Computation and Memory Efficient City-Scale Image-based Localization

Ngoc-Trung Tran[1], Dang-Khoa Le Tan[1], Anh-Dzung Doan[1], Thanh-Toan Do[2],
Tuan-Anh Bui[1], Ngai-Man Cheung[1]

***Abstract*— We present the design of an entire on-device system for potentially city-scale localization. The proposed design integrates compact image retrieval and 2D-3D correspondence search to estimate the camera pose in a city region of extensive coverage (∼15km). Our design is GPS agnostic and there is no need of network connection. In order to overcome the resource constraints of mobile devices, we carefully optimize the system design at every stage: we use state-of-the-art image retrieval to quickly locate candidate regions and limit candidate 3D points; we propose an efficient and effective pose estimation method. We explores the use of a collection of 227K Google Street View (GSV) images, to evaluate our system. The experiments are also conducted on benchmark datasets for 2D-3D correspondence search. Results show that our 2D-3D correspondence search achieves competitive accuracy, and requires much lower requirements of memory and computation than the state of the art. Our overall system can achieve good results on GSV dataset; the median error is about 4 meters. To the best of our knowledge, our system is the only work can handle such scale data on device and still achieve high accuracy on this challenging dataset.**

I. INTRODUCTION

Estimating accurately the absolute camera pose (with respect to a real world coordinate) is a fundamental problem of robot navigation (both indoor and outdoor). It is important for robot to firstly determine its own position before planning the path, navigating or interpreting the scene representation. Designing an accurate large-scale camera localization system running on limited computational/memory devices is thus a key problem in robotics. This system is also useful for many applications, such as augmented reality, autonomous vehicle navigation, location recognition, etc...

Using visual/image sensors (e.g., camera) are often the first consideration to develop the localization system because it provides more information about the scene structures than other sensors. Then, sensors data such as GPS (Global Positioning System), WiFi, Bluetooth are used to support/simplify the localization. However, the accuracy of embedded GPS sensors in devices highly depends on the surrounding environments. The GPS-based location performs poorly in downtown areas and urban canyons, e.g., the localization error is up to 100m or more [27]. Moreover, the GPS information is sometimes not available for indoor. The GPS is also sensitive to magnetic disturbances and can be denied/lost or easily hacked, which is not suitable for high security robots. WiFi/Bluetooth can be also used for localization, but they are not always available for outdoor. Thus, it is necessary to build an entirely on-device image-based localization system without the support of GPS/Bluetooth/WiFi.

State-of-the-art methods for image-based localization [1], [2], [3] assume the existence of 3D scene models. The 3D models are often pre-built from image datasets by using advance techniques of Structure-from-Motion (SfM) [4]. While such methods are suitable for powerful machines (workstation/servers), they are difficult to be employed on mobile devices due to the resource constraints of the devices. Current solutions for the location using mobile devices are to confine to small areas [5] or require back-end supports [6], [7], [8]. Compressing schemes [9], [10] can also be considered [11] but the issue still remains when the map grows. Alternatively, partitioning data/models into smaller parts, and manipulating only one part at a time is a scalable approach. However, determining the appropriate parts usually require GPS/WiFi or manual inputs [12].

In this paper, we propose an entirely on-device large scale camera localization system, i.e., we can localize the camera position in a large real-world environment. Firstly, in order to overcome the on-device resource constraints for large-scale problem, we propose a design that integrates a compact image retrieval and 2D-3D correspondence search methods. This design allows us to load only a sub model into RAM for processing, hence reduce the memory and computation requirement. We show that with the proposed design, it is feasible to enlarge the coverage to city-scale. Secondly, we propose an efficient and effective 2D-3D correspondence search which relies only on Hamming computation. This makes the proposed method is suitable for the memory and computation capabilities of devices. Our method is in a cascade manner that allows us to quickly search top nearest descriptors. To achieve high accuracy, we first adopt a low ratio test threshold [14] to accept more matches (both inliers and outliers). We then propose a simple but efficient RANSAC to handle matches. We demonstrate the 2D-3D correspondence search on benchmark datasets. Finally, we demonstrate our system for outdoor, by using street view images of Google Street View (GSV) [16]. Our purpose is to investigate the potential of using GSV dataset for camera lolcalization problem, which is interesting for practical localization systems. While there exists a number of prior works building their systems on GSV [17], [18], [19], [20], our work is different from them by focusing on the estimation of camera pose in city scale using a mobile device.

