Accelerated Coordinate Encoding: Learning to Relocalize in Minutes using RGB and Poses – Supplement –

Eric Brachmann
Niantic

Tommaso Cavallari
Niantic

Victor Adrian Prisacariu
Niantic, University of Oxford

In this supplementary document we provide further details on ACE, its evaluation, and the comparison with other methods published in literature. Specifically, in Section 1, we start by describing some details of the overall network architecture and training protocol we used to deploy and evaluate ACE in the main text:

- architecture and training process of the scene-agnostic feature backbone;
- architecture and training process of the scene-specific head network;
- choice of the loss function, including experimental validation;
- details of the "Ensemble" variant of ACE (see *Poker* in Table 2 of the main text), including an experimental analysis on the size of the ensemble.

In Section 2 we extend the experimental evaluation section of the main paper:

- derivation of the numbers we give for "mapping time" and "map size" of the other algorithms in Tables 1 and 2 of the main paper;
- details on the curation of the new Waypots dataset;
- experiments showing the impact of training with and without correlated gradients;
- experiments coupling different off-the-shelf feature backbones with ACE;
- experiments varying backbone feature dimensionality;
- details on how we extract 3D point clouds from a trained ACE network to visualize the map.

1. Implementation Details

1.1. Backbone Training

Our backbone architecture consists of the first 10 layers (including skip connections) of the DSAC* [7] network design. The backbone takes a gray scale image as input, and successively decreases the spatial resolution to $\frac{1}{8}$ while increasing the channel dimensions to 512.

We train our backbone on 100 scenes in parallel, attaching 100 regression heads to it. Each regression head is a multi-layer perceptron with 6 layers and width 512. There is a skip connection after the first 3 layers of each head. We train the backbone with half-precision floating point weights. We apply strong data augmentation to the mapping images during backbone training. We use a brightness and contrast jitter of 40%; a saturation and hue jitter of 30% (before converting to grayscale); and we randomly re-scale the images between 240px and 960px height. We use an inverse scale sampling strategy, *i.e.* we rescale images according to $\frac{1}{s}$ where s is a uniformly random scale factor. This mimics scale changes caused by the camera uniformly moving towards a scene. We warp images using homographies that correspond to random 3D rotations of up to 40°.

We train the backbone with a batch size of 6 images per regression head. To avoid memory issues, we compute forward and backward passes for 10 regression heads at a time. We accumulate gradients for all 100 regression heads before triggering a parameter update. For training we use the first 100 training scenes of ScanNet [9], *i.e.* scenes *scene0000* to *scene0099*. ScanNet provides multiple scans per scene, but they are not aligned. Thus, we use only one (the first) scan for each scene, *i.e.* scans *scene0000_00* to *scene0099_00*.

1.2. ACE Head Training

1.2.1 Network Design

The multi-layer perceptron we use as scene coordinate regression head is composed of 8 1x1 convolutional layers, all of width 512, with skip connections after layer 3 and 6; followed by a final 1x1 convolutional layer producing the scene coordinates. All the head layers use half-precision floating point weights. As mentioned in the main text, we experimented with both directly regressing 3D scene coordinates (in which case the last layer would output a 3-channel tensor), and regressing homogeneous coordinates. In the latter case, the last layer outputs a 4D tensor of $(\dot{x}, \dot{y}, \dot{z}, \hat{w})$, with $\dot{\mathbf{y}} = (\dot{x}, \dot{y}, \dot{z})^T$ being the homogeneous representation of the 3D scene coordinates, and $\hat{w} \in \mathbb{R}$ being an un-normalized homogeneous parameter. Before de-homogenizing the scene coordinates we compute $w \in \mathbb{R}^+$ from $\hat{w}$ applying a biased (and clipped) Softplus operator to ensure that the final 3D scene coordinates would be valid and usable in the following RANSAC step.

Specifically, we compute w as follows:

$$w = \min\left(\frac{1}{S_{min}}, \beta^{-1} \cdot \log\left(1 + \exp\left(\beta \cdot \hat{w}\right)\right) + \frac{1}{S_{max}}\right) \tag{1}$$

Where S_{min} and S_{max} are the values we use to clip the scale factor determined by w (in our experiments we set them to 0.01 and 4.0 respectively), and β is the parameter used to ensure that when the network outputs $\hat{w} = 0$, the resulting homogeneous parameter w becomes 1. This is to steer the network towards producing a "neutral" homogeneous parameter (centered on 1). As such, for our experiments, we set $\beta = \frac{\log(2)}{1 - S_{max}^{-1}}$.

We then de-homogenize the output of the network into the tensor $\mathbf{y}$ containing 3D scene coordinates, as usual:

$$\mathbf{y} = \frac{\dot{\mathbf{y}}}{w} \tag{2}$$

In both cases the coordinates output by the network, for numerical stability, are learned relatively to the "mean" translation of the camera poses associated to the mapping frames. The final step of the scene coordinate regression process is then to add the mean back to the prediction, before passing the tensor of 3D scene coordinates to the PnP-RANSAC stage.

