Barcode: Global Binary Patterns for Fast Visual Inference

Lee, T.-Y.; Patil, S.; Ramalingam, S.; Taguchi, Y.; Benes, B.

TR2017-145 October 2017

Abstract

We present Barcode, a global binary descriptor for images captured from a vehicle-mounted camera with two applications: localization and turn classification. Barcode characterizes an image by encoding the distribution of vertical lines into a binary descriptor: in each vertical stripe of an image, if any vertical line exists the corresponding bit is set to 1, otherwise 0. For localization, our approach uses a database of geolocated images, each having its Barcode precomputed during a preprocessing stage. In the run time, we first generate the binary descriptor for each image and then use the descriptor to find the location in the database via Hamming distance metric. For turn classification, we train a deep neural network that uses a set of Barcodes from consecutive images to classify turns (left, right, straight, and stationary). We show that Barcode extraction can be done at 100-1000 Hz, localization at 10 kHz, and turn classification at 1 kHz. We show compelling experimental results on KITTI dataset and other sequences captured near Harvard and Purdue campuses.

International Conference on 3D Vision

Barcode: Global Binary Patterns for Fast Visual Inference

Teng-Yok Lee[1]* Sonali Patil[2]* Srikumar Ramalingam[3] Yuichi Taguchi[1] Bedrich Benes[2]

[1] Mitsubishi Electric Research Labs
{tlee,taguchi}@merl.com

[2] Purdue University
{patil19,bbenes}@purdue.edu

[3] The University of Utah
srikumar@cs.utah.edu

Abstract

We present Barcode, a global binary descriptor for images captured from a vehicle-mounted camera with two applications: localization and turn classification. Barcode characterizes an image by encoding the distribution of vertical lines into a binary descriptor: in each vertical stripe of an image, if any vertical line exists the corresponding bit is set to 1, otherwise 0. For localization, our approach uses a database of geolocated images, each having its Barcode precomputed during a preprocessing stage. In the run time, we first generate the binary descriptor for each image and then use the descriptor to find the location in the database via Hamming distance metric. For turn classification, we train a deep neural network that uses a set of Barcodes from consecutive images to classify turns (left, right, straight, and stationary). We show that Barcode extraction can be done at 100-1000 Hz, localization at 10 kHz, and turn classification at 1 kHz. We show compelling experimental results on KITTI dataset and other sequences captured near Harvard and Purdue campuses.

1. Introduction

Reporting a position of a moving car (localization) has been traditionally a task that is solved by Global Positioning Systems (GPS). However, many cars are also equipped with car-mounted cameras (see Figure 1) that are used for safety purposes or, as in the case of autonomous cars, for proximity detection and driving. We introduce a novel, minimal, but highly informative global image descriptor that we denote *Barcode* and we show how it can be used for image-based localization and turn-classification with almost negligible computational cost. We envision the use of Barcode for providing quick and high-level feedback in the context of autonomous driving applications, even with limited computational resources.

Image-based localization has been identified as a complementary method to GPS and used previously [7, 9, 29, 42]. By turn-classification, we refer to the problem of computing whether an on-road vehicle makes a left or right turn. This can be seen as a very special case of the problem of visual odometry where one computes six degree-of-freedom (DOF) camera motion (three for rotation and three for translation) from an image pair [33]. Both problems of localization and visual odometry have been studied in the context of computer vision and robotics communities and several real-time algorithms exist for solving such tasks. While the major focus is on accuracy and several benchmark datasets [1]

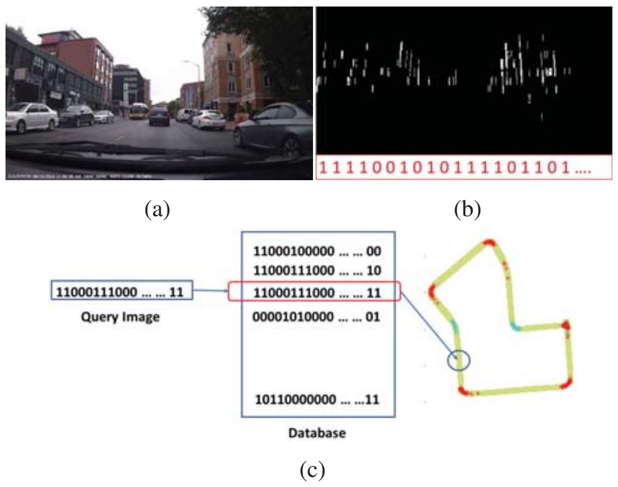


Figure 1: We propose a global binary descriptor Barcode for the tasks of localization and turn classification. (a) An image captured from a vehicle-mounted camera. (b) The binary descriptor describing the image using 128 bits. (c) Schematics of the localization and the turn-classification problems. For the localization, we find the best match for the query binary descriptor from a database of binary descriptors associated with 1000s of images. Based on the matching, we can identify the location on a driving trajectory as shown. We show the turn-classification results using color-coding: cyan denotes straight motion with no turns, red denotes right turns, and blue denotes left turns.

*Denotes joint first authorship.

[1] http://www.cvlibs.net/datasets/kitti/index.php

rank the existing methods, we would like to take a pause and ask the following question: By relaxing the requirement on high accuracy, can we develop algorithms that can reach very high speed (i.e., several orders of magnitude faster than existing methods)?

In this paper, we heavily rely on the prior that the images are captured from a vehicle-mounted camera, and exploit vertical lines that are typical in urban canyons with buildings, where GPS is known to be error-prone. We use histogram of vertical lines in adjacent locations to build a binary vector that encodes the presence and absence of vertical lines in different angular segments from a camera center. We show that this descriptor can be extracted at very high speed and it is highly informative to provide location information. Note that our method relies only on line segments, which are less sensitive to lighting variations.

We show two applications using this binary descriptor: localization and turn-classification. For localization, we compare the Barcode of the query image with those of the images in the database using Hamming distance. The location of the query image can be seen as approximately the location of the closest image in the database. For turn-classification we use a neural network where the input is a binary matrix obtained by stacking the Barcodes from consecutive image frames.

In summary, we claim the following contributions:

1. We propose Barcode, a novel binary image descriptor that is compact, highly informative, and can be efficiently extracted from an image at 100-1000 Hz.

2. We show that Barcode enables image-based localization by using simple Hamming distance. The localization approach can search a query image from a database of 20k images at 10 kHz.

3. We show a novel turn-classification algorithm that can use Barcodes from consecutive images using a neural network algorithm. The algorithm runs at 1 kHz on a CPU.

4. Compelling experimental results are shown for localization and turn-classification on challenging datasets like KITTI [8].

2. Related Work

In the last decades, the vision community has witnessed a wide variety of methods for image-based localization that share the common underlying idea that is to match a query image to some entity that can act as a fingerprint for a certain location. We can broadly classify the localization methods to be based on three entities: (1) image databases, (2) 3D models, and (3) other modalities such as satellite weather imagery and road maps.

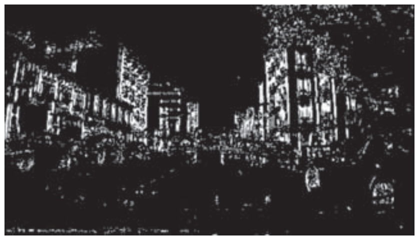
Image databases: One of the earlier methods used a database of GPS-tagged images of building facade and localized mobile camera images using wide-baseline matching [29]. Using SIFT descriptors, Zhang and Kosecha showed impressive results in the ICCV 2005 computer vision contest (”Where am I?”) [42]. Hays and Efros [9] took the problem of localization to global scale by using millions of GPS-tagged images from the web and matching the query image using a wide variety of image features such as color and texton histograms, line features, gist descriptor, and geometric context (labeling of the image pixels into sky, vertical and ground). The use of time-stamped photographs has also proven beneficial for geo-localization [13]. FABMAP is a large-scale localization method that uses local descriptors like SURF and showed localization over 1000 kilometers trajectory [6]. The bag-of-words representation that aggregates local descriptors into a global descriptor has also been a popular method for large-scale image search [25, 34, 18], and its extensions including the Fisher vector and Vector of Locally Aggregated Descriptors (VLAD) showed state-of-the-art performance [11].

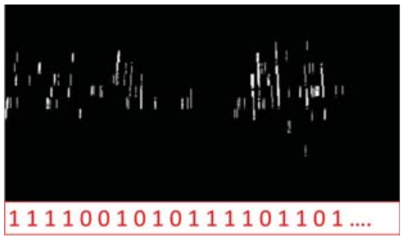
Conventional local descriptors such as SIFT and SURF had large computational requirement, and the advent of binary descriptors such as ORB [30] has led to substantial gains in matching speeds. In the case of binary feature descriptors, multi-index hashing (MIH) techniques enable efficient and exact K-nearest neighbor search in Hamming space [26]. We present a global binary descriptor and use MIH technique for finding the nearest neighbors in Hamming space. BRIEF-GIST [36] is a global binary descriptor that extracts a single BRIEF descriptor for an entire image after subsampling the image. Such global descriptors are extremely light-weight with a trade-off on the performance. In this work, we also propose a global binary descriptor that allows a better tuning of the parameter that controls performance and speed. While increasing the resolution of the binary descriptor, we allow better performance.

3D models: Several existing methods use 3D models and/or omni-directional cameras for geolocalization [15, 35, 20, 37, 41, 31, 19, 38]. Koch and Teller proposed a localization method using a known 3D model and a wide angle camera for indoor scenes by matching lines from the 3D model with the lines in images [15]. Moreover, skylines or horizon can also provide strong localization cues [2, 20, 28, 32]. By directly aligning the lines from query images to the ones in a line-based 3D model we can achieve localization [27, 21].

Satellite weather imagery and road maps: It was demonstrated that by correlating the satellite weather imagery at a specific time with time-stamped web camera images, we can also detect the location [10]. By correlating the local trajectories obtained from noisy visual odometry measurements with roadmaps, we can compute the location of a moving vehicle as shown by [1, 3]. These methods are sim-

(a) Input image (Cambridge).


(b) Thresholded difference between gradients in x and y.



(c) Window Search and Barcode (bottom).

Figure 2: Overview of Barcode. First we resize the input image to the desired resolution based on the number of bits in the Barcode. Then we compute $I_\triangle = |\frac{\partial I}{\partial x}| - |\frac{\partial I}{\partial y}|$ and binarize the image by thresholding. Finally we run a window search over each set of columns to find long vertical lines.

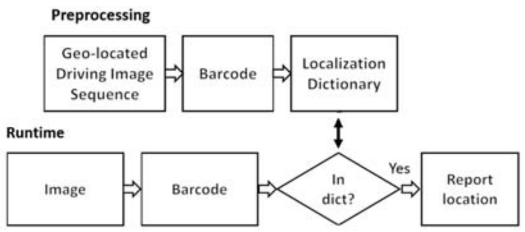


Figure 3: Localization method overview.

ple, but work at rather low speed of 15 Hz, and do not match the speed achievable by binary descriptors.

Our method uses image databases for locating the position and the idea of using vertical lines to build a descriptor for localization is not new. Lamon et al. [17] used a non-binary descriptor, based on vertical edges and associated colors, for localization of mobile robots in indoor scenes. They used string matching algorithms to exactly match the descriptors. However, such methods do not scale well for larger databases. Similarly Yuan et al. [40] used the same binary descriptor in the context of event sensors that only record changes in a scene.

Neural networks have been used for SLAM algorithms before. RATSLAM [22] is a classical SLAM algorithm that uses a neural network algorithm that uses local view cells to denote locations and pose cells to denote heading directions. The algorithm produces "very coarse" trajectory in comparison to existing SLAM techniques that employ filtering methods or bundle-adjustment machinery. Kendall *et al*. [14] presented PoseNet, a 23 layer deep convolutional neural network (CNN) used to compute the pose in a large-region at 200 Hz. While PoseNet employs CNN, we do not use any learning and solve the pose just by using Hamming distance. Recently, CNN has been used for solving the problem of visual odometry from stereo [16] and monocular sequences [23]. While the visual odometry results for the stereo case were promising, the monocular problem is still unresolved. In [23], it was concluded that the problem of visual odometry from monocular videos is very challenging for unknown scenes, and the work showed promising results for only known scenes. This makes us wonder whether CNN can solve the true 6 DOF visual odometry problem where the scene is typically unknown.

We show high-speed turn-classification using neural networks at 1 kHz. Note that PoseNet and turn-classification are two different problems. PoseNet is trained using the images from the exact same region where we estimate the pose of a query image. It takes one image as input and returns a 6 DOF pose. In the turn-classification, we use consecutive images (typically 12-20) to classify the turns. The training data for turn-classification need not come from the same region where we do the testing. The turn-classification can be seen as a simplified version of visual odometry [33], but several orders of magnitude faster than the existing methods.

3. Method overview

Barcode extraction (Section 4): Given an image, we use an efficient algorithm to extract a binary descriptor to summarize the contents of the image as shown in Figure 2. The basic idea is to compute the gradients in the image and look for peaks in every column of the image.

Localization (Section 5): The method works in two steps, preprocessing and runtime, as shown in Figure 3. The input to the *preprocessing* step is a sequence of images from a camera captured during a driving session. Each image is associated with location information (GPS coordinates). The image sequences are converted into the Barcodes and stored in a hierarchical data structure for fast indexing. During the *runtime* step, an image is converted into the Barcode and queried from the hierarchical data structure.

Turn-Classification (Section 6): We propose a CNN-based turn-classification algorithm that classifies the local vehicle trajectory into four classes: stationary, straight, left-turn, and right-turn. The input to the turn-classification algorithm is a stack (e.g., 12) of Barcodes from consecutive images.

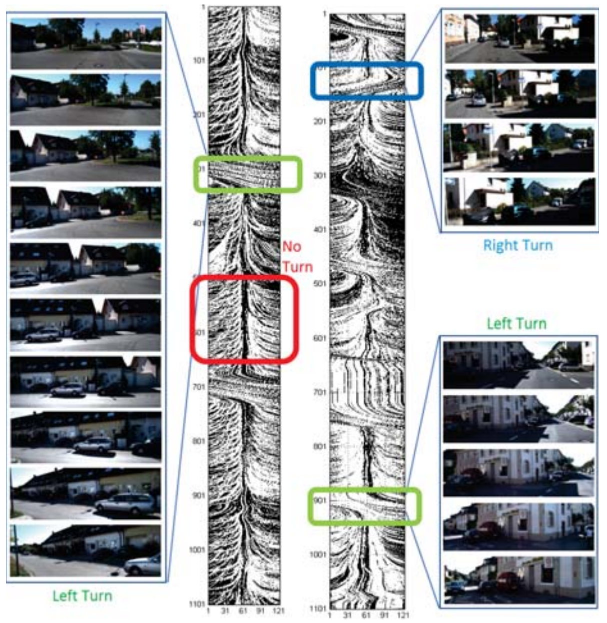


Figure 4: Spatio-temporal patterns for two driving sequences from KITTI dataset [8]: We stack Barcodes from individual frames of a driving sequence to form spatio-temporal images. Every row corresponds to a Barcode from a single image and the columns correspond to the temporal domain. The top row corresponds to the Barcode from the first frame. The turn categories (left turn, right turn, no turn) have specific visual signatures as shown.

4. Barcode extraction

Our binary descriptor is built using vertical lines in the image. Our definition of feature vector is similar to [1], with an exception that we use binary values instead of actual gradient values. Figure 2 summarizes the steps for computing the binary descriptor.

The Barcode is used either in a real-time session or during the preprocessing step in creating the dictionary. In the first step we compute two gradient images with respect to x and y as $\partial I/\partial x$ and $\partial I/\partial y$, and we subtract them to compute a difference image (see Figure 2b) $I_{\Delta} = |\frac{\partial I}{\partial x}| - |\frac{\partial I}{\partial y}|$. The difference image is then binarized by thresholding. For robustness to noise, we group w consecutive columns and generate W/w bit Barcode for the image having the width W. For each group of columns, we shift a window of $w \times h$ pixels along the vertical direction and count the number of nonzero pixels in each column within the window. If any of the columns within the window includes nonzero pixels more than a certain threshold, we set the corresponding bit 1, otherwise 0. The detected vertical lines using this window search, and the binary descriptor are shown in Figure 2c. This leads to a compressed image representation and each image is represented by a small binary descriptor (e.g., 128 bits). Our method is significantly faster (100-1000 Hz) than generic line extraction methods such as LSD line detection [39].

5. Feature vector matching for localization

Barcodes can be matched or compared using Hamming distance. Let S and T denote the binary descriptors corresponding to two images I_1 and I_2 respectively. The Hamming distance between the two binary descriptors is given by $H(S,T) = \sum_{i=1}^{n} S_i \otimes T_i$, where $\otimes$ is an XOR operator that returns 1 when the bits are different and 0 otherwise. To efficiently match a query image with a database of images, we use multi-index hashing (MIH) [26]. The key idea of MIH is to find a superset of nearest neighbors to reduce the search space. This can be achieved by partitioning the input binary descriptor into p disjoint descriptors and building p hash-tables to search the nearest neighbors for each sub-descriptor, such that the source and target descriptors S and T should be within the hamming radius r to be called as nearest neighbors.

6. Turn classification

6.1. Spatio-temporal patterns

In a driving sequence, we can extract Barcode from every frame. In Figure 4, we visualize the Barcodes from a video in spatio-temporal domain where the x-axis corresponds to the dimension (say 128) of each Barcode, and the y-axis corresponds to the time. As shown in Figure 4, we can easily spot the visual pattern or shape for different turn classes (left turn, right turn, and no turn). In general, whenever the car takes a left turn, the lines move to the right and vice versa. When the vehicle moves in a straight line, then the lines start to move away from the center. The different visual signatures can be explained using epipolar geometry.

6.2. Geometrical interpretation of spatio-temporal patterns

Given a pair of images with known camera motion, the point correspondences satisfy the classical epipolar geometry. In other words, every point on the first image lies on its corresponding epipolar line on the second image. Epipole is the point where all the epipolar lines meet. In other words, the image points move along the line joining the point and the epipole. In a general 6 DOF camera motion, the epipole can be anywhere in the image or even at infinity. However, the position of the epipole is well-constrained for image sequences obtained from a car-mounted camera.

As shown in Figure 5, the epipoles move only in the same horizontal line in the images captured from a moving car on a plane. When the car moves in a straight line, the epipole is in the middle. When the car turns left, the epipole moves to the left. We show the displacement of two line segments a and b in two consecutive frames. The

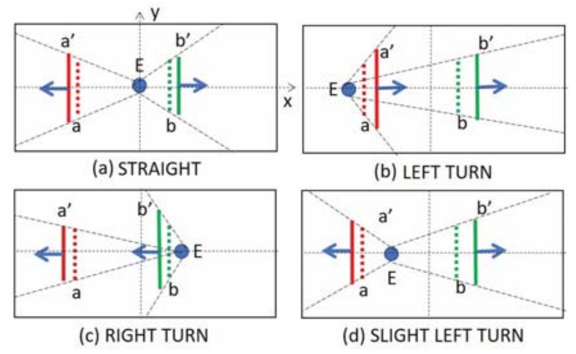


Figure 5: Let a and b denote two lines in one image in a video sequence. Let a' and b' denote their corresponding locations in the next image. We show the locations of lines in the first frame as dotted lines, and those in the second frame as bold lines. Let E denote the location of the epipole. We show the epipoles and the directions in which the lines move in four different motions: straight, left turn, right turn, and slight left turn. Note that the lines do not move in the same direction for the categories left turn and slight left turn.