[1]The authors are with the Information Systems Technology and Design (ISTD) pillar, Singapore University of Technology and Design (SUTD), 487372, Singapore.

[2]The authors are with the Australian Centre for Robotic Vision, University of Adelaide, SA 5005, Australia.

Contact: ngoctrung_tran@sutd.edu.sg

II. RELATED WORKS

A. Image based Localization

The state of the art of image based localization is the 3D model based approach. This approach directly performs the 2D-3D matching between 2D features of query image and 3D points of 3D model. A 3D model is a set of 3D points, given a set of images, the 3D point cloud is built from the images by using modern Structure-from-Motion (SfM) approaches, e.g. [4]. This approach achieves better results than image retrieval based approach [21], [22], [23], [17] as it imposes stronger geometric constraints. Preferably, it tells more information about 3D structure of the scene. The camera pose can then be computed from 2D-3D correspondences by RANSAC within DLT inside. The representative works of 3D model based approach are [24], [9], [25], [15], [1]. [24] use image retrieval to find visible 3D points on top retrieved images and then computes 2D-3D matches between 2D features in the query against those points. Synthetic views of 3D points are generated to improve image registration. [9] compresses the 3D model and prioritize 3D points that allows the "common" views to be localized quickly. [25] propose the efficient prioritization scheme which is able to early stop the 2D-3D direct search. [15], [1] propose bi-directional searche procedure from 2D image features to 3D points and vice versa, which can recover some lost matches due to ratio test.

B. On-device systems

Some works tried to build localization systems running on mobile devices. [12] keep the 3D model out-of-core and manually divide it into multiple segments, and index 3D points by potentially visible set. The exhaustive matching is expensive that confine this work to small workspaces. It additionally requires the initial location by sensor data or manual inputs. The work is extended for outdoor localization [6], but still need prior knowledge of coarse location, and wireless network to download relevant portions of pre-partitioned databases. [7] and [8] use the client-server architectures. Pose tracking is employed on devices and its pose is corrected by matching to global model to avoid the drift. While [7] need to keep a part of global model locally to obtain fast tracking, [8] build their local map of the environment and using global results received from external server to align this map. [5] use Harris corner detectors and extract two binary features for tracking and 2D-3D matching. It avoids wasting computation by performing matching over only a small batch of tracked keypoints. [11] implement pose estimation and tracking on device. It uses Inverted Multi-Index (IMI) ([26]). To reduce the computation and make it processable on device, they compress their 3D models by a similar method of [9].

C. Using Street View images for localization

One of difficulties as developing a large-scale image based localization is data collection, which requires ground-truth data, e.g. GPS, in real-world. Serveral on-device systems [6], [7], [8], [5], [11] have to collect their own dataset for experiments which are usually confined to small areas. Mining images from online photo collections like Flickr [4] is an attractive solution. However, this undertaking is challenging due to poor labels, noisy distortions to be distributed in the real world. Also, the coverage of images are often popular places, e.g. city landmarks. [27] approached by using cameras-mounted surveying vehicles to harness the street-level data in San Francisco. They published a dataset containing 150k high-resolution panoramic images of San Francisco to the community. [17] use GSV images to localize UAV by generating virtual views and matching images with strong view point changes. [19] perform tracking the vehicles inside the structure of street-view graph by a Bayesian framework. This system requires compasses measurements, fixed cameras within many assumptions of video capturing conditions. [20] track the pose of a camera from a short stream of images and geo-register the camera by including GSV images into local reconstruction of image stream. Nearby panoramic images determined by image retrieval with restricts of locations inferred by GPS or cellular networks in surrounding 1km area.

III. PROPOSED SYSTEM

We first provide an overview of our proposed on-device localization system. Then, we discuss our 2D-3D correspondence search.

