A Details on experimental setup

A.1 Setup for LiDAR and camera input

Setup for LiDAR input: To learn robust features for the calibration of different transforms, we train our network to recover the true calibration from a variety of initial guesses $\hat{\mathbf{T}} = \mathbf{T}_p\mathbf{T}$. Where we perturb the ground truth $\mathbf{T}$ using uniformly sampled $\mathbf{T}_p$, modeled by a translation and angle-axis vector. We sample these vectors uniformly on the 3-sphere using [19] and scale them with a uniformly distributed scalar. For training, the translation and rotation magnitudes are in the range $[0, 1.5]$m and $[0, 15]^\circ$, respectively.

To account for different intensity profiles registered by different LiDAR models, and cater to the fact that some LiDAR drivers don't provide intensity readings, we augment the intensity channel of our LiDAR data to minimize our dependence on intensity information. We apply uniform random scalar perturbations in the range $[0, 1.0]$ to the intensity channel of the LiDAR points, meaning that in some samples the intensity information is close to dropped out.

Lastly, to process the LiDAR data using our sparse 3-D CNN, we voxelize the points using isotropic voxels of 2 cm per side. We found this resolution to be reasonable since it is in the range of the measurement error reported by most LiDAR manufacturers.

Setup for camera input: We've found experimentally that the camera feature extractor fails to learn generalizable geometric features unless spatial augmentations are applied. In our training, we performed random crop augmentations of [512, 256] pixels in width and height to the input image, and updated the camera intrinsic parameters accordingly.

A.2 Model setup

Both image and point cloud feature extractors in our model follow the U-Net [14] architecture with 5 layers of coarse-to-fine features, each layer being a factor of 2 finer than the previous layer. We use features from 3 layers for our alignment, the 1/16-scale, the 1/4-scale, and the 1-scale. The feature dimensionalities at these layers are 128, 128, and 32, respectively. To help adaptation, at each pyramid level, the features from both domains are passed through a shared 2-layer MLP with the same input and output dimensions at each layer and Leaky ReLu activation.

With recent advancements in 3-D convolutions [20], we've chosen the spconv [21] implementation of sparse 3-D CNNs for our LiDAR feature extractor, as it efficiently handles the large point clouds, and effectively treats the sparsity pattern of the data.

Learning features in the image domain is relatively straightforward, we've found that using the same U-Net [14] architecture (similar to Pixloc [11]) with the 2-D convolutional VGG [22] backbone was sufficient for learning image features.

We initialize the visual extractor using weights from a pre-trained PixLoc [11] model trained on the CMU Seasons dataset [23], and the LiDAR extractor using only the sparse 3D CNN weights of a SphereFormer [24] model pre-trained on Semantic KITTI [25]. For the pose optimization, we trained with $M = 5$ iterations at every pyramid level.