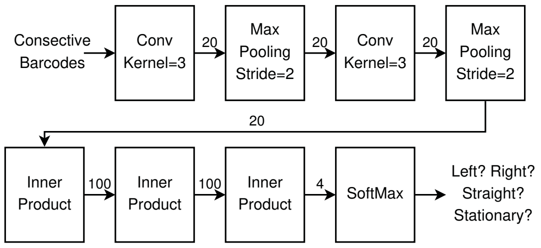


Figure 6: Network architecture for the turn-classification. The input data consists of concatenation of a few Barcodes from consecutive image frames. The arrows after convolution and pooling layers indicate the numbers of feature maps. The numbers after the inner product layers indicate the number of output. The final output is one of the four turn-classes (stationary, straight, left-turn, and right-turn). Each convolution and inner product layer is followed by a ReLU layer, which is hidden in this figure.

line segments move to the right since the end points of the line segments have to move along their associated epipolar lines. Similarly we can observe the movement of the line segments for different turn categories. The line segments move in different directions for left and right turns. It is important to note that the velocity and direction of motions of the line segments depend on their 3D geometry in the scene and the turn directions of the car. For example, on a slight left turn the lines may move in opposite direction compared to regular left turn as shown in Figure 5.

6.3. Turn-classification using deep neural network

It is possible to track the line segments in two consecutive images to understand the nature of the turns. A traditional vision or robotics approach would be to detect and match line segments and extract motion from the frames. Most approaches that employ such techniques would work at a few frames per second and involve heavy computational components such as line detection and line matching. We propose a novel turn-classification algorithm using a deep neural network that provides fast feedback on the nature of the turns. In particular, we show that it is possible to classify the turns at more than 1 kHz.

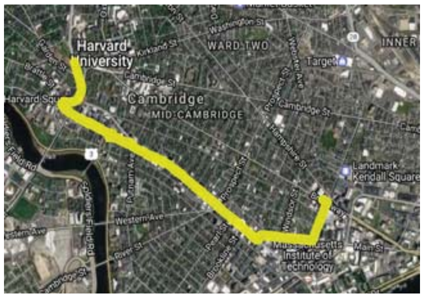
Training data: The input to the network is a sequence of Barcodes from a few consecutive frames. For example, we can consider 10 Barcodes from 10 consecutive images to form a binary image of size 128×10, where 128 denotes the dimension of each Barcode. The output labels are the turn classes. We assume that we are given the ground truth camera poses for each image in the video. Using the actual camera poses, we can compute the turn angles for the associated frames. In the simplest case, we can consider four turn classes: stationary, straight motion, left turn, and right turn. The stationary class denotes the case where there is no motion, especially when the car is at a traffic intersection. The straight motion class denotes the case where the turn angle lies in between α and $-\alpha$ (e.g., $\alpha = 5$) degrees. Using the ground truth pose values, we can compute the different output labels based on the turn-angles.

Network architecture: Figure 6 shows the network architecture for the turn-classification algorithm with four output classes (stationary, straight, left-turn and right-turn). The input data is passed to two convolution layers. The parameter settings for convolution and pooling layers are shown. Following the two convolution layers, we use three fully connected layers with different number of outputs. In the final layer we use soft-max to obtain the classification for turns.

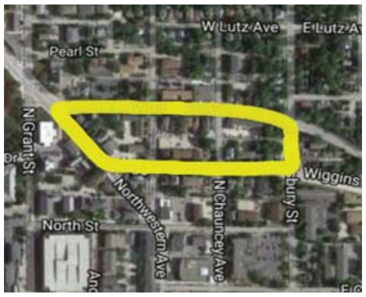
7. Results

7.1. Localization

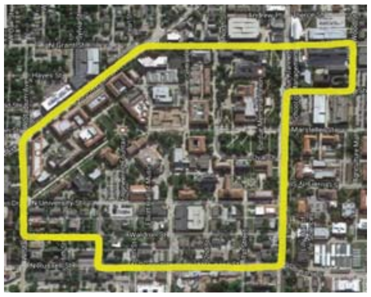
Datasets: We tested our localization algorithm on four urban datasets with man-made buildings, because our method relies on vertical lines from such structures. One of the datasets is the *KAIST* west campus sequence, which was captured within the campus of Korea Advanced Institute of Science and Technology (KAIST) and released by Choi *et al.* [5]. Three other datasets were collected using Garmin Dashcam sensors. The sequences *Harvard*, *Purdue1*, and *Purdue2* were captured near Harvard and Purdue universities as shown in Figure 7. We extract Barcodes from these sequences and the Hamming distance is computed between Barcodes from two different sequences following the same trajectories. Our results were measured on a Windows 7 desktop with Intel(R) Core(TM) i7-4770 CPU, NVIDIA GTX 1080 GPU, and 32GB of system memory.



(a) *Harvard*



(b) *Purdue1*



(c) *Purdue2*

Figure 7: The driving trajectories of our three sequences.

Table 1: Sequences for localization, parameters, and average construction time (build) and query time (query) per image of testing feature descriptors. All times are measured in milliseconds. The query time is for 50 nearest neighbors, the largest numbers tested in our experiments. Next to the query times for Barcode and BRIEF-GIST are the numbers of hash tables.

Dataset		SURF-VLAD			Barcode			BRIEF-GIST	
#images	Image size	Build	Query	Bit Length	Window height	Build	Query	Build	Query
Harvard	1280 × 720	518.93	0.51	256	15	6.65	8.41 (8) 0.09 (16)	0.55	0.95 (8) 0.04 (16)
DB: 19587	640 × 360	147.99	1.17	128	15	1.81	0.06 (8)	0.53	0.01 (8)
Query: 21446	320 × 180	31.97	0.31	64	8	1.02	0.02 (8)	N/A	N/A
Purdue1	1280 × 720	485.31	0.18	256	13	5.81	15.70 (8) 0.05 (16)	0.53	0.10 (8) 0.02 (16)
DB: 6929	640 × 360	116.98	0.24	128	11	1.77	0.02 (8)	0.52	0.01 (8)
Query: 6929	320 × 180	28.826	0.16	64	8	0.99	0.01 (8)	N/A	N/A
Purdue2	1280 × 720	247.98	0.35	256	13	6.14	11.46 (8) 0.12 (16)	0.55	0.43 (8) 0.06 (16)
DB: 27716	640 × 360	106.68	0.52	128	11	1.98	0.03 (8)	0.56	0.02 (8)
Query: 16558	320 × 180	27.01	0.17	64	8	1.04	0.03 (8)	N/A	N/A
KAIST	1280 × 960	480.45	0.22	256	18	8.50	9.97 (8) 0.10 (16)	0.53	0.54 (8) 0.02 (16)
DB: 16269	640 × 480	135.11	0.30	128	13	2.84	0.02 (8)	0.57	0.01 (8)
Query: 11035	320 × 240	38.54	0.17	64	8	1.23	0.02 (8)	N/A	N/A

Parameters: The parameters to build Barcodes include the window width and height to detect edges, and the number of bits in the Barcode. In our experiments, we used the window width of 5 columns and the window height was set as a fraction ($\frac{1}{8}$ to $\frac{1}{18}$) of the image height. Within each window, we counted pixels with high response per column. If any column had more than 80% of pixels with high response, the corresponding bit in the Barcode was set to 1. Table 1 lists the detailed parameters of all sequences.

We used two parameters during the process of localization. The first parameter denotes the number of nearest Barcodes that we look for, and the second parameter is a distance threshold (30 meters in our experiments). We see if the distance between the query Barcode and any of these K nearest Barcodes is less than the distance threshold. If we find at least one such Barcode, we treat the retrieval process a success.

Comparison: We compare Barcode with two algorithms. One is *SURF-VLAD*, namely VLAD [11] with SURF as the local descriptor. We used 32 clusters and aggregated the 64-dimensional SURF descriptors into a 2048 dimensional VLAD descriptor for each image. To accelerate the query of VLAD, we used Fast Library for Approximate Nearest Neighbors (FLANN) [24]. The other is *BRIEF-GIST* [36], which downsamples an image into a small icon and computes a single BRIEF descriptor [4] from the icon. In our experiments, we downsampled an image into 60×60 pixels. Both SURF and BRIEF were based on the implementations of OpenCV. Note that because the OpenCV implementation of BRIEF can support 128, 256, and 512 bits, we only evaluate the performance of BRIEF-GIST with 128 and 256 bits. Similar to Barcode, we used MIH to accelerate the query of BRIEF-GIST. Our MIH implementation is based on publicly available source code [26].

Speed: Table 1 includes the time to build all feature descriptors under different image resolutions. Here the time to resize the images is excluded. In such a case, BRIEF-GIST is the fastest one to construct. The speed of Barcode is related to image size because the time to detect and count the vertical edges is proportional to the number of columns. In contrast, SURF-VLAD is the most time-consuming one to construct. Even for the smallest image size, its frame rate is only 30 Hz, while the Barcode construction can be achieved at about 1000 Hz.

Table 1 also includes the query time for 50 nearest neighbors, the largest number for our experiments. These provide upper bounds of the query time of different feature descriptors. While the query time is proportional to the image size, when using 8 hash tables, the query performance of 256 bits for both Barcode and BRIEF-GIST was much slower than using 64 or 128 bits. A work around was using more hash tables. By using 16 hash tables, the query performance of both was apparently reduced to shorter than 1 millisecond.

It could be seen that the query time of SURF-VLAD can be close to the Barcode and BRIEF-GIST, despite the use of floating-point arithmetic. We believe that this could be due to the fact that FLANN only provides approximate K-nearest-neighbors, whereas MIH provides an exact solution for the same problem with Hamming distance. Nevertheless, SURF-VLAD is much slower to compute, and thus the overall query time is still 10 - 20 times slower than Barcode and 100 times slower than BRIEF-GIST.