A. On-device localization system

Our proposed system design aims to: i) leverage a large-scale collection for localization; ii) overcome the constraints of memory and computation when using a large-scale dataset on a typical mobile device. The input of our system is a query image taken by user's camera. The output is the real-world GPS location. Our system is shown in Figure 1: a set of 3D reconstruction models to represent the scene, an image retrieval system to identify 3D models for pose estimation, and a system to compute correspondences between 2D features and 3D points Section III-B. In this work, our choice of image retrieval, a state-of-the-art method [28], is for memory-efficiency, good accuracy and fast indexing. It aims to demonstrate the possibility of our system design. We then focus more on improving localization part to reduce its latency and to get our on-device system work more practical. We choose SIFT [14] features as the input for both image retrieval and 2D-3D correspondence search, simply because its reliability and efficiency has been demonstrated in the literature of those two fields. Though the same pipeline can be exactly applied for other kinds of features.

1) Dataset: We collect 227K GSV images at resolution of 640×640 pixels, that corresponds to approximately 15km road distance coverage. These images have exact GPS. We also collect 576 query images with the accurate GPS ground-truth position. These query images are captured using different devices and under different conditions from that of GSV. Those images that cover city regions in Singapore. Contrary to previous works, many images in our dataset are typical residential or commercial buildings, which are much less distinctive. At each Street View *place mark* (a spot on the street), the 360 degree spherical view is sampled

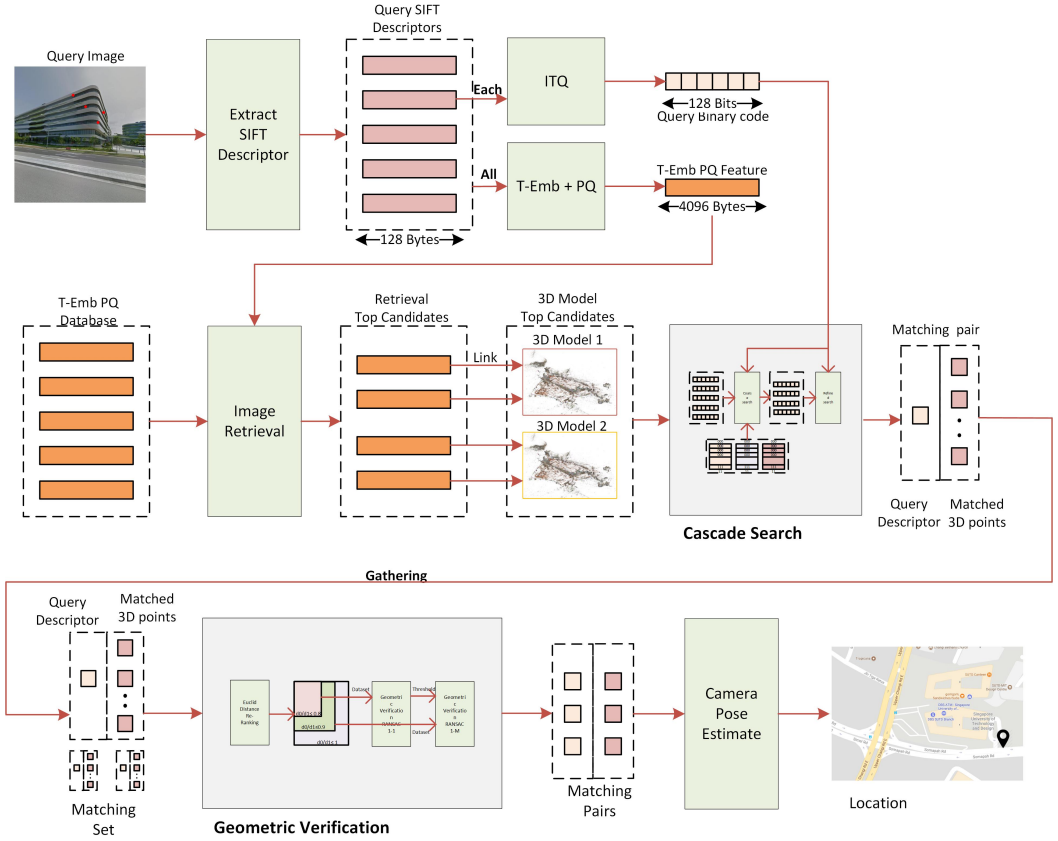

Fig. 1. Overview of our proposed system: (a) Feature are extracted and convert to PQ and Image retrieval (IR) identifies reference images similar to the query image. The reference images returned by IR indicate 3D models for use in camera pose estimation.

by 20 rectilinear view images (18° interval between two consecutive side view images) at 3 different elevations (5°, 15° and 30°). Each rectilinear view has 90° field-of-view and is considered a pinhole camera. Therefore, 60 images are sampled per place mark. The distance between two place marks is about 10-12m.

2) Scene representation using small 3D models: We divide scene into sub-regions and build each individual model per region. In addition to fit the model into the RAM, there are many other benefits: (i) building small is cheap and can be done in parallel, (ii) representing the scene by a single model is hard to manage/maintain: It is rather difficult to update a large model when some region of the city changes (e.g. newly constructed building). Moreover, provided that the correct small 3D models can be identified, localization using small 3D models can achieve better accuracy as there are less number of distracting 3D points. Furthermore, localization time can be reduced using small 3D models. We use 8-10 consecutive GSV place marks to define a segment of the scene. As we sample 60 street view images per place mark, there are 480-600 images for a segment. We use Incremental SfM [29] to reconstruct a 3D model from the images of a segment.

Overlapping of segments: We investigate overlapping between two consecutive segments. This is to ensure accurate localization for query images capturing buildings at the

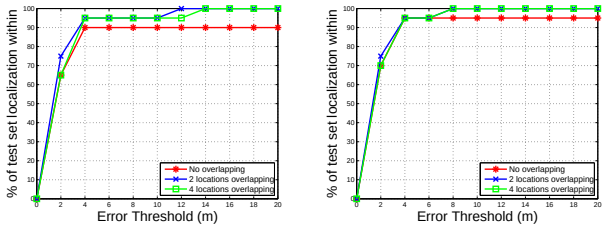

Fig. 2. The number of overlapped place marks of two segments, and its influence on image-retrieval top list of 20 or 50.

segment boundaries. We conducted an experiment to evaluate the localization accuracy at zero, two and four place marks overlapped. In this experiment, we used image retrieval to find the 20 or 50 top similar database images, given a query image. Results in Fig. 2 suggest that with segments overlapped at two place marks can ensure good localization accuracy. Note that the extent of overlapping is a trade-off between accuracy and storage. Besides, a retrieved list of 20 database images achieves good accuracy-speed trade-off.

Coverage of each segment: As the coverage (size) of each segment increases, the percentage of overlapped place marks decreases and hence storage (3D points) redundancy decreases. However, the localization accuracy decreases as the segment coverage increases, because there are more

#Place marks	#Images	% of #queries with error $\leq$ 5m
8-10	480-600	90%
11-14	660-840	80%
20-25	1200-1500	60%

TABLE I

THE INFLUENCE OF SEGMENT SIZE ON THE LOCALIZATION ACCURACY.

distracting 3D points in a 3D model. We conducted an experiment to determine an appropriate segment size: We reconstructed a 3D model from a number of images: 480-600 images (8-10 place marks), 660-840 images (11-14 place marks), and 1200-1500 images (20-25 place marks). We applied the state of the art method, Active Correspondence Search (ACS) [15], to compute the localization accuracy using the 3D model. Table I shows the results, which suggest that using segment with 8-10 place marks achieves the best accuracy. The localization accuracy degrades rapidly as we increase the segment coverage for GSV dataset. Therefore, in our system, we use 8-10 consecutive GSV place marks to define a segment. Although [12] has also proposed to divide a scene into multiple segments, their these design parameters have not been studied. Moreover, its design is not memory-efficient, the 2D-3D matching is expensive. Therefore, it is just confined in small workspace, and also requires many manual tasks to build/index/register models. It needs sensor data, e.g. GPS, WiFi to determine the search region. On the other hand, our models are automatically reconstructed or registered, and our system can localize entirely on mobile device at large scale.

3) Model indexing by image retrieval: We also use image retrieval (IR) in our framework to index the list of 3D models $\mathbb{M}_i$. IR serves as a coarse search to limit the searching space for pos estimation. Let $\{I_j\}_{j=1:N}$ be the N images in dataset. If the image I_j was used to reconstruct 3D model $\mathbb{M}_i$, we set $r(\mathbb{M}_i, I_j) = 1$, otherwise $r(\mathbb{M}_i, I_j) = 0$. Given a query image I_q, image retrieval seeks top N_t similar images from the dataset, namely $I_{j_1}, I_{j_2}, ..., I_{j_{N_t}}$. $\mathbb{M}_i$ is a candidate model if $\exists I_{j_s}$: $r(\mathbb{M}_i, I_{j_s}) = 1$, $s = 1 : N_t$. Note that IR may identify multiple candidate models (N_m) for localizing the query image. In this case, the camera pose is estimated using the 3D model with the maximum number of 2D-3D correspondences (Section III-B). The summary of image retrieval is as follows: First, we extract SIFT features [14] and embed them into high dimensional using Triangulation Embedding (T-embedding) [28]. As a result, each image has a fixed-length T-embedding feature as a discriminative vector representation. We set the feature size to 4096. To reduce the memory requirement and improve the searching efficiency, we apply Product Quantization (PQ) with Inverted File (IVFADC) [30] to the T-embedding features. Details can be found in [30], [28]. Note that the PQ codes are compact. As a result, we can fit the entire PQ codes of 227K reference images into the RAM of a mobile device. Processing time for IR is less than 1s (GPU acceleration) for 227K reference images on a mobile device.

Using IR to index 3D models is memory efficient because only a few number of models are processed each time. However, it is more expensive by 2D-3D correspondence search due to matching between query and N_m models. That leads to our idea of the proposed correspondence search, which aims to reduce this computational complexity.

B. Fast 2D-3D correspondence search

We propose a new 2D-3D correspondence search for mobile constraints by leveraging the efficiency of Hamming computation. Our proposed method consists of two parts: 2D-3D matching aims to be memory and computational efficiency, and an effective RANSAC to achieve the accuracy.

1) Cascade search for 2D-3D matching: Our matching scheme is inspired the Pigeon Principle of hashing code [31]. We apply it into our 2D-3D context with further improvements: (i) In our work, since the 3D models are built off-line and SIFT descriptors for 3D points are available during off-line processing, we propose to train a data-dependent hash function to improve matching accuracy. ITQ [33] is used in this work. (ii) We use a single 128-bits hash function for both coarse search and refined search of Hamming distance ([31]). Splitting the long 128-bits code into N_l short codes of b bits to construct N_l look-up tables (LUT) for coarse search and full 128-bit vector for refined search. Our prioritized search speed up about $\sim 10\times$ as validated on one benchmark dataset. (iii) We propose to use ratio test based on Hamming distance to decide whether a match is inliers or outliers. And we further improve the speed by using prioritizing scheme based on the number of candidates resulted from the first layer.

The pipeline of our proposed 2D-3D matching method as cascade search is shown in Fig. 1. The method consists of two main steps: coarse search, refined search. Let $d = 128$ be the feature dimension of SIFT descriptors. Given a 3D model, each descriptor of d dimensions in $\mathbb{R}^d$ are pre-mapped into h in Hamming space $\mathbb{B}^d$ of d dimensions by using ITQ. ITQ mapping function was learned from all data points of 3D model.

Coarse search: we split binary vectors (hash codes) into N_l sub-vectors $\{h_i\}, i = 1 : N_l$ of b bits ($N_l * b = d$). In training, we build a LUT for each sub-vector, which has $K_b = 2^b$ buckets. Each 3D point are assigned to buckets according to their binary sub-vectors. For searching, a query descriptor is first mapped into Hamming space, and was divided into N_l sub-vectors of b bits. Each sub-vector is used to index its matched bucket in the corresponding LUT. All 3D points in the matched bucket are retrieved. All retrieved 3D points from the N_l LUT are the list of coarse search results. By using LUT, the search complexity is constant $O(1)$. This step results a short list l_C of candidates for the next search. It is important to choose appropriate values of N_l and b to trade-off between the memory requirement of LUT and computation time (which depends on the number of 3D points returned by coarse search and therefore requiring Hamming distance re-ranking). As shown in Table II, ($N_l = 4$,$b = 32$) is impractical due to over-large size requirement of lookup tables. ($N_l = 16$,$b = 8$) results in too many candidates for

Method(N_l,b)	l_C	Look-up tables size
(4,32)	-	$4 \times 2^{32} \times 4$ = 64GB
(8,16)	$\sim$ **5K**	$8 \times 2^{16} \times 2$ = **1048KB**
(16,8)	$\sim 60K$	16×2^{8} = 4096B

TABLE II

THE TRADE-OFF BETWEEN THE NUMBER OF CANDIDATES l_C AND THE SIZE OF LUT ON DUBROVNIK DATASET.

step of refined search by using 128-bit Hamming distance, which slows down the search. ($N_l = 8$,$b = 16$) results in a small number of candidates from coarse search and requires a small amount of memory (excluding the overhead memory of descriptors indexing).

Refined search: It uses 128-bit h to encode one descriptor. The goal is to refine the l_C candidates to select a shorter list l_R ($l_R < 50$) based on Hamming distance. Computing Hamming distance is efficient because it can leverage low-level machine instructions (XOR, POPCNT). Experiments on our machine show that computing Hamming distance of two 128-bits vectors is ($\geq$ 30x) faster than Euclidean distance of that. First, computing exhaustively the Hamming distance between the 128-bit query code to that of l_C candidates. Those distances are re-ranked to select top l_R out of l_C according to those distances. The Hamming distance of 128-bit code has maximum range of 128 values, which enables to build the online lookup table of those values to remain low complexity $O(l_R)$ of top-l_R search.

Prioritized search: Finding all matches between 2D features and 3D points to infer camera pose is expensive even with our current cascade hashing search, because the query image can contain thousands of features. In practice, the method can early stop once found a sufficient number of matches ([25]). Therefore, we perform prioritized search to first process those query features with small number of retrieved 3D points resulted from the coarse search. In our implementation, the search stops once $N_{early} = 100$ correspondences have been found.

2) Geometric verification: After matching, correspondences (one-one matches) is again verified by geometric constraints. RANSAC within 6-DLT algorithm are often used. Empirically, we found that the threshold $\nu_h = 0.6$ rejects a lot of good matches, and the good ones are not always top-ranked. It is probably due to repetitive features in buildings that is a common issue of localization in urban environment [3], [35]. Accepting multiple matches before geometric verification seems a potential solution. Recent works [2], [35] aim to approach this way, but their proposed methods are slow on their workstations that is impractical especially for low computational devices.

We relax the threshold, e.g. $\nu = 0.8$, to keep multiple matches per query descriptor. We propose a RANSAC method to effectively infer the camera pose from those relaxed matches. More detail, for a 2D query feature and given a ratio test threshold ν, we accept all candidates in the top-l_R list with distance less than $h_0/h_1 < \nu$, where h_0 and h_1 are the first and second smallest Hamming distances of the query to l_R candidates. However, processing all these matchings leads to an exponential increase in computational time in RANSAC. We avoid this issue by considering its subset to generate hypothesis. As consequence, we propose two different sets of matchings in the hypotheses and verification stages of RANSAC. The first set contains the one-one (1-1) matchings that pass a ratio test with threshold $\nu_h = 0.6$. The second set contains one-many (1-M) matchings as mentioned above (containing hypothesis set). We propose to use the first set to generate hypotheses and the second set for verification. We found that using 1-M matchings in verification can increase the number of inliers, leading to the accuracy improvement. We further speed up our method by using the pre-verification step like [36] in the middle, to quickly reject "bad" samples before performing verification.

IV. EXPERIMENTAL RESULTS

We evaluate our 2D-3D correspondence search on four benchmark datasets: Dubrovnik [9], Rome [9], Vienna [24], and Aachen [37], and compare to the state of the art. And we investigate our on-device system, which trained from our dataset of 227K images, by using 576 mobile queries for IR and localization. Experiments are conducted on our workstation: Intel Xeon Octa-core CPU E5-1260 3.70GHz, 64GB RAM, and Nvidia Tablet Shield K1. We use "mean" descriptor of each 3D point in all experiments. We also setup different setting of our 2D-3D matching. We have different settings of our method: **Setting 1** indicates our method with threshold $\nu_h = 0.6$, **Setting 2** with the relaxed threshold $\nu = 0.8$ and proposed RANSAC (fixed 5000 iterations) and **Setting 3** with prioritizing scheme and fast RANSAC included.

A. Hashing based 2D-3D matching

First experiment is conducted on Aachen dataset to determine a good ratio-test threshold. In this experiment we represent each 3D point by "mean" of its descriptors, and fix 5000 RANSAC iterations to remove randomness in RANSAC. A query image is "registered" if at least twelve inliers found, same as [9]. It is shown in Fig. 3 that threshold $\nu_h = 0.6$ achieves the highest registration rate. The value is different from some previous works, e.g. [25], probably due to the difference between Hamming distance and Euclidean distance. In the figure, the horizontal axis indicates the value of thresholds, and the vertical axis is a number of registered images. In the second experiment, we conduct experiments to choose the good value of l_R output from the second layer, Fig 3. All experimental contritions are the same, except we choose the best threshold $\nu_h = 0.6$ from the first experiment. The experiment suggests $l_R = 40$ is a good option because increasing it does not significantly change the registration rate.

Because of aforementioned reasons of repetitive features, the threshold $\nu_h = 0.6$ reject many good matches. Therefore, we propose to relax this threshold to $\nu = 0.8$, and keep many matches per query descriptor. First, we see that (Setting 2) adopts more matches really outperforms (Setting 1) at both

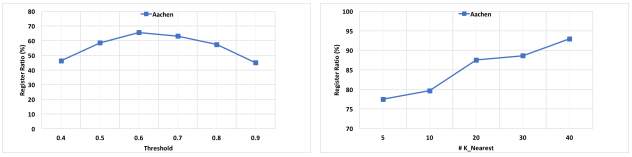

Fig. 3. The number of registered images using various thresholds (left). The number of registered images according to various value of l_R (right).

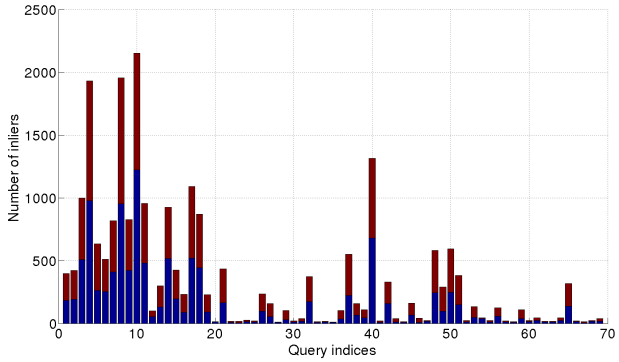

Fig. 4. The number of inliers found by strict threshold (blue) and relaxed threshold (red).

the the number of registered images and the accuracy. It confirms that using relaxed candidates per query RANSAC verification is useful to improve the performance. The performance of (Setting 2) is competitive to the state of the art. Specially, we achieve the best result on Aachen dataset. We try to visualize the inliers found by strict threshold and relaxed threshold on Aachen dataset in Fig. 4. Contribution of number of inliers found by relaxed threshold and 1-M matches are significant, increase 120% from the strict threshold. This is probably the reason we achieve good result on Aachen dataset.

Then, we report its improvements with prioritized scheme and fast RANSAC (Setting 3). Results are shown in Table III. The running time of (Setting 2) is fast enough as compared to previous works, but its running time can be further speeded with prioritizing scheme and fast RANSAC (Setting 3). (Setting 3) obtains similar performance as the full search but is about $\sim 5\times$ faster. By prioritizing scheme, we achieve the competitive accuracy, while faster matching time than all of previous works. Our fast RANSAC uses pre-verification. Our proposed fast RANSAC remains the accuracy, but $\sim 20\times$ faster from RANSAC of fixed 5000 iterations (0.60(s) vs. 0.03(s) on Dubrovnik query), as shown in Tables III. Our (Setting 3) significantly improve the localization performance of our method, while the results in Table III show its low complexity when comparing its running time as similar as RANSAC of ACS (on much smaller set of correspondences), e.g. 0.03(s) vs. 0.01(s) per Dubrovnik query. In addition to that, our proposed method is more efficient with regards to memory because of the use of compressed descriptors.

B. Overall system

We evaluate our overall system by considering the robustness of image retrieval on large-scale dataset, and the

Method	Dubrovnik		Aachen	
	#reg	Time (s)	#reg	Time (s)
Kd-tree [25]	795	34.1*	317	-
P2F [9]	753	-	-	-
RT_AP [38]	784.1	-	298.5	-
[37]	786	-	327	-
[3]	797	-	318	-
[39]	796	-	329	-
[35]	798	3.78$^+$	-	-
[2]	798	5.06$^+$	-	-
[1]	**800**	-	-	-
VPS (Int. Mean) [25]	782	2.33*	-	1.93*
ACS [15]	795.9	0.72*	-	0.56*
Setting 1	735	1.66	240	1.27
Setting 2	794	1.86	**343**	1.46
Setting 3	794	**0.47**	334	**0.48**

TABLE III

RESULTS ON DUBROVNIK AND AACHEN DATASETS. '*' ARE REPEATED ON OUR MACHINE, '+' REPORTED ONLY FOR OUTLIERS REJECTION/VOTING SCHEME TAKEN FROM ORIGINAL PAPERS. #REG INDICATES THE NUMBER OF REGISTERED IMAGES.

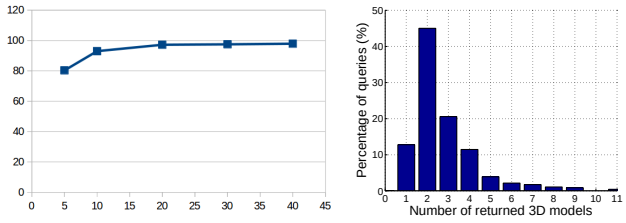

Fig. 5. The accuracy of image retrieval and the histogram of model number of threshold 20.

localization accuracy of overall system on our dataset.

1) Image retrieval: IR is to find correct 3D models that a query likely belongs to. A query image is "success", if N_t top list of retrieval images match to *at least* one correct model. For ground-truth, we manually index our set of queries to their corresponding 3D models. It is important to investigate IR's performance, because it have influence to the robustness of overall system, especially with large-scale dataset. We follow the parameters reported in T-embedding method [28] as represented above and use sum-pooling. The goal is to determine the number of retrieved image N_t should be resulted from image retrieval. N_t has to balance between the accuracy and the number of models found. Fig. 5 shows that $N_t = 20$ is an appropriate number. The horizontal axis is the number of references resulted from image retrieval, and vertical axis is the percentage of queries found at least one correct model. The histogram of model numbers is visualized in the same figure. More than 80% of queries found ≤ 4 candidate models, therefore, we practically perform 2D-3D matching with maximum number of four models if the list results more than this number.

2) Overall system localization: In this experiment, the localization accuracy is measured by GPS distance between ground-truth and our estimation. The results are drawn in form of Cumulative Error Distribution (CED) curve. The horizontal axis indicates the error threshold (in meters), and

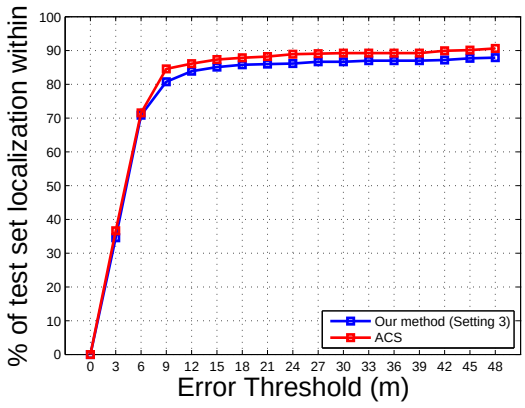

Fig. 6. The performances of overall system tested on 576 query images using two different 2D-3D matching method ACS and our method.

the vertical axis indicates the percentage of images that have lower or equal errors than the threshold. We compare correspondence search methods on the overall system: our method (Setting 3) and ACS [15] (current state-of-the-art method of image based localization). The same image retrieval component is used for both that was trained on 227K images. Fig. 6 presents real-world accuracies, e.g. at threshold of 9(m), about 80% queries are well-localized by (Setting 3), and ACS is just slightly better than our method. Our (Setting 3) use compressed SIFT descriptors (16 bytes), which optimized better memory requirements than ACