Text2Pos Supplementary Material

In this supplementary material, we provide additional information to further understand our proposed coarse-to-fine language-based localization model – Text2Pos. In Sec. 1, we provide more details about our *KITTI360Pose* dataset such as the query position sampling mechanism and the clustering and describing of scene object instances. Next, we describe implementation details about the cell database construction, data augmentation, model training and PointNet++ pre-training in Sec. 2. We further present thorough ablation experiments to study the impact of data variations on the localization performance in Sec. 3. There we also prove our concept of utilizing street names (when available) to better overcome the ambiguity in descriptions caused by the fact that many locations have semantically similar or equal surroundings. Finally, Sec. 4 shows qualitative results of top-k candidate cells retrieved during the coarse localization stage. We will release our code upon the paper's acceptance.

1. KITTI360Pose Dataset Details

Query position sampling and description. To obtain query positions, we first sample equidistant locations along the original KITTI360 [5] capturing paths. For each of these trajectory locations, we then sample a number of random nearby positions that are up to ½ cell-sizes away from it. We sample 4 positions for the baseline and present detailed studies on the impact of training our baseline on a larger dataset that samples more query positions (per trajectory location) in Tab. 3.

As detailed in the main document, a query position is described based on the set of its surrounding object instances, and thus we discard a sampled query position if there are insufficient instances, *i.e.* $N_h < 6$ in practice, in its vicinity. Given a query position with enough nearby instances, we can select the subset of the instances (6 instances in practice) to describe the query using three different strategies: (i) simply selecting the closest-by instances, (ii) choosing the instances that cover as many different directions (pointing towards the query position) as possible and (iii) picking instances with as many different classes as possible. If two or more of these strategies yield the same set of instances, the duplicates are omitted.

Object clustering. In order to also incorporate stuff class objects into our descriptions, it is necessary to cluster them into separate instances, *e.g.*, clustering one large *sidewalk* object covering the entire scene into separate instances left and right of a query position. Since we did not find a way to achieve such a clustering globally for a whole scene, we decide to cluster a stuff-class object locally for a given cell, where we first ignore those points of the complete stuff object that are not contained in that cell. For its points inside the cell, we then cluster them into multiple separate instances using the DBSCAN [2] and retain the clustered instances that contain enough points (250 in practice).

Dataset Statistics. As described above, we can obtain datasets of different scales by varying the number of sampled positions at each trajectory location. In Tab. 1, we show statistics about the number of query positions, descriptions and covered scene area of the two *Kitti360Pose* dataset versions generated by sampling 4 and 8 positions per trajectory location.

Challenges. Notice, the mapping between a query description and a target position is often not unique, since there exist many positions with semantically similar or even equal surroundings. Such ambiguous nature makes the task of text-based outdoor localization very challenging. Compared to the indoor environment which is rich in semantics of various objects such as table, sofa, chair and cup [1], outdoor scenes have fewer static objects, yet more complicated semantic cues that are mainly based on *stuff* classes, such as tree, vegetation, fence and road. Those *stuff*-based semantic features are highly repetitive for real-world large-scale outdoor scenes, which is also verified by our statistics in Tab. 1, which show that only a small fraction of the generated descriptions is actually unique. Therefore, our method is designed to endure such ambiguity by retrieving several candidate cells and then performing more refined pose estimations for each of them. In future work, we expect to reduce the ambiguity by incorporating the use of unique landmarks like street names or named buildings, which can be easily integrated into our general coarse-to-fine localization pipeline. We present our experiment in Tab. 4 as a proof of concept, where we manage to utilize simulated street names to improve our localization performance up to 20 percent points.