Accuracies: Figure 8 shows the localization accuracy of Barcode, SURF-VLAD and BRIEF-GIST. In general, the use of higher resolution or longer feature descriptors leads to better performance. While SURF-VLAD achieves the highest accuracy for all cases, when using 50 nearest neighbors, the difference between Barcode and SURF-VLAD can be small. For the Harvard sequence, the performance of BRIEF-GIST is much lower than the others even with 50 nearest neighbors. With the Purdue2 sequence, Barcode can outperform BRIEF-GIST while considering more nearest neighbors.

Robustness: Our supplementary material lists the accuracies under different parameter values, including the number of nearest neighbors and windows height. We also measured the accuracies under 10, 20, and 40 meters as the distance thresholds. The result shows that the distance threshold can impact not only Barcodes but also SURF-VLAD and BRIEF-GIST. Also, under all testing distance thresholds, the relationship among the feature descriptors was similar.

In summary, Barcode is slightly inferior in accuracy to one of the state-of-the-art localization algorithms like SURF-VLAD, but significantly faster (100 Hz for high resolutions and 800-1000 Hz for low resolutions).

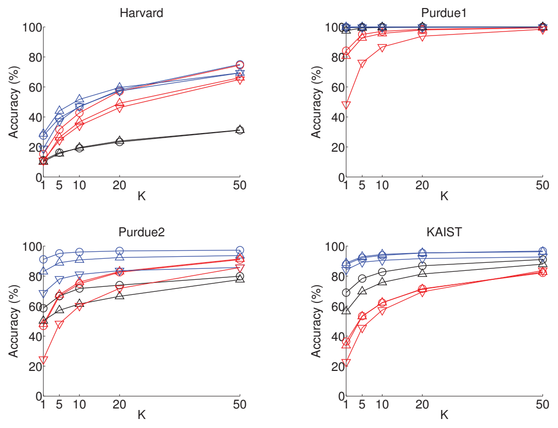


Figure 8: Localization accuracy of Barcode (Red), SURF-VLAD (Blue), and BRIEF-GIST (Black). For each feature descriptor, the lengths are represented by the line markers. The line markers from longest ones to shortest ones are represented by ◯, △, and ▽, respectively. The X axis is for K, the numbers of nearest neighbors.

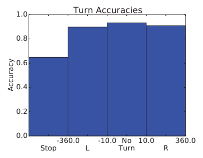
7.2. Turn classification

Experiment settings: We used the deep learning platform Caffe [12] to build our turn classifier. The parameters such as learning rate, momentum, weight-decay, and the number of iterations were set to 0.0001, 0.9, 0.0005, and 50000, respectively. We used KITTI [8] visual odometry sequences for evaluating the turn-classification results. The dataset comes with ground truth poses that is essential for generating training data with turn class labels. We chose 4 of the 10 sequences that predominantly have urban city scenes. Two of these sequences (sequences 7 and 8) were used for training, which have 5170 frames, and we tested on the remaining 2 sequences (sequences 5 and 6), which have 3860 frames. The Barcode of each image was extracted using a single window from rows 80 to 120 of the KITTI images because these rows cover the buildings. The length of Barcode was 128 for turn classification. For each bit, if any pixel in the corresponding image window had high response to vertical edges, its value was set to 1. Given the number of consecutive Barcodes to evaluate, says n, we iterated each training frame and used the Barcodes of this frame and the following $n-1$ ones to form a data item to train the network. Similarly, our testing procedure iterated each testing frame. The Barcodes of the iterated frame and the following $n-1$ ones were tested to measure the accuracies.

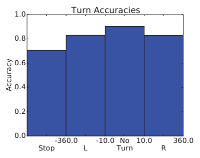
Accuracies: Figure 9 lists the class-wise classification accuracy of KITTI odometry sequence 5 with different parameters, whereas the supplementary material includes the predicted results of all training and testing sequences. The columns from left to right represent the accuracies with 12, 16, and 20 consecutive Barcodes, respectively. By comparing the bars charts from left to right for each row, we can see that 12 Barcodes leads to best performance compared to the others in the same row. When the angle threshold is 10 degrees, we achieve the best performance (higher than 80% for left, right, and no turn), as shown in the top row of Figure 9. With too many Barcodes, the performance can degrade. This could be due to the fact that while using too many Barcodes, the underlying local motion cannot strictly be one of the four classes (stationary, straight, left-turn, and right-turn). The top three rows of Figure 9 present the accuracies with angle thresholds of 10.0, 5.0, and 2.5, from top to bottom. It can be seen that the accuracy of the third row (α is 2.5) is apparently smaller than rows 1 and 2.

Network architecture: Another factor is the design of CNN networks. From the network in Figure 6, we removed the convolution layers and re-trained the network. Here we set the angle threshold as 10 degrees, the optimal one in the previous experiments. The bottom row of Figure 9 shows the accuracies without the convolution layers. For each subfigure in this bottom row, the performance is much lower than the other ones in the same column. Thus the convolution layers are necessary to capture the turn-signatures.

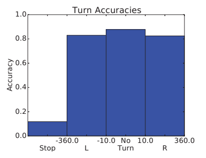
Training and testing speed: Our turn classifiers were tested on a Ubuntu 14 desktop with the same CPU and GPU as those in Section 7.1 except smaller system memory (16GB). Our training was executed on the GPU, which took less than 7 minutes with the convolution layers. In the absence of convolution layers, the training time was around 2 minutes. The testing can be done at 700-1300 Hz on CPU with the network with convolution layers. Without convolution layers, the testing can be done faster than 2 kHz.



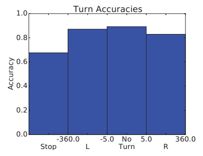
(a) $\alpha = 10, n = 12$



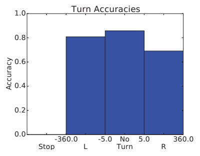
(b) $\alpha = 10, n = 16$



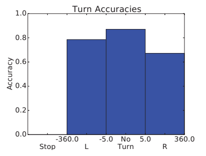
(c) $\alpha = 10, n = 20$



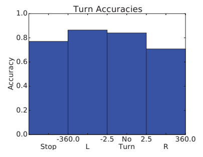
(d) $\alpha = 5.0, n = 12$



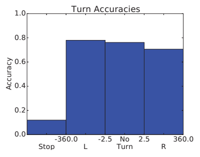
(e) $\alpha = 5.0, n = 16$



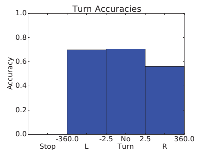
(f) $\alpha = 5.0, n = 20$



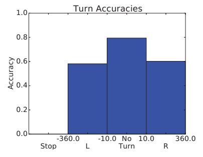
(g) $\alpha = 2.5, n = 12$



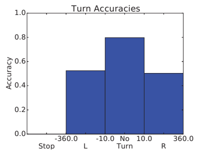
(h) $\alpha = 2.5, n = 16$



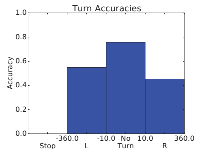
(i) $\alpha = 2.5, n = 20$



(j) $\alpha = 10, n = 12$, no convolution



(k) $\alpha = 10, n = 16$, no convolution



(l) $\alpha = 10, n = 20$, no convolution

Figure 9: Class-wise accuracies for the turn-classification network in Figure 6 of KITTI odometry sequence 5. α represents the angle threshold in degrees that determines the different turn classes. n denotes the number of Barcodes used in constructing the input data for the network. Note that the bottom row shows the result of the same network but without the convolution layers.

8. Discussion

We have shown a simple global binary descriptor that can perform high speed localization and turn-classification. Although the localization accuracy is not as high as the state-of-the-art methods like VLAD [11], the method is several orders of magnitude faster (100-1000 Hz). We did not use any video or road map priors during localization, and such techniques can further improve the accuracy. We observed that the turn-classification achieves about 80% accuracies in certain parameter settings and runs at 1 kHz. As shown in Figure 5, the location of the epipoles is correlated with the turn-classes. We believe that these light-weight turn-classifiers can be used to improve the computation time of other tasks like 6 DOF visual odometry, or provide motion priors for enhanced pedestrian and vehicle detection algorithms.

Our turn-classifier can be complementary to IMUs. While images are prone to errors in bad-weather conditions, IMUs suffer from drift issues. One interesting future direction would be a careful comparison of IMUs and images, and possibly their fusion to build a robust and fast turn-classifier. In some situations, the camera may have a large rotation with respect to the vertical direction and this may degrade the Barcode features. However, we can easily address this by pre-calibrating the camera with respect to the ground plane or by just using an IMU for fixing the rotation offset [40].

The proposed method relies on vertical lines and are only suitable for urban scenes with man-made buildings. We believe that in the absence of such buildings, GPS might already provide reasonable location estimates. In the future, we would like to explore the expressivity of our operator when considering different line orientations, and extend the use cases from urban scenes to other environments.

Acknowledgments

This work was supported by Mitsubishi Electric Research Labs (MERL). We thank Hao Kang for the data collection, Alan Sullivan and Chen Feng for useful discussions, and the anonymous reviewers for valuable feedback.

References

[1] H. Badino, D. Huber, Y. Park, and T. Kanade. Real-time topometric localization. In *ICRA 2012: Proceedings of IEEE International Conference on Robotics and Automation*, 2012. 2, 4

[2] M. Bansal and K. Daniilidis. Geometric urban geo-localization. In *CVPR 2014: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 2

[3] M. A. Brubaker, A. Geiger, and R. Urtasun. Lost! leveraging the crowd for probabilistic visual self-localization. In *CVPR 2013: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3057–3064, 2013. 2

[4] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *ECCV 2010: Proceedings of European Conference on Computer Vision*, 2010. 6

[5] Y. Choi, N. Kim, K. Park, S. Hwang, J. Yoon, and I. Kweon. All-day visual place recognition: Benchmark dataset and baseline. In *CVPR 2015 Workshop on Visual Place Recognition in Changing Environments*, pages 8–10, 2015. 5

[6] M. Cummins and P. Newman. Appearance-only slam at large scale with fab-map 2.0. *International Journal of Robotics Research*, 30(9):1100–1123, 2011. 2

[7] E. Garcia-Fidalgo and A. Ortiz. Vision-based topological mapping and localization methods: A survey. *Robotics and Autonomous Systems*, 64:1–20, 2015. 1

[8] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR 2012: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012. 2, 4, 7

[9] J. Hays and A. Efros. Im2gps: estimating geographic images from single images. In *CVPR 2008: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008. 1, 2

[10] N. Jacobs, S. Satkin, N. Roman, R. Speyer, and R. Pless. Geolocating static cameras. In *ICCV 2007: Proceedings of International Conference on Computer Vision*, 2007. 2

[11] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, Sept. 2012. 2, 6, 8

[12] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 7

[13] E. Kalogerakis, O. Vesselova, J. Hays, A. Efros, and A. Hertzmann. Image sequence geolocation with human travel priors. In *ICCV 2009: Proceedings of International Conference on Computer Vision*, 2009. 2

[14] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV 2015: Proceedings of International Conference on Computer Vision*, 2015. 3

[15] O. Koch and S. Teller. Wide-area egomotion estimation from known 3d structure. In *CVPR 2007: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007. 2

[16] K. R. Konda and R. Memisevic. Learning visual odometry with a convolutional network. In *VISAPP 2015: International Conference on Computer Vision Theory and Applications*, pages 486–490, 2015. 3

[17] P. Lamon, I. Nourbakhsh, B. Jensen, and R. Siegwart. Deriving and matching image fingerprint sequences for mobile robot localization. In *ICRA 2001: Proceedings of IEEE International Conference on Robotics and Automation*, 2001. 3

[18] J. Lee, S. Lee, G. Zhang, J. Lim, and I. S. W.K. Chung. Outdoor place recognition in urban environments using straight lines. In *ICRA 2014: Proceedings of IEEE International Conference on Robotics and Automation*, 2014. 2

[19] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua. Worldwide pose estimation using 3d point clouds. In *ECCV 2012: Proceedings of European Conference on Computer Vision*, 2012. 2

[20] J. Meguro, T. Murata, H. Nishimura, Y. Amano, T. Hasizume, and J. Takiguchi. Development of positioning technique using omni-directional ir camera and aerial survey data. In *Advanced Intelligent Mechatronics*, 2007. 2

[21] B. Micusik and H. Wildenauer. Descriptor free visual indoor localization with line segments. In *CVPR 2015: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015. 2

[22] M. J. Milford, G. Wyeth, and D. Prasser. Ratslam: A hippocampal model for simultaneous localization and mapping. In *ICRA 2004: Proceedings of IEEE International Conference on Robotics and Automation*, 2004. 3

[23] V. Mohanty, S. Agrawal, S. Datta, A. Ghosh, V. D. Sharma, and D. Chakravarty. Deepvo: A deep learning approach for monocular visual odometry. *arXiv preprint arXiv:1611.06069*, 2016. 3

[24] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP 2009: International Conference on Computer Vision Theory and Applications*, 2009. 6

[25] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR 2006: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2006. 2

[26] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR 2012: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3108–3115, 2012. 2, 4, 6

[27] S. Ramalingam, S. Bouaziz, and P. Sturm. Pose estimation using both points and lines for geo-localization. In *ICRA 2011: Proceedings of IEEE International Conference on Robotics and Automation*, 2011. 2

[28] S. Ramalingam, S. Bouaziz, P. Sturm, and M. Brand. Localization in urban canyons using omni-skylines. In *IROS 2010: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. 2

[29] D. Robertson and R. Cipolla. An image-based system for urban navigation. In *BMVC 2004: Proceedings of British Machine Vision Conference*, 2004. 1, 2

[30] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to sift or surf. In *ICCV 2011: Proceedings of International Conference on Computer Vision*, 2011. 2

[31] T. Sattler, B. Leibe, and L. Kobbelt. Improving image-based localization by active correspondence search. In *ECCV 2012: Proceedings of European Conference on Computer Vision*, 2012. 2

[32] O. Saurer, G. Baatz, K. Koeser, L. Ladicky, and M. Pollefeys. Image based geo-localization in the alps. *International Journal of Computer Vision*, 2015. 2

[33] D. Scaramuzza, F. Fraundorfer, and R. Siegwart. Real-time monocular visual odometry for on-road vehicles with 1-point ransac. In *ICRA 2009: Proceedings of IEEE International Conference on Robotics and Automation*, pages 4293–4299, May 2009. 1, 3

[34] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *CVPR 2007: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, 2007. 2

[35] F. Stein and G. Medioni. Map-based localization using the panoramic horizon. In *IEEE Transactions on Robotics and Automation*, 1995. 2

[36] N. Sunderhauf and P. Protzel. Brief-gist closing the loop by simple means. In *IROS 2011: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011. 2, 6

[37] J. Tardif, Y. Pavlidis, and K. Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. In *IROS 2008: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008. 2

[38] A. Torii, J. Sivic, T. Pajdla, and M. Okutomi. Visual place recognition with repetitive structures. In *CVPR 2013: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2013. 2

[39] R. G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. Lsd: a line segment detector. *Image Processing On Line*, 2:35–55, 2012. 4

[40] W. Yuan and S. Ramalingam. Fast localization and tracking using event sensors. In *ICRA 2016: Proceedings of IEEE International Conference on Robotics and Automation*, 2016. 3, 8

[41] B. Zeisl, T. Sattler, and M. Pollefeys. Camera pose voting for large-scale image-based localization. In *ICCV 2015: Proceedings of International Conference on Computer Vision*, 2015. 2

[42] W. Zhang and J. Kosecka. Image based localization in urban environments. In *3DPVT 2006: Proceedings of International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 33–40, 2006. 1, 2

Barcode: Global Binary Patterns for Fast Visual Inference

Supplementary Materials

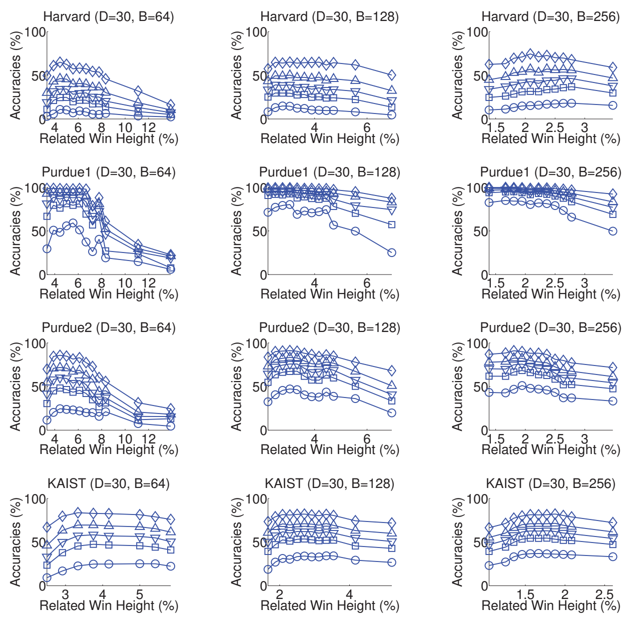
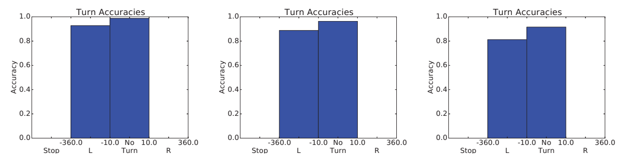
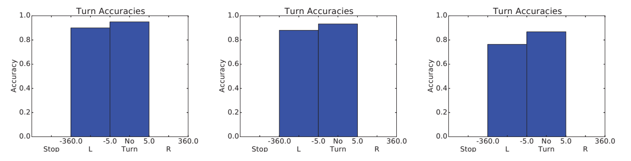


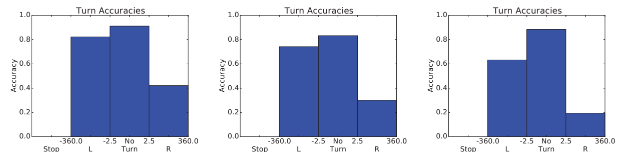
Figure 1: Localization accuracies of Barcodes under different window heights. Each curve represents a number of nearest neighbors, as indicated by the markers: 1 (◯), 5 (□), 10 (▽), 20 (△), and 50 (◇).



(a) $\alpha = 10, n = 12$ (b) $\alpha = 10, n = 16$ (c) $\alpha = 10, n = 20$



(d) $\alpha = 5.0, n = 12$ (e) $\alpha = 5.0, n = 16$ (f) $\alpha = 5.0, n = 20$



(g) $\alpha = 2.5, n = 12$ (h) $\alpha = 2.5, n = 16$ (i) $\alpha = 2.5, n = 20$

Figure 2: Class-wise turn classification accuracies of KITTI odometry sequence 6. α represents the angle threshold in degrees that determines the different turn classes. n denotes the number of Barcodes used in constructing the input data for the network.

1. Localization

1.1. Sensitivity of Barcodes

Figure 1 lists the localization accuracies of Barcodes under different parameters. The columns from left to right are for 64, 128, and 256 bits, respectively. Each chart lists the accuracies under different window heights and nearest neighbors. In our submission, the window heights in Table 1 are those with highest accuracies with 50 nearest neighbors. Figure 1 shows that the accuracies vary little near that point with highest values when considering more than 1 nearest neighbor. Nevertheless, if the windows height is too large, it could be difficult to detect vertical edges and thus the accuracies can drop. This is especially obvious for 64-bits of Barcodes, which are for smaller images and thus a pixel can represent larger area than longer Barcodes.