1.2.2 Training Details

As mentioned in the main text, key to the speed with which we are able to train an ACE map is the fact that, first, we pre-fill a buffer of patch features extracted from the set of training images (using the scene-agnostic backbone, this takes $\sim$1min), and then we just need to optimize the weights of the regression head.

More specifically, we allocate a buffer able to contain 8 million 512-channel patch descriptors, together with their associated 2D location in the source image, mapping camera pose, and intrinsic parameters. We fill the buffer by repeatedly cycling over the shuffled training sequence. Each image is augmented using a similar approach as the backbone training, but with a different set of hyperparameters that we detail next. We apply a brightness and contrast jitter of 10%, but no color and saturation jitter; randomly rescale the input images between 320px and 640px height; and, finally, we apply in-plane random rotations of up to $15°$. We can use these weaker data augmentation parameters to still train the ACE regression heads accurately because the backbone has already been trained on strongly augmented images. The training images are passed through the convolutional backbone and, for each one, we randomly select 1024 patches and their corresponding feature descriptors, which we then copy into the training buffer, together with the other metadata (2D patch location, camera pose and intrinsics).

We train the ACE head by repeatedly iterating over the shuffled training buffer, as described in the main text. Specifically, for a 5 minute mapping time (including the time spent filling the buffer), we do this 16 times (epochs).

1.2.3 Choice of Loss Function

In order to train the ACE head networks to regress accurate 3D scene coordinates, we use a $\tanh$-based loss on the reprojection errors, dynamically rescaled according to a circular schedule with a threshold decreasing throughout the length of the training process (see Sec. 3.2 of the main text for details). We experimented with alternative loss functions as well, before settling on the one used in the main text, and in Table 1 we show their accuracy on the 7 Scenes dataset [27].

Firstly, we experimented with the reprojection loss described in the DSAC* paper [7]. This loss is computed piecewisely, using the L1 norm of the reprojection error up to a threshold of $\tau = 100$px, and switching to the square root of the L1 norm past it. As described in the main text, we also experimented with $\tanh$ losses, which effectively clamp each patch's individual contribution to the overall error used for optimization during training. The right side of Table 1 shows the accuracy for different variants of the $\tanh$ loss: (a) using a constant τ threshold set at 50px throughout the training process; (b) *linearly* decreasing τ from 50px to 1px, *i.e.* the $w(t)$ in Eq. 8 of the main text computed as: $w(t) = 1 - t$; (c) *quadratically* decreasing τ from 50px to 1px (the circular schedule, as in Eq. 8 of the main text); and, finally, (d) applying the circular schedule as in (c) but also using the homogeneous coordinate prediction, as described in the previous section (note that all other columns of the table were directly predicting 3D coordinates).

Overall we see that using the circular schedule, with a threshold decreasing over time, in order to focus the network training to improve the prediction of reliable scene coordinates (with the knowledge that unreliable predictions will be effectively filtered by the RANSAC stage during localization) allows us to improve from the results obtained using DSAC*'s piecewise loss. We also see how introducing the overparametrized homogeneous representation consistently improves the overall accuracy of the method, especially on difficult scenes such as "RedKitchen" and "Stairs". As such, the latter is the variant we chose to deploy throughout the paper.

Scene	$L1 \rightarrow \sqrt{L1}$ (DSAC* [7])	tanh-based			
		fixed $\tau = 50px$	linear schedule $\tau = 50 \rightarrow 1px$	circular schedule $\tau = 50 \rightarrow 1px$	circular schedule + homogeneous coords
Chess	100	100	100	100	100
Fire	99.1	99.3	99.5	99.3	99.5
Heads	99.8	99.8	99.8	99.7	99.7
Office	99.5	99.6	99.5	99.7	100
Pumpkin	100	99.9	99.8	99.8	99.9
RedKitchen	96.7	97.0	97.0	98.3	98.6
Stairs	69.4	68.1	66.8	70.5	81.9
Average	94.9	94.8	94.6	95.3	97.1

Table 1. **Loss Functions.** Performance of ACE on the 7 Scenes dataset [27] when trained using different loss functions. For each scene we report the %-age of frames localized within 5cm/5° pose error. All columns except the right-most one deploy an ACE regression head directly predicting 3D scene coordinates (see Sec. 1.2.1). The last column shows the contribution of the homogeneous coordinate encoding, which is helpful, especially in a complex scene like "Stairs". As such, the results in the main text use the tanh-based loss with circular schedule and homogeneous coordinates.

1.3. Ensemble Variant

In the main text, we introduced an *ensemble* variant of ACE that we evaluated on the five scenes of the Cambridge Landmarks dataset [16]. We named that variant *"Poker"*, as it is composed of four identical but independently trained network heads, collaboratively contributing to the estimation of the camera poses in the larger environments part of the dataset. The main reasoning behind the introduction of the ensemble of networks is that some of the scenes of the dataset are covering a large area around the landmarks in the city of Cambridge.