Split	# Scenes	Area $[km^2]$	# Positions	# Descriptions	# Unique Desc.
4 sampled position per trajectory location					
Training	5	11.59	9961	28807	601
Validation	1	2.25	1116	3187	416
Testing	3	2.14	3932	11505	518
All	9	15.98	15009	43499	629
8 sampled position per trajectory location					
Training	5	11.59	19688	57482	612
Validation	1	2.25	2224	6405	435
Testing	3	2.14	7747	22878	542
All	9	15.98	29659	86765	641

Table 1. *Kitti360Pose* dataset statistics

2. Implementation Details

Cell database. Our method relies on a database of cells to first retrieve top-k candidate cells which potentially contain our target position and then perform more accurate fine localization within those cells. To construct a database of cells that can fully cover the scene area, we use a sliding window with size $\mathcal{W}$ and sliding stride $\mathcal{S}$ to sample the cells along both the horizontal and vertical directions. We empirically fix the cell size to be $\mathcal{W} = 30m$ which usually covers enough instances for our experiments and we use a stride of $\mathcal{S} = 1/3 \times \mathcal{W}$. After the raw sampling, we further reject cells that have not enough objects inside to describe a position, *e.g.* a cell with mostly empty space. This leads to a database of $11259/1434/4308$ cells for the training/validation/testing scene split and in total $17,001$ cells for the whole dataset.

***In-cell* instances.** For fine localization, we cut-off or pad *in-cell* instances to keep the same number of instances per cell to allow mini-batching in the matching module. We consider each cell to contain 16 *in-cell* instances, *i.e.*, $N_p = 16$ for both training and inference, which has led to the most promising performance across different data settings in our experiments. In the case of too few instances, we pad dummy instances of 10 points which have black color and random point coordinates close to zero so that they are easily recognized by the model. In addition, we normalize the coordinates of *in-cell* instances w.r.t. the size of their belonging cell such that each coordinate value is $\in [0, 1]$, which aids the regressor's training stability.

Query description grounding. To learn the task of text-to-cell retrieval, we need to ground a query position description onto a ground-truth (GT) cell. We define a GT cell to be the cell in the database that contains the described position and whose center is closest to the described position. To learn the task of hint-to-instance matching, we need to further identify the GT correspondences between the query hints and the *in-cell* instances within a candidate cell. For *instance*-class instances, the hint-to-instance correspondences are established using GT instance IDs from the original KITTI360 dataset. Notice, *stuff*-class hint instances were clustered using a synthetic cell centered on the query position, which means the same instance might appear in a slightly different position within a retrieved cell since the query position can be anywhere inside that retrieval cell. Therefore, to match between a *stuff*-class hint instance and a point-cloud *in-cell* instance which are clustered w.r.t. a synthetic cell and a retrieval cell, we compare their semantic class IDs and the two direction vectors pointing from the query position to those two instances. We consider them as a match, if they have the same semantic class and their direction vectors are close enough.

Data augmentation. During training, we also use data augmentations for the point cloud instances and the pose descriptions to aid our model performance. In the *instance augmentation*, we randomly rotate the instance across the z-axis during training and normalize-scale all its points to a $[0, 1]$ interval during the training and inference. For the *description augmentation*, we (i) randomly shuffle the order of the hints that make up a query description and (ii) randomly flip cells horizontally and/or vertically by flipping the location of the pose and instance centers in each cell and changing the corresponding words in the description. The description augmentations are not used when training the refinement module.

Text2Pos model training. We train our model using five training scenes and use one scene for model validation. For coarse localization, we train the retrieval model using an Adam optimizer [3] with batch size 64 and learning rate 0.001 for at most 64 epochs. We set the margin parameter α in the pairwise ranking loss (Eq. (1)) as $\alpha = 0.35$. For fine localization, we train the matching module and regression jointly using an Adam optimizer with batch size 32 and learning rate 0.0003 for 16 epochs.