1.2. Impact of distance thresholds

Figure 3 lists the localization accuracies under different distance thresholds. As our main paper uses 30 meters as the threshold to measure the accuracies, we also tested 10, 20, and 40 meters. The curves shows that although the distance threshold can impact the accuracies of Barcodes, it also impacts VLAD and BRIEF-GIST. When reducing the distance threshold to 10, the dropping of accuracies is especially apparent for sequences Harvard and Purdue2 and when only 1 nearest neighbor is considered. Also, the relationship among the feature descriptors is similar under all testing thresholds.

1.3. Testing video sequences

Figures 4 - 11 list the frames of the driving sequences to evaluate localization accuracies. The caption of each frame describes the corresponding frame index. For sequences Harvard, Purdue1, and Purdue2, the frames are uniformly

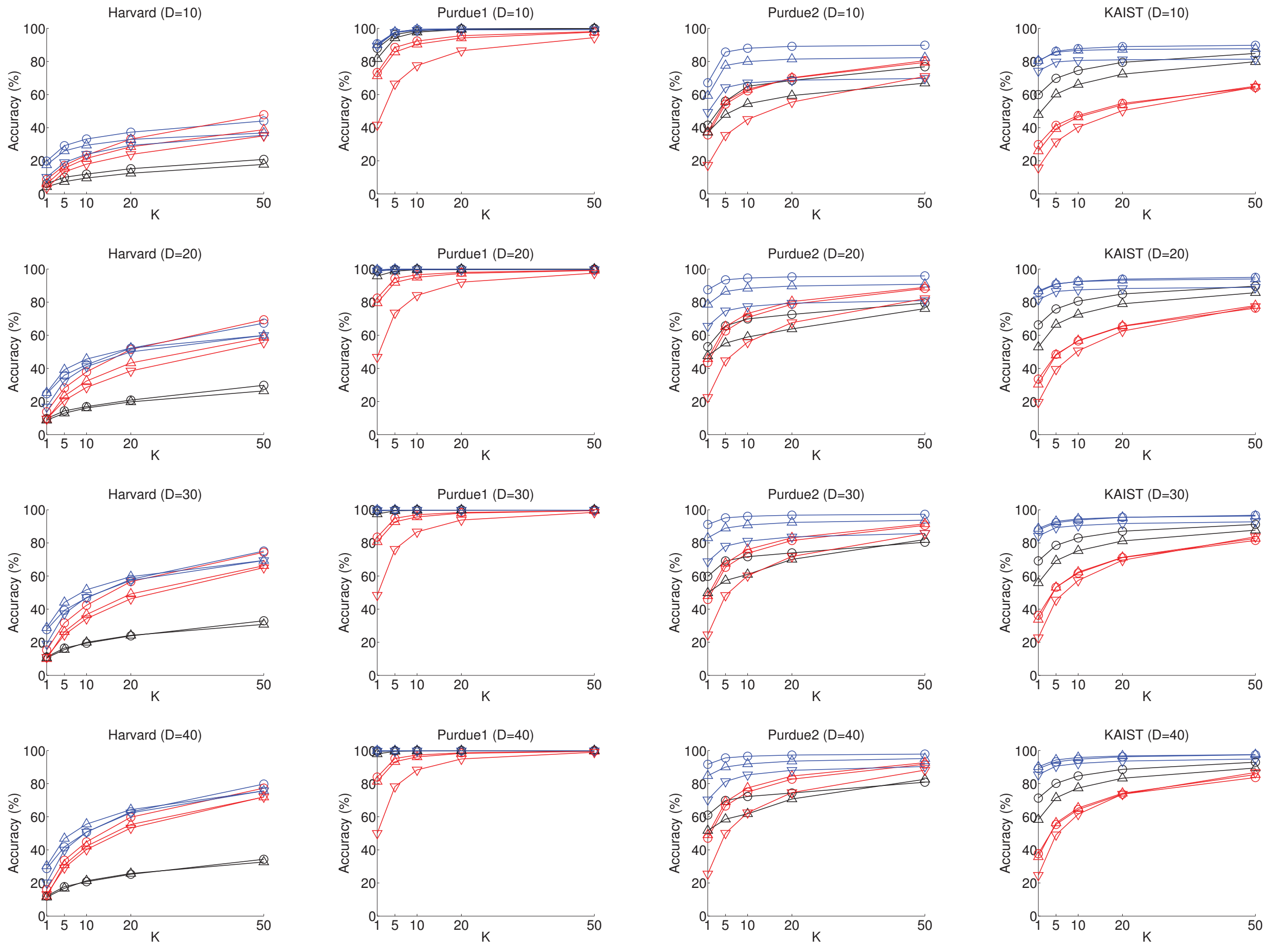


Figure 3: Localization accuracy of Barcode (Red), SURF-VLAD (Blue), and BRIEF-GIST (Black) under different distance thresholds (D). For each feature descriptor, the lengths are represented by the line markers. The line markers from longest ones to shortest ones are represented by $\bigcirc$, $\triangle$, and ∇, respectively. The X axis is for K, the numbers of nearest neighbors.

sampled within the video sequence to provide an overview. To keep anonymity and protect the privacy, we mask the license plates of these three sequences, which were captured by ourselves. For sequences KAIST, since the vehicle was not moved at the beginning, we sampled frames from later part of the sequence with equal interval.

2. Turn Classification

Figure 12 shows the turn classification result with the training and testing sequences, and Figure 13 shows the video frames uniformly sampled from these sequences. Those four sequences were part of the KITTI benchmark suite to evaluate SLAM algorithm performance [2]. While KITTI has 22 sequences for SLAM evaluation, we chose these four since they were captured in suburban areas and thus contain more buildings than the others. The top two rows show the results for the testing sequences, which can incorrectly show left (right) and right (cyan) turns along straight paths. It should be noted that in our two testing sequence, one of them rarely contains right turns, as shown in the second row of Figure 13. Figure 2 presents the measured accuracies of that sequences, which consequently show low accuracies for right turns due to the lack of samples.

References

[1] Y. Choi, N. Kim, K. Park, S. Hwang, J. Yoon, and I. Kweon. All-day visual place recognition: Benchmark dataset and baseline. In *CVPR 2015 Workshop on Visual Place Recognition in Changing Environments*, pages 8–10, 2015. 6

[2] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR 2012: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012. 2

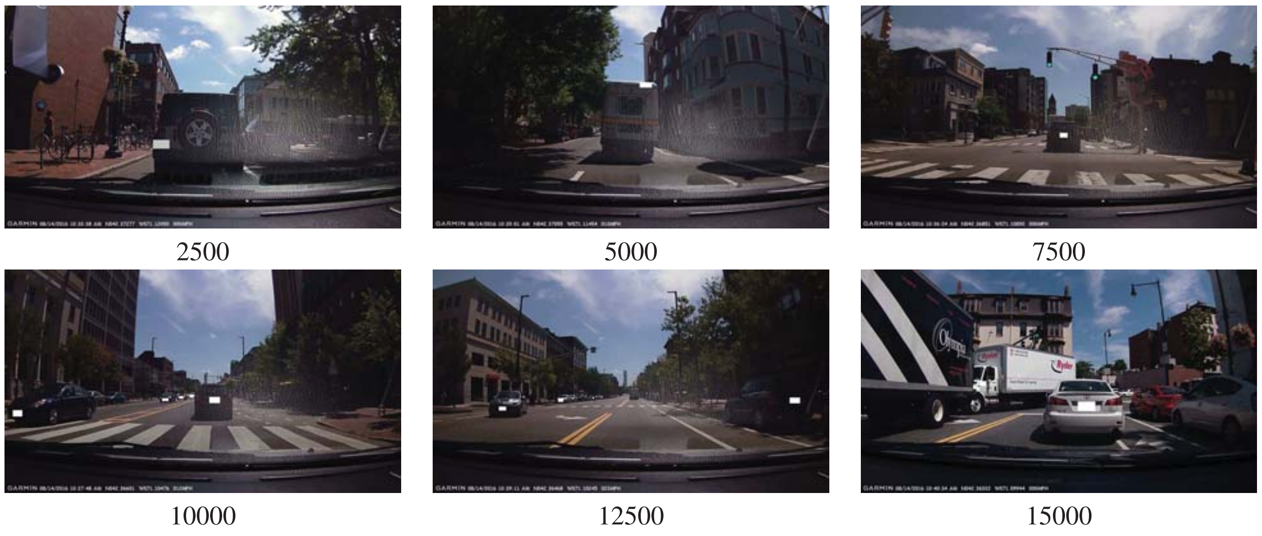


Figure 4: Sampled frames from the database of sequence Harvard.

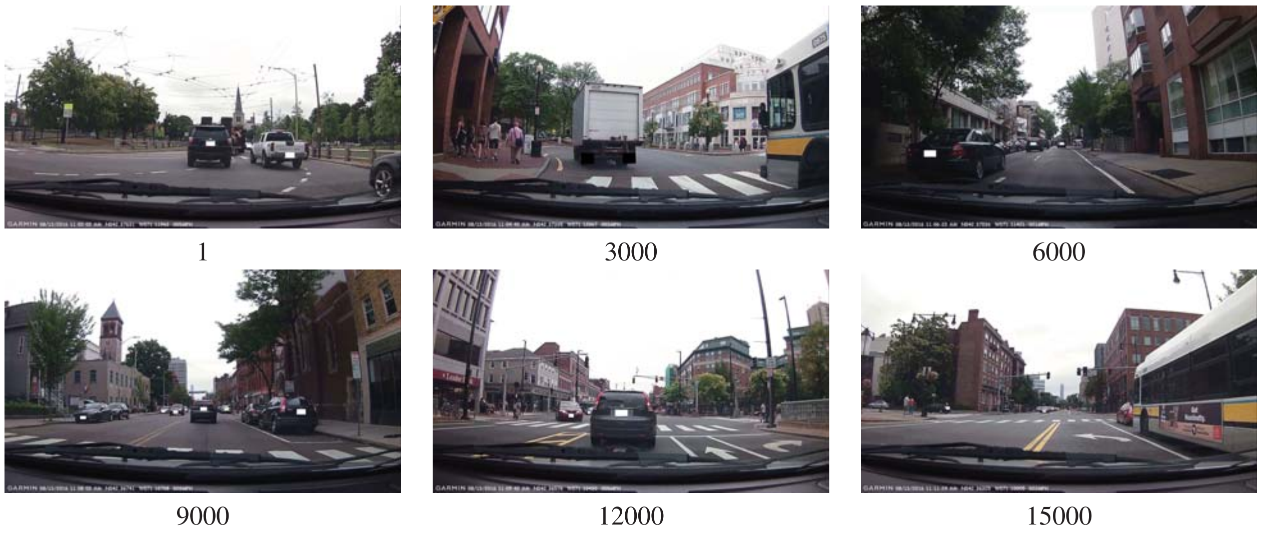


Figure 5: Sampled frames of sequence Harvard for query.

