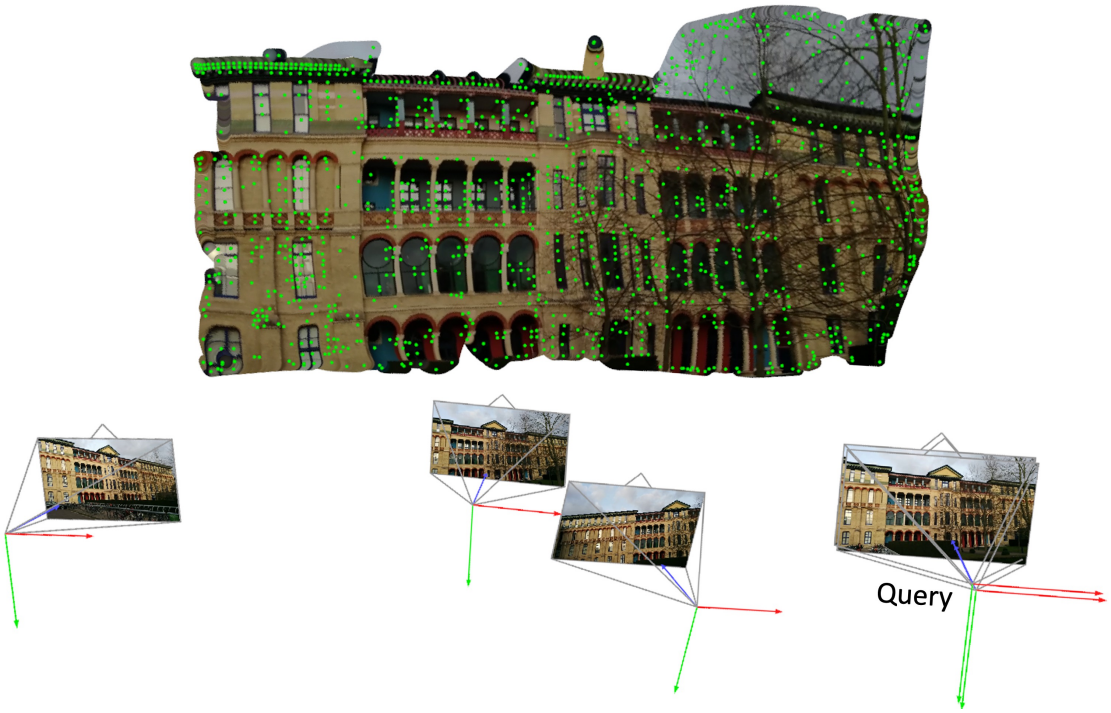

Figure 6. **Network smoothness on OldHospital**. A dense reconstruction of OldHospital from Cambridge Landmarks learned by our network. The green points are the only ones used during training, whereas the rest of the pixels (the concave hull of the query keypoints) lifted to 3D space are *given for free* following the network smoothness. The 3D points here have not been refined with bundle adjustment, so in practice, the locations of the query keypoints are even more precise.

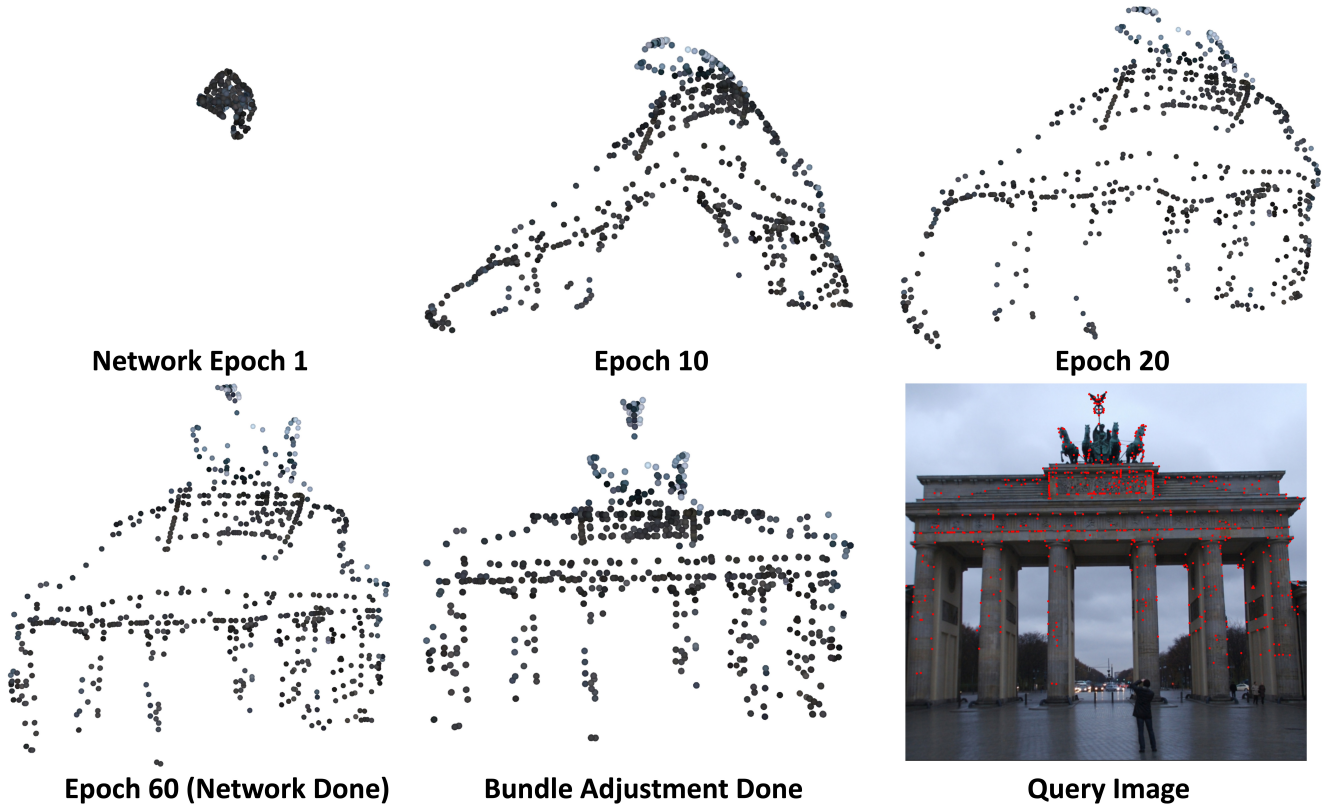

Figure 7. **Point cloud evolution**. The regressed 3D point cloud gradually improves over training epochs. Starting from a random initialization, the network quickly captures the coarse structure of the gate after 10 epochs and progressively refines it until convergence at 60 epochs. The final bundle adjustment further sharpens the reconstruction, yielding a realistic and geometrically consistent outline. In the bottom right corner, we show the query image along with the matched query keypoints, all of which have a corresponding 3D point.