Stride	# Cells (val split)	Localization Recall ($\epsilon < 5/10/15m$) $k=1$	$k=5$	$k=10$
$\mathcal{S} = 3m$	15899	**0.22/0.35/0.41**	**0.41**/0.53/0.58	**0.52**/0.63/0.68
$\mathcal{S} = 5m$	5724	0.18/0.29/0.35	0.39/0.53/0.57	0.51/0.64/0.68
$\mathcal{S} = 10m$ (baseline)	1434	0.14/0.25/0.31	0.36/**0.55/0.61**	0.48/**0.68/0.74**
$\mathcal{S} = 15m$	629	0.09/0.19/0.25	0.25/0.46/0.54	0.35/0.61/0.70
$\mathcal{S} = 20m$	362	0.07/0.14/0.19	0.19/0.37/0.45	0.25/0.49/0.59

Table 2. Ablation on varying sampling stride for cell database construction.

Positions (per traj. loc)	Localization Recall ($\epsilon < 5/10/15m$) $k=1$	$k=5$	$k=10$
4 (Baseline)	0.14/0.25/0.31	0.37/0.54/0.60	0.48/0.68/0.73
8	0.16/0.28/0.33	0.39/0.57/0.63	0.52/0.70/0.75
12	0.16/0.29/0.35	0.41/0.60/0.66	0.52/0.72/0.77
16	0.16/0.28/0.34	0.39/0.56/0.62	0.51/0.70/0.75

Table 3. Training on larger dataset. We generate a larger dataset for training by varying the number of positions sampled at every trajectory location.

PointNet++ pretraining. We pre-train our PointNet++ [4] backbone for the point cloud classification on KITTI360 and use it to initialize the two instance encoders used in the coarse and fine localization stages. We aggregate the objects from all our database-side cells into a training set of $159,828$ objects from 22 classes and again use the Adam optimizer [3] with a learning rate of 0.003 and a batch size of 32.

3. More Ablation Studies

In the main document, we have presented several ablation studies that focus on analysing the localization performance of each component of our proposed Text2Pos model, *i.e.*, the coarse retrieval component and the fine matching component. In this section, we first complete the results of the cell stride ablation (that has been presented in Tab. 1 of the main paper). And we further provide an additional ablation to study the impact of training on a larger version of our dataset on the localization performance, to thoroughly understand our proposed method. Finally, we confirm our hypothesis that our localization performance can be further improved when provided with additional landmarks information such as street names.

Full results on sliding stride ablation. We variate the stride size for our database-side cells between values decreasing from $\mathcal{S} = 20m$ down to $3m$, while keeping a fixed cell size of $30m$. As shown in Tab. 2 where we mark the best recall with **bold**, our baseline setting $\mathcal{S} = 10m$ leads to the best performance at $10/15m$ error thresholds when using top-5/10 candidates. For $5m$ errors or top-1 candidates, the recall steadily increases for smaller strides, *e.g.*, from 7% up to 22% recall for a top-1 candidate prediction within a $5m$ error threshold. However, we consistently observe a larger performance drop (if any) when going from $\mathcal{S} = 15m$ to $10m$ than going from $\mathcal{S} = 10m$ to $5m$. While, $\mathcal{S} = 3m$ is the most promising one for the finest error threshold, it leads to less efficient computation in both memory and run time due to the large amount of generated cells (11 times more than in the $10m$ setting). For our current implementation, the clustered stuff instances are pre-computed and stored separately for each cell, meaning it takes approximately 9 GB of memory to store 15k cells (0.6MB per cell) just for the validation split. Another option to avoid big memory consumption is to perform instance clustering on-the-fly during training and inference, however, loading plus clustering of all cells in the validation split already takes around 50 minutes ($0.2s$ per cell). This also prevents us from pushing the stride to more extreme settings, *e.g.*, $\mathcal{S} = 1m$ where one needs to handle approximately $135k$ cells for the validation split. To maintain the feasibility of performing intensive evaluations, we choose our baseline setting as $\mathcal{S} = 10m$. This gives us the best trade-off between computational efficiency and localization recall, as we consider further technical engineering to improve the computational overhead coming from instance clustering out of the scope of this research, leaving it to future work.