We postulated that, instead of training a single ACE head to regress 3D scene coordinates for the entirety of each landmark (a hard task, considering the map size and training times we target), we could instead cluster the mapping frames into sub-regions and deploy multiple regression heads, each focusing on a specific area of the scene.

In the following paragraphs, we describe how we split the training data and ran localization to produce the results we showed in Table 2 of the main text. We also present additional results detailing how the ensemble variants perform with a varying number of independently trained regression heads.

1.3.1 Ensemble Training

Previous works [6] showed how using a *Mixture of Experts* can help improving the performance of large-scale outdoor camera localization. In this paper we adopt an approach inspired by the ESAC paper [6], training multiple ACE heads on independent subsets of frames part of the training split of each scene. We adapt the code they publicly released in order to spatially cluster the images according to the position of the camera at capture time. This is slightly different from the approach described in the ESAC paper, as they used the median scene coordinate for each image – which is available as part of the Cambridge Landmarks training set. We decided not to use that information in order to make the method more resembling of a realistic use-case, where the only information available at mapping time would be the pose of the camera (provided by ARKit [1]/ARCore [13]), but not the 3D point cloud of the scene (which would instead have to be reconstructed offline via SfM).

Specifically, given a set of posed RGB images $\mathcal{I}_\mathrm{M}$ to use as training to map a large-scale environment, we apply hierarchical clustering to the translation component of each frame's pose in order to obtain N disjoint training sets $[\mathcal{I}_{\mathrm{M}_1}..\mathcal{I}_{\mathrm{M}_N}]$. Clustering is performed as follows:

- Initialization: Put all images in the first cluster $\mathcal{I}_{\mathrm{M}_1}$.
- While the target number of clusters N has not been reached:
  - Select the largest cluster (by number of images associated to it).
  - Split it into two clusters using kMeans [4] (The input to kMeans is the translation component of the camera pose).
  - Replace the large cluster with the two new ones output by kMeans.
- Return the N clusters $[\mathcal{I}_{\mathrm{M}_1}..\mathcal{I}_{\mathrm{M}_N}]$.

After clustering, we train N ACE regression heads f_{H_i}, one for each set of training images. As each head is trained independently from each other, both the mapping time and the map size scale linearly with the number of elements part of the ensemble, although the process can be trivially parallelized to save time by using multiple GPUs, if available.

1.3.2 Ensemble Localization

Localization for the ensemble variant of ACE is straightforward: we simply run the standard localization pipeline separately for each trained map, then pick the camera pose having the most inliers after RANSAC and LM-based refinement.

It's worth noting that, since the feature extraction backbone is scene-agnostic, we can optimize the localization phase by running the query images through the backbone f_B just once, outputting image features $\mathbf{f}_i$. We can then pass such features through the N ACE coordinate regression heads f_{H_i} to regress the scene coordinates, which are then passed to N instances of the RANSAC algorithm for pose estimation.

1.3.3 Ensemble Results

In Table 2 we show how the performance of ACE on the Cambridge Landmarks dataset [16] varies when changing the number of maps part of the ensemble. Note how the error initially decreases while we increase the number of ACE heads being trained – hinting to the fact that limiting the spatial extent covered by the frames used for training is advantageous with such a fast/aggressive training regime – then saturates past the ensemble with 4 ACE regressors. The fact that the "Shop" scene does not seem to benefit from using multiple ACE heads for regression can be explained by considering that the mapped area is already limited, thus not requiring further clustering. We chose to present the results for an ensemble of four maps (which we named *"Poker"*) in the main text because it achieves better results than the baseline method (DSAC*) within a fraction of the mapping time.

The clustering and localization approach described in this section has been deliberately kept simple to showcase the potential of using multiple regression heads to map large areas (as this is not the focus of this work, but merely an extension) but can, with clever engineering and tuning, be made faster and more accurate, for example by tailoring the size of each ensemble to the area covered by its scene.

2. Experimental Details

2.1. Resource Footprint of Competitors

2.1.1 Active Search [23, 24]

Mapping Time. For 7Scenes [27] and 12Scenes [30], we report the average triangulation time of COLMAP [25] using SIFT [18] features and known camera poses. These timings were kindly provided by the authors of [5] who generated alternative SfM ground truth for the aforementioned datasets. They ran feature detection and parts of feature matching on GPU, while triangulation ran on CPU. Reconstruction times vary a lot between scenes: *Heads* is the fastest scene with 6 minutes reconstruction, *RedKitchen* is the slowest scene with 10 hours reconstruction. On average, 7Scenes took 3.3h per scene to reconstruct, and this is the time we report for Active Search in Fig. 1 of the main paper. 12Scenes reconstructed in 35 minutes on average per scene, due to having fewer mapping frames than 7Scenes. Averaging the reconstruction times across all scenes of 7Scenes and 12Scenes yields 1.5h, the number we give in Table 1 of the main paper.

For Cambridge Landmarks [16], we found no public record of reconstruction times. The SfM pseudo ground truth was generated by running VisualSfM [31] on the combined set of training and test images. It is unclear whether running SfM on the training set alone, as one would have to do in practise, would result in camera poses of comparable quality. In terms of mapping frame count, we found Cambridge Landmarks comparable to 12Scenes on average. Thus, we assume that the 35 minutes reconstruction time for 12Scenes would approximately also hold for Cambridge Landmarks, and report this number in Table 2 of the main paper.

Note: These times do not include any scene-specific preprocessing that the relocalizer might do on top of the SfM reconstruction, such as building acceleration data structures for faster matching.

Map Size. For each scene, Active Search needs to store the 3D point cloud, a list of feature descriptors per 3D point, as well as co-visibility information and a visual dictionary. Overall, feature descriptors constitute the largest portion of the map size, and we disregard all other factors. Active Search stores SIFT [18] descriptors, *i.e.* 128 byte if stored in unsigned char precision. At most, Active Search could store all observed descriptors for each 3D point in the SfM reconstruction, *i.e.* one descriptor for all feature detections that lead to a triangulated 3D point. However, in practise, Active Search runs a clustering algorithm on all descriptors per 3D point, and creates one representative descriptor per cluster. Based on communication with the authors of [24], we assume that Active Search stores descriptors for only 30% of feature observations. The COLMAP reconstruction statistics reported in [5] include the number of triangulated 3D points, and the number of feature observations for 7Scenes [27] and 12Scenes [30]. We use these statistics to calculate an average descriptor storage demand of 200MB per scene for these two datasets. This is the number we give in Table 1 of the main paper.

For Cambridge Landmarks [16], Camposeco *et al.* report an average memory demand of 200MB per scene in [8] for Active Search. This is the number we give in Table 2 of the main paper.

Scene	DSAC* (Full) [7]	ACE	ACE Ensemble					
			2 models	3 models	4 models	5 models	6 models	7 models
Court	34/0.2	43/0.2	32/0.2	32/0.2	28/0.1	29/0.1	28/0.1	27/0.1
King's	18/0.3	28/0.4	23/0.4	20/0.4	18/0.3	18/0.3	18/0.3	16/0.3
Hospital	21/0.4	31/0.6	22/0.4	23/0.5	25/0.5	24/0.5	25/0.5	23/0.5
Shop	5/0.3	5/0.3	6/0.2	5/0.3	5/0.3	5/0.3	6/0.3	5/0.3
St. Mary's	15/0.6	18/0.6	13/0.4	12/0.4	9/0.3	9/0.3	9/0.3	9/0.3
Average (cm/°)	19/0.4	25/0.4	19/0.3	18/0.4	17/0.3	17/0.3	17/0.3	16/0.3
Mapping Time	15h	5min	10min	15min	20min	25min	30min	35min
Map Size (MB)	28MB	4MB	8MB	12MB	16MB	20MB	24MB	28MB

Table 2. **ACE Ensemble.** Performance of the ACE Ensemble variants on scenes from the Cambridge Landmarks dataset [16]. We report the median position and rotation errors, together with the time and storage required for each map.

2.1.2 D.VLAD+R2D2 [19]

Mapping Time. We report the average triangulation time of COLMAP [25] in Table 1 of the main paper. Please see our reasoning for Active Search, explained above. This assumes that a COLMAP reconstruction using R2D2 [19] takes approx. the same time as when using SIFT [18].

Map Size. Following our reasoning for Active Search (see above) we approximate the map size with the memory needed to store descriptors. We assume that D.VLAD+R2D2 stores descriptors for all feature observations, and uses 256 bytes per descriptor for 128 floating-point entries with half precision. We found no public record of R2D2 reconstruction statistics for 7Scenes [27] and 12Scenes [30]. Re-using the reconstruction statistics of [5], we arrive at 1GB map size on average. This is the number we give in Table 1 of the main paper.

2.1.3 hLoc (SP+SG) [20, 21]

Mapping Time. We report the average triangulation time of COLMAP [25] in Table 1 and Table 2 of the main paper. Please see our reasoning for Active Search, as explained above. This assumes that a COLMAP reconstruction using SuperPoint [10] takes approximately the same time as when using SIFT [18] features.

Map Size. Following our reasoning for Active Search (see above) we approximate the map size with the memory needed to store descriptors. We assume that hLoc stores descriptors for all feature observations, and uses 512 bytes per descriptor for 256 floating-point entries with half precision. We found no public record of SuperPoint reconstruction statistics for 7Scenes [27] and 12Scenes [30]. Re-using the reconstruction statistics of [5], we arrive at 2GB map size on average across 7Scenes and 12Scenes. This is the number we give in Table 1 of the main paper. For 7Scenes alone, we arrive on 3.5GB average map size, and this is the number we state in Fig. 1 of the main paper. This aligns well with the 3.3GB average map size for hLoc on 7Scenes that is reported by Zhou *et al.* in [33]. For Cambridge Landmarks [16], Zhou *et al.* report an average memory demand of 800MB per scene in [33] for hLoc (SP+SG). This is the number we give in Table 2 of the main paper.

2.1.4 pixLoc [22]

Mapping Time. pixLoc is a method that works on top of existing 3D map reconstructions. Thus, as mapping time in Table 1 and Table 2 of the main paper, we report the average triangulation time of COLMAP [25] using SIFT [18] features, see our reasoning for Active Search above.

Map Size. pixLoc relies on dense features to optimize map-to-image alignment. We assume that it is more advantageous to store the original mapping images and compute features on the fly, than it is to store dense descriptors for all mapping images. Additionally, [22] uses DenseVLAD [28] to retrieve the top reference images for each query image, together with the scene point cloud to associate 3D coordinates to the pixels of the reference images. Thus, in Table 1 and Table 2 of the main paper we approximate the map size of pixLoc with the average memory demand of storing all mapping images, as the additional space required to store retrieval descriptors and point cloud is tiny in comparison.

2.1.5 hybridSC [8]

Mapping Time. hybridSC is a compression algorithm that reduces the average size of feature descriptors. It constitutes a post-processing on top of existing 3D map reconstructions. We ignore any overhead incurred by descriptor compression, and report the average triangulation time of

COLMAP [25] using SIFT [18], see our reasoning for Active Search above. This is the number we give in Table 2.

Map Size. We report the average memory demand per scene given in [8] in Table 2 of the main paper.

2.1.6 GoMatch [33]

Mapping Time. GoMatch estimates a pose by matching constellations of 2D feature detections against a 3D point cloud. Thus, as mapping time in Table 2 of the main paper, we report the average triangulation time of COLMAP [25] using SIFT [18] features, see our reasoning for Active Search above.

Map Size. We report the average memory demand per scene given in [33] in Table 2 of the main paper.

2.1.7 PoseNet17 [15]

Mapping Time. As mapping time in Table 2 of the main paper, we report the range of training times given in [15].

Map Size. We report the memory demand given in [8].

2.1.8 MS-Transformer [26]

Mapping Time. We used the code released by the authors of the paper [26] to get an estimate of the time required to train a Multi-Scene Transformer network on the four scenes from the Cambridge Landmarks dataset [16] they report results for (the paper doesn't report results for the "Great Court" scene). We found that, on a machine equipped with a Nvidia RTX A6000 GPU, training the model for 1 epoch takes approximately 3 minutes. From the paper we saw that, on the Cambridge dataset, the training should run for 550 epochs initially, followed by 40 epochs in which only the position branch is optimized. This means that the network would be fully trained in approximately 29.5h but, as the same network can then be used to localize in all four scenes of the dataset, in Table 2 of the main text we report the "average" per-scene time of 7h.

Map Size. As per the paper [26], the trained network has a memory footprint of 74.6MB. Since the network can be used to localize in all four scenes of the Cambridge Landmarks dataset considered by the authors, in the main text we report an "average" per-scene map size of 18MB.

2.1.9 SANet [32]

Mapping Time. The SANet paper uses a scene-agnostic network to interpolate the coordinate maps associated to the top-scoring images returned by a retrieval method such as NetVLAD [2]. As such we can consider the time it takes to create the retrieval index for the set training images as the mapping time. In [32], the authors report that each forward pass of NetVLAD takes approximately 0.06s. We can compute the average mapping time per scene by simply multiplying the above by the number of frames in the training sets of the various datasets. For the indoor datasets (7 Scenes [27] and 12 Scenes [30]) the average scan length is 2259 frames. As such, we report in Table 1 of the main paper a mapping time of $\sim$2.3min. For the Cambridge Landmarks dataset [16] the average training scan length is 1073 frames, so we report a mapping time of $\sim$1min in Table 2.

Map Size. According to [32], for each element of the training set, SANet requires to store the (resized) RGB-D image, its pose, and the corresponding NetVLAD description, for a total of 248kB. Similarly to the previous paragraph, we use the avg. length of the training scans in both indoor and outdoor datasets to derive the "average" size of SANet maps. In the main paper we thus report $\sim$550MB for the indoor maps and $\sim$260MB for the outdoor maps.

2.1.10 SRC [11]

Mapping Time. The Scene Region Classification approach [11] is based on a neural network classifying each pixel of the query images into one of the discrete regions determined via hierarchical partitioning of RGB-D mapping images. As such, the mapping time is comprised of two components: (a) the time required to create the region partitions, and (b) the training of the Scene Region Classification network, learning to classify pixels in the input images into their respective region. In [11] the authors claim that the training of the region classification network converges in $\sim$2min, so we report that in Tables 1 and 2 of the main text, but that doesn't include the time required to partition the training data. In our experiments with the code publicly released by the authors we saw that hierarchical partitioning takes between 1 and 2 minutes per scene (on 7 Scenes [16]), and that ought to be added to the mapping time.

Map Size. In [11] the authors report that a trained network occupies $\sim$40MB.

2.2. Wayspots Dataset

2.2.1 Scene Selection

We take the 10 scenes of our Wayspots dataset from the training split of the MapFree [3] dataset. The MapFree dataset is designed for relative pose regression from pairs of images. As training data, Arnold *et al.* [3] provide 460 outdoor scenes with two fully registered scans each. These scans are meant as source for sampling image pairs to train relative pose regression. We re-purpose this data for standard visual relocalisation using one scan as mapping sequence, and using the second scan for queries. We reserve the first 200 scenes of the MapFree training corpus for training an ACE backbone variant, see Sec. 2.4. For our Wayspots dataset, we selected the scenes *s00200 - s00209*.

2.2.2 Pseudo Ground Truth

The MapFree dataset consists of scans from phones that ran RGB-based visual odometry to track camera poses. Arnold *et al*. [3] triangulated each scan using the phone's camera poses, and registered scans of the same scene using COLMAP [25]. They also refined camera poses using bundle adjustment. The bundle-adjusted SfM poses are what the MapFree dataset provides as pseudo ground truth.

However, we are interested in using the original phone poses for mapping since they can be generated in real-time, and thus incur practically no further mapping overhead. Arnold *et al*. [3] kindly provided the original phone poses based on our request. We can use the phone poses of the mapping scan for creating a scene representation. However, phone poses of the query scan are not registered to the mapping poses. To enable evaluation, we register the phone poses of the mapping scan to their SfM counterpart poses, essentially registering them to the SfM query poses. In doing so, we regard the SfM poses as the unobservable pseudo ground truth, and the phone poses as the less precise – but observable – mapping input.

Note: Although our dataset does not use SfM poses for mapping, SfM-based relocalizers might still have an advantage on these scenes. The MapFree dataset does only contain scenes where COLMAP triangulation and COLMAP registration was successful. Thus, there might be a selection bias in favour of sparse feature-based methods. In our main paper, we only compare scene coordinate regression approaches on this dataset, where we do not expect a bias towards one of the methods [5].

2.2.3 Phone-to-SfM Pose Alignment

To register phone poses to SfM poses, we use the Kabsch algorithm [14] within a RANSAC loop [12], relying on 3D camera positions alone. We sample 1000 random triplets of corresponding camera positions to generate registration hypotheses. We rank hypotheses by inlier counting with a threshold of 10cm. We re-fit the winning hypotheses to all inlier correspondences. Note that Arnold *et al*. have already re-scaled the SfM poses according to the phone trajectories after bundle adjustment to prevent scale drift [3].

Due to SfM poses being bundle-adjusted, we cannot expect perfect alignment with the original phone poses. We list alignment errors for each scene in Table 3. Since we do not use camera rotation in our pose registration, statistics over rotation alignment provide a good plausibility check on the success of our approach. We see that some scenes exhibit noticeable drift between poses from phone tracking and poses from SfM. However, the median alignment error is below 10cm for all scenes, such that at least 50% relocalisation rate is achievable for all scenes using our proposed error threshold of 10cm and 5°. Better results are possible if a method refines mapping poses during mapping.

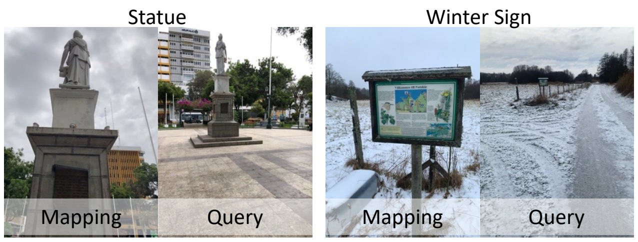

Figure 1. **Hard Scenes.** Example frames from the two hardest scenes of the Wayspots dataset. These scenes where mapped from close-up but queried from afar.

To verify that alignment errors are not the main cause of poor results on some scenes, we repeat experiments with ACE using SfM poses for mapping, see Table 3. While results do indeed improve, they improve only moderately on average. Difficult scenes remain difficult (such as "Winter Sign" and "Statue") even when factoring out alignment errors of the pseudo ground truth. See Fig. 1 for example frames of these two scenes, highlighting their challenges.