Figure 6: Sampled frames from the database of sequence Purdue1.

Figure 7: Sampled frames of sequence Purdue1 for query.

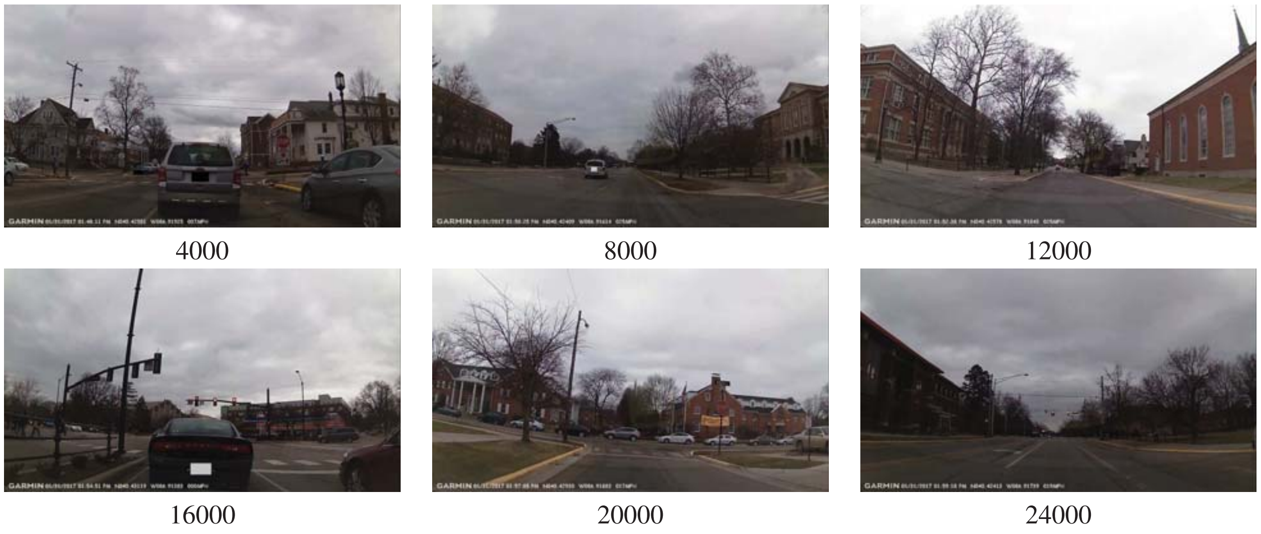


Figure 8: Sampled frames from the database of sequence Purdue2.

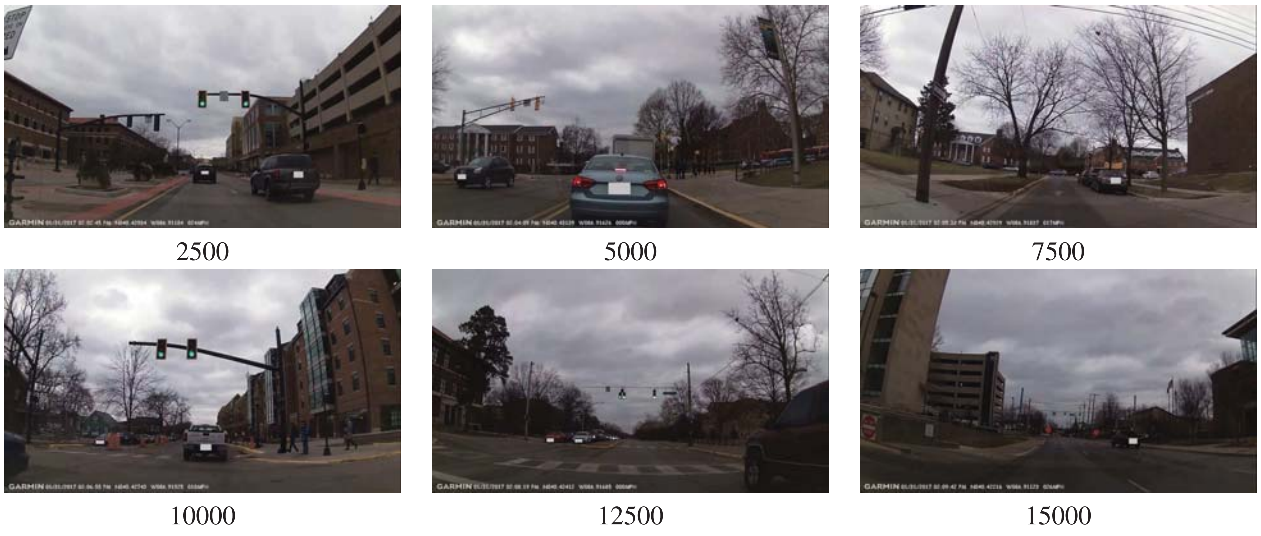


Figure 9: Sampled frames of sequence Purdue2 for query.

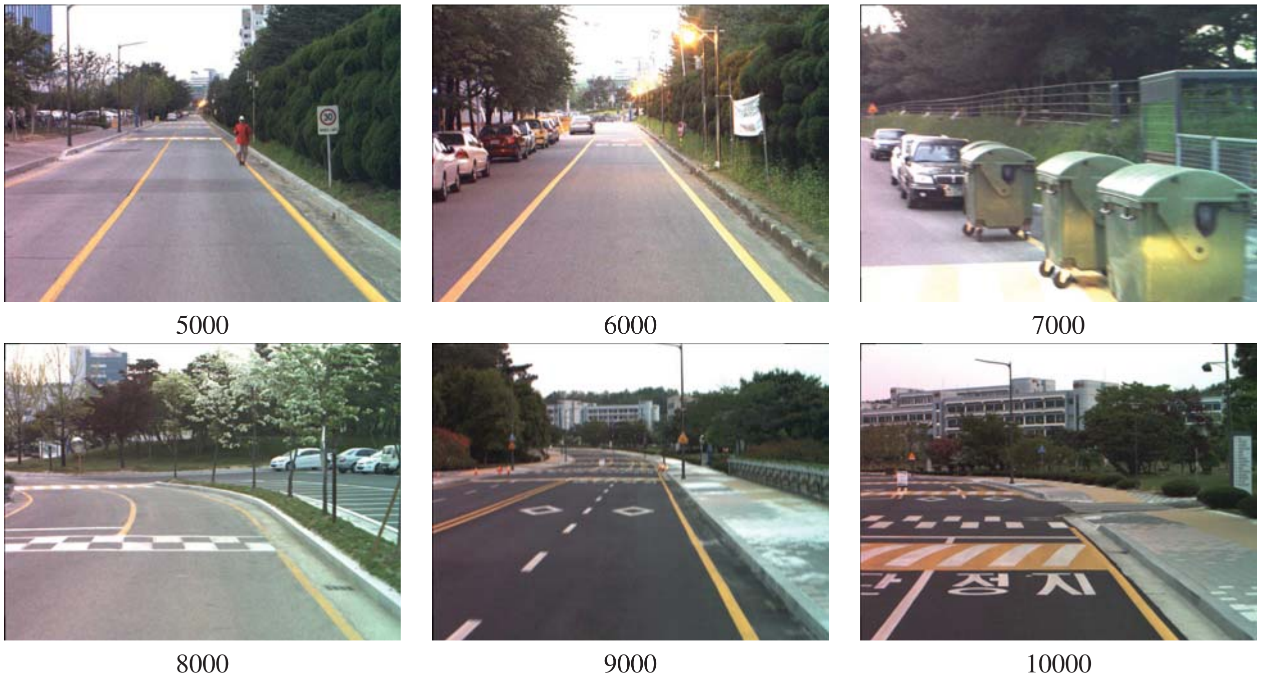


Figure 10: Sampled frames from the database of sequence KAIST. This video sequence was captured at 5:00am in the west campus of KAIST [1].

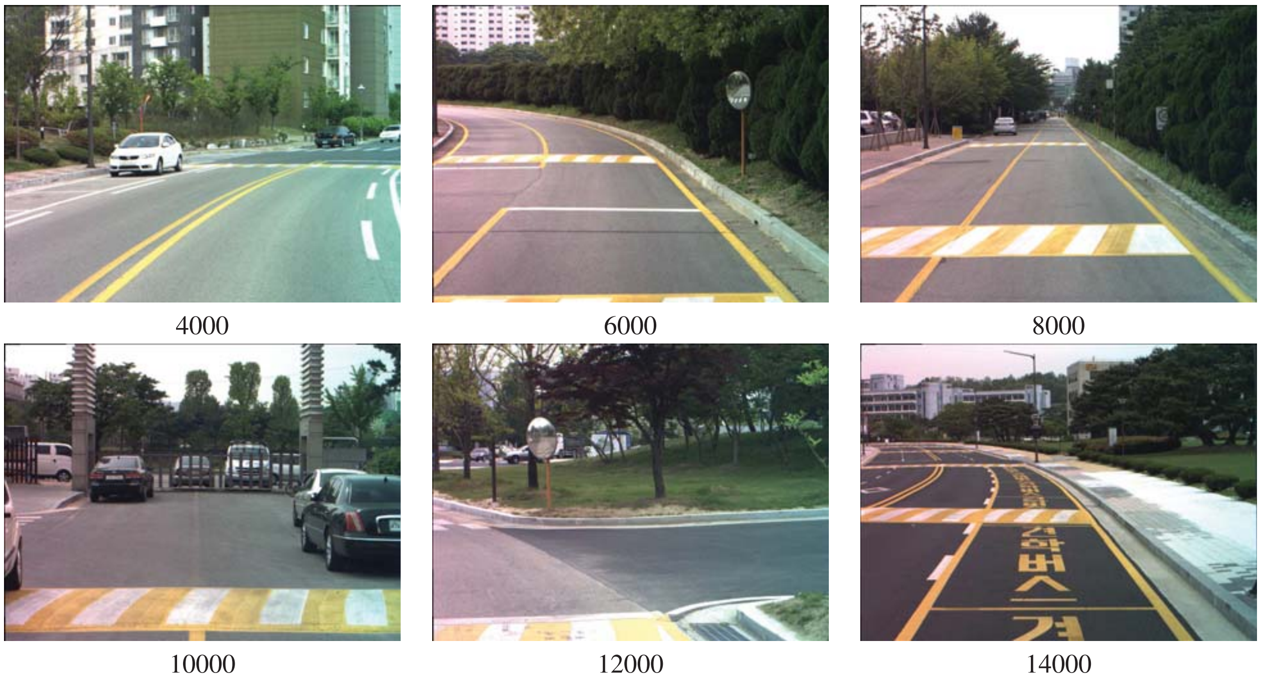


Figure 11: Sampled frames of sequence KAIST for query. This video sequence was captured at 9:00am in the west campus of KAIST [1].

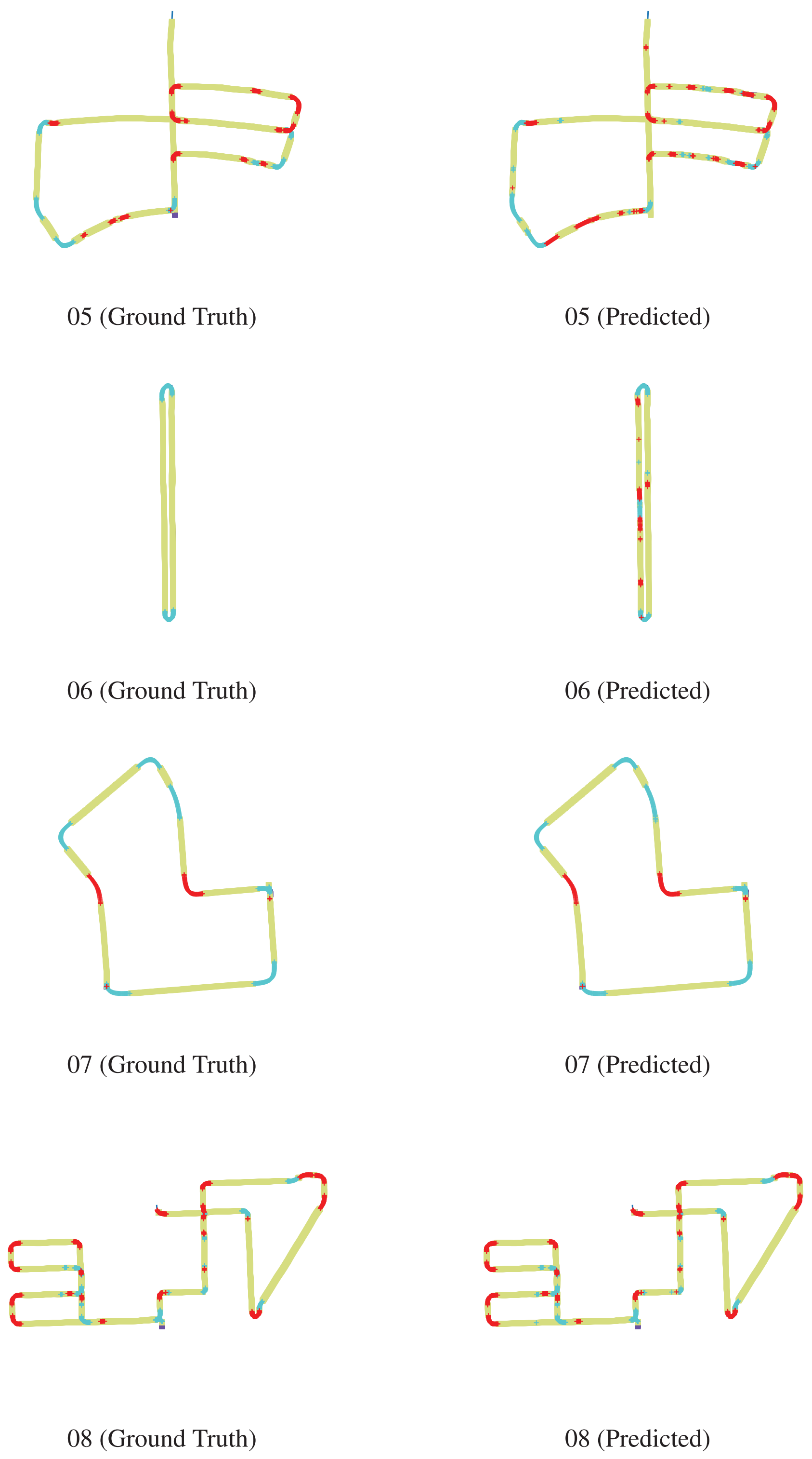


Figure 12: 2D visualization of the 4 KITTI sequences for SLAM evaluation and the result with our turn classification algorithm. Sequences 05 and 06 were used to test the turn classifier, whereas sequences 07 and 08 were used to train the turn classifier. The colors along the path indicate the turn categories, which are red for left turn, yellow for straight, and cyan for right.

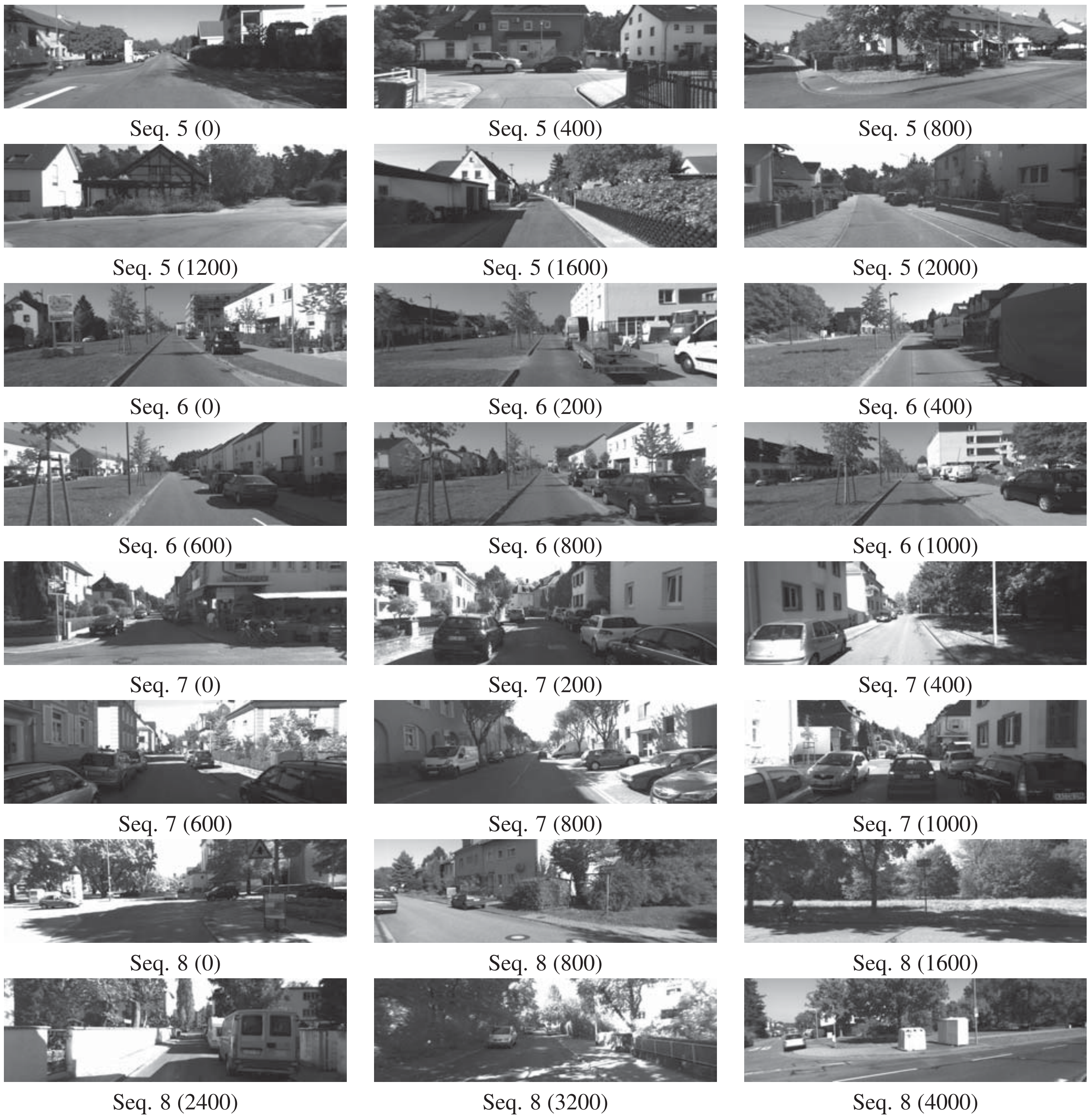


Figure 13: Sampled frames of training (sequences 7 and 8) and testing (sequences 5 and 6) KITTI sequences. The numbers in the parentheses indicate the corresponding frame indices.