Using a larger dataset. As described before, it is possible to increase our dataset scale by sampling more poses around each trajectory location. In our final experiment, we vary this scale by sampling 8, 12 and 16 poses per location and train our coarse and fine models on these larger datasets. To maintain the comparability, we evaluate models trained under varying settings on the same validation set that is obtained using the baseline configuration. As shown in Tab. 3, localization recall improves incrementally by training on a larger dataset, acquired by sampling poses from 4 to 12 positions per trajectory location. However, the performance starts to decrease on the 16-positions sampling configuration, which indicates a limit of performance improvement by sampling denser query positions. Our intuition is that the increase in sampling density leads to increases in the ambiguity of the mapping between a query description and a position, which starts to harm the learning of the global mapping on the cell level and the local mapping on the instance level. We note that we chose the sub-optimal setup with 4 poses per location as our experimental baseline due

Variant	Localization Recall ($\epsilon < 5/10/15m$)		
	$k = 1$	$k = 5$	$k = 10$
Text2Pos (coarse)	0.10/0.23/0.30	0.27/0.52/0.60	0.37/0.65/0.72
Text2Pos (coarse + fine)	0.14/0.25/0.31	0.37/0.54/0.60	0.48/0.68/0.73
Text2Pos + street names (coarse)	0.15/0.34/0.45	0.40/0.69/0.79	0.52/0.83/0.89
Text2Pos + street names (coarse + fine)	**0.19/0.36/0.46**	**0.47/0.72/0.80**	**0.61/0.85/0.90**

Table 4. Using street names as additional localization cues.

to time constraints, but will publish our results based using a more optimal sampling in our camera ready version.

Using street names as additional cues. As mentioned in the main paper as well as Sec. 1, our method can potentially use additional information such as nearby landmarks or street names to reduce the inherent cell ambiguity caused by places with semantically similar surroundings. To confirm this hypothesis we perform an additional experiment in which we use simulated street names as our additional cues to improve our localization recall. To simulate street names, we split our validation scene into nine separate streets as shown in Fig. 1 (each street in a unique color). All database-side cells and query-side poses are then assigned to their corresponding streets. During inference, we still perform coarse retrieval as usual, but additionally reject all cells from incorrect streets before taking the *top-k* retrievals. Hence, this experiment serves as a "semi-oracle" for the coarse retrieval. As Tab. 4 shows, using the simulated street names (as prior ground truth) boosts the performance of our baselines up to 20 percentage points. With this improvement, our full pipeline is able to localize 19% of the queries with top-1 retrieval within a strict error radius of $\epsilon = 5m$. If we consider all top-10 retrievals, it then manages to localize $61/85/90\%$ of the queries at different error thresholds of $5/10/15m$. Therefore, our experimental results indicate that additional text-based cues like street names are valuable information to complement the challenge of description ambiguity. We believe our proof of concept shows this direction is worthy of further research investigation to properly incorporate existing landmark information such as street names, zip codes or named buildings, into text-based localization pipelines for performance improvement.

4. Qualitative Text-to-Cell Retrieval Analysis

Finally, we show examples of the top-3 candidate cells retrieved by our text-to-cell retrieval model in Fig. 2. As we can see in both the successful (*left*) and failure cases (*right*), top-3 retrieved cells often exhibit high similarity in semantics to the query cell and to each other, yet can be more than $100m$ apart, which again highlights the inherent ambiguity of large-scale outdoor text-based localization. As a consequence, as shown in the 2nd successful example (*left*), the correct cell is retrieved but ranked lower (as the 3rd) than the other two candidates which are much further away. Furthermore, the 2nd failure example (*right*)) shows that the top-1 candidate is indeed a close-by candidate yet will not be considered as a correct one according to our manual $10m$ threshold, meaning the pose error is too coarse. This also suggests the need of another refinement step to improve the localization recall, which will turn such a case into a successful localization.