2.3. Gradient Decorrelation

We train ACE with correlated gradients to show the detrimental effect on accuracy and stability. Akin to DSAC* [7], we train ACE by using all features per mapping image, and by shuffling the ACE buffer image-wise. In this variant, ACE sees correlated features from a single mapping image during each training iteration, as opposed to random features selected from the entire mapping sequence. Accuracy deteriorates across all datasets, see Table 4. The effect is particularly pronounced on Cambridge where individual scenes, such as St. Marys Church, diverge completely. Note that ACE with correlated gradients still exceeds the 5 minute version of DSAC*. ACE starts with strong backbone features, and does more training iterations in the same time due to the small network.

2.4. Feature Backbone

We combine ACE training with some popular feature backbones and report results in Table 5. Training a backbone specifically for the task of scene coordinate regression achieves best results. Descriptor backbones of sparse feature-matching pipelines use 128 or 256 feature dimensions for faster matching, and leaner storage. In contrast,

Scene	Number of Frames		Phone-to-SfM Alignment Residuals		ACE Results (10cm,5°)	
	Mapping	Query	Position (mean/median/max)	Rotation (mean/median/max)	Mapping w/ Phone Poses	Mapping w/ SfM Poses
Cubes	581	575	2.6cm/1.9cm/8.6cm	0.6°/0.6°/0.9°	97.0%	97.6%
Bears	581	580	1.3cm/1.2cm/3.1cm	0.5°/0.5°/0.9°	80.7%	80.9%
Winter Sign	580	580	2.1cm/1.4cm/13.8cm	0.6°/0.6°/2.2°	1.0%	0.7%
Inscription	578	555	4.4cm/3.7cm/21.5cm	1.0°/1.0°/1.5°	49.0%	59.5%
The Rock	579	578	5.1cm/5.5cm/10.2cm	0.7°/0.8°/1.1°	100%	100%
Tendrils	580	581	8.4cm/4.6cm/26.9cm	1.7°/1.7°/2.0°	34.9%	34.6%
Map	575	503	7.2cm/4.5cm/23.2cm	1.0°/0.9°/1.9°	56.5%	87.7%
Square Bench	580	577	8.4cm/5.9cm/22.1cm	0.7°/0.6°/1.0°	66.7%	93.9%
Statue	560	553	14.5cm/9.3cm/43.6cm	0.5°/0.5°/1.2°	0%	0%
Lawn	575	579	3.3cm/3.2cm/6.2cm	0.4°/0.5°/0.9°	35.8%	40.4%
Average					52.2%	59.5%

Table 3. **Wayspots Pose Alignment.** Evaluation of the alignment quality between the phone's poses (estimated using visual odometry, in real time) and bundle-adjusted SfM poses obtained offline (used as pseudo ground truth to evaluate camera relocalization results) on the Wayspots dataset. We also show, on the right, the difference in performance achieved by ACE when using phone poses, vs. using SfM poses during mapping. In Figure 4 of the main paper we report the results obtained using the former, as that's ACE's intended use case.

Method	Gradients	Mapping Time	7Scenes		12Scenes		Cambridge	Wayspots
			SfM poses	D-SLAM poses	SfM poses	D-SLAM poses	(avg. median error)	
DSAC*	correlated	🟥 15h	96.0%	81.1%	99.6%	98.8%	19cm/0.4°	50.7%
DSAC*	correlated	🟩 5min	3.6%	5.5%	0.0%	0.2%	2375cm/49°	7.4%
ACE	correlated	🟩 5min	91.2%	72.9%	74.3%	75.2%	406cm/8.7°	49.7%
ACE	decorrelated	🟩 5min	97.1%	80.8%	99.9%	99.6%	25cm/0.4°	52.2%

Table 4. **Gradient Decorrelation.** The gray line marks a version of ACE where the training buffer is shuffled image-wise instead of feature-wise, akin to the training procedure of DSAC*. Gradient decorrelation improves accuracy and stability of ACE.

scene coordinate regression can benefit from higher dimensional features with neglectable impact on computation time or memory. Training our backbone on ScanNet [9] yields best results. We also trained a backbone variant on the MapFree dataset [3]. We used scenes *s00000* to *s00099*[1], and both available scans per scene since they are registered. The MapFree backbone performs slightly worse than the ScanNet backbone. Presumably ScanNet scenes, being indoors, contain less uninformative areas like sky or ground.

2.5. Map Visualization

To create 3D representations of the maps learned by ACE (as shown in Fig. 4 and 5 of the main paper), we run the mapping sequence through the learned network and accumulate 3D scene coordinate predictions. To obtain colored point clouds, we rescale the mapping images to the network output resolution, and read out the color of each predicted point. To de-clutter the point clouds, we remove coordinate predictions that are more than 10m from the image plane.

[1]We reserved the first 200 scenes of the MapFree training set for encoder experiments but ended up only using the first 100 scenes.

Backbone	Descriptor Dim.	Accuracy (7Scenes, 5cm5°)
DenseSIFT [17]	128	83.0%
DISK [29]	128	73.6%
R2D2 [19]	128	87.7%
SuperPoint [10]	256	91.6%
ACE [MapFree [3]]	512	91.5%
ACE [ScanNet [9]]	128	95.8%
	256	96.4%
	512	**97.1%**

Table 5. **Backbones.** We can use multiple off-the-shelf backbones with ACE. Our backbone trained on ScanNet achieves best results.

To visualize camera trajectories, we connect consecutive camera positions with a line. We skip lines when cameras are further than 0.5m apart, assuming an outlier estimate. To also show camera orientation, we place a camera frustum for 1 out of 25 frames.

References

[1] Apple. ARKit. Accessed: 11 November 2022. 3

[2] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *CVPR*, 2016. 6

[3] Eduardo Arnold, Jamie Wynn, Sara Vicente, Guillermo Garcia-Hernando, Áron Monszpart, Victor Adrian Prisacariu, Daniyar Turmukhambetov, and Eric Brachmann. Map-free visual relocalization: Metric pose relative to a single image. In *ECCV*, 2022. 6, 7, 8

[4] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. volume 8, pages 1027–1035, 01 2007. 3

[5] Eric Brachmann, Martin Humenberger, Carsten Rother, and Torsten Sattler. On the limits of pseudo ground truth in visual camera re-localisation. In *ICCV*, 2021. 4, 5, 7

[6] Eric Brachmann and Carsten Rother. Expert sample consensus applied to camera re-localization. In *ICCV*, 2019. 3

[7] Eric Brachmann and Carsten Rother. Visual camera re-localization from RGB and RGB-D images using DSAC. *TPAMI*, 2021. 1, 2, 3, 5, 7

[8] Federico Camposeco, Andrea Cohen, Marc Pollefeys, and Torsten Sattler. Hybrid scene compression for visual localization. In *CVPR*, 2019. 4, 5, 6

[9] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In *CVPR*, 2017. 1, 8

[10] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *CVPRW*, 2018. 5, 8

[11] Siyan Dong, Shuzhe Wang, Yixin Zhuang, Juho Kannala, Marc Pollefeys, and Baoquan Chen. Visual localization via few-shot scene region classification. In *3DV*, 2022. 6

[12] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981. 7

[13] Google. ARCore. Accessed: 11 November 2022. 3

[14] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 1976. 7

[15] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. *CVPR*, 2017. 6

[16] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-DOF camera relocalization. In *CVPR*, 2015. 3, 4, 5, 6

[17] Ce Liu, Jenny Yuen, and Antonio Torralba. SIFT flow: Dense correspondence across scenes and its applications. *TPAMI*, 2011. 8

[18] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004. 4, 5, 6

[19] Jerome Revaud, Philippe Weinzaepfel, César Roberto de Souza, and Martin Humenberger. R2D2: repeatable and reliable detector and descriptor. In *NeurIPS*, 2019. 5, 8

[20] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *CVPR*, 2019. 5

[21] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperGlue: Learning feature matching with graph neural networks. In *CVPR*, 2020. 5

[22] Paul-Edouard Sarlin, Ajaykumar Unagar, Måns Larsson, Hugo Germain, Carl Toft, Victor Larsson, Marc Pollefeys, Vincent Lepetit, Lars Hammarstrand, Fredrik Kahl, and Torsten Sattler. Back to the Feature: Learning Robust Camera Localization from Pixels to Pose. In *CVPR*, 2021. 5

[23] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Improving image-based localization by active correspondence search. In *ECCV*, 2012. 4

[24] Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Efficient & Effective Prioritized Matching for Large-Scale Image-Based Localization. *TPAMI*, 2017. 4

[25] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 4, 5, 6, 7

[26] Yoli Shavit, Ron Ferens, and Yosi Keller. Learning multi-scene absolute pose regression with transformers. In *ICCV*, 2021. 6

[27] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in RGB-D images. In *CVPR*, 2013. 2, 3, 4, 5, 6

[28] Akihiko Torii, Relja Arandjelovic, Josef Sivic, Masatoshi Okutomi, and Tomas Pajdla. 24/7 place recognition by view synthesis. In *CVPR*, 2015. 5

[29] MichałTyszkiewicz, Pascal Fua, and Eduard Trulls. DISK: Learning local features with policy gradient. In *NeurIPS*, 2020. 8

[30] Julien Valentin, Angela Dai, Matthias Nießner, Pushmeet Kohli, Philip Torr, Shahram Izadi, and Cem Keskin. Learning to navigate the energy landscape. In *3DV*, 2016. 4, 5, 6

[31] Changchang Wu. VisualSFM: A visual structure from motion system, 2011. 4

[32] Luwei Yang, Ziqian Bai, Chengzhou Tang, Honghua Li, Yasutaka Furukawa, and Ping Tan. SANet: Scene agnostic network for camera localization. In *ICCV*, 2019. 6

[33] Qunjie Zhou, Sérgio Agostinho, Aljoša Ošep, and Laura Leal-Taixé. Is geometry enough for matching in visual localization? In *ECCV*, 2022. 5, 6