References

[1] 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

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996. 1

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 2, 3

[4] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017. 3

[5] Jun Xie, Martin Kiefel, Ming-Ting Sun, and Andreas Geiger. Semantic instance annotation of street scenes by 3d to 2d label transfer. In *CVPR*, 2016. 1

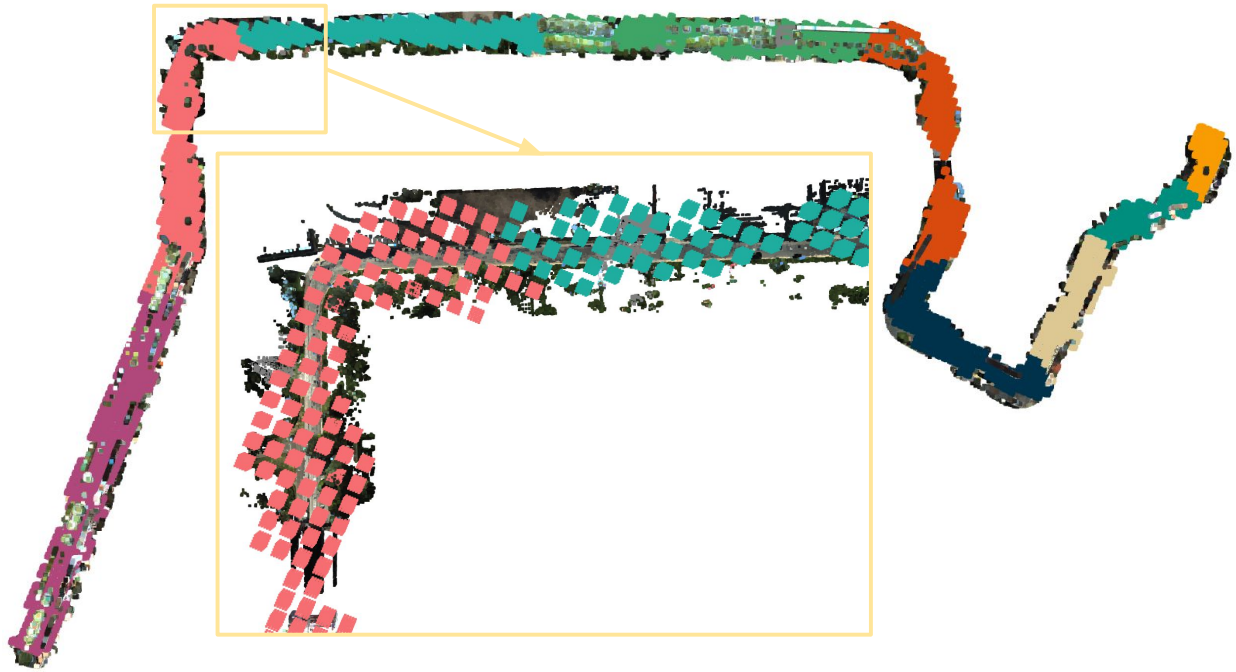

Figure 1. Simulated streets. *Overview:* We split our validation scenes into nine hypothetical streets marked with different colors. *Zoom-in:* Each colored square placed on top of a street represents a retrieval cell belonging to that street area.

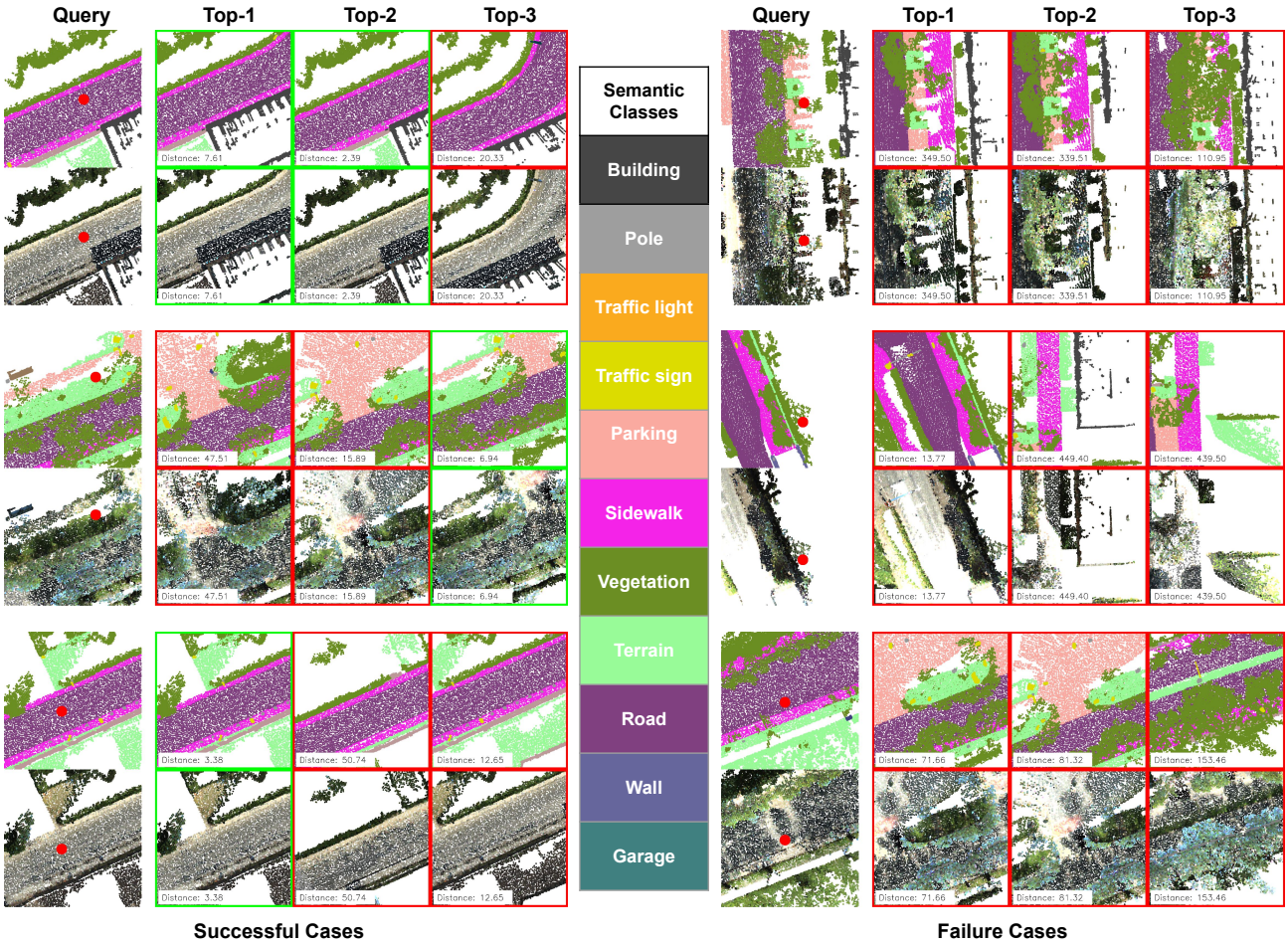

Figure 2. **Examples of the top-3 retrieved cells.** In the *left* part of the figure, we show 3 successful examples where the correct cell is within the top-3 candidates. In the *right* part, we show 3 failure cases where none of the top-3 candidates is the correct one. Each example consists of the dataset cell that is closest to the query pose (the 1st column) and the top-3 retrieval cells, where the $in-cell$ instances are colored by their semantic classes (the top row) and RGB values (the bottom row). We plot the mapping from semantic colors to class labels in the center of the figure. We further mark the correct cells with green borders and wrong cells with red borders, where we consider a cell to be correct if its center is at most $10m$ away from the query position (the red dot in the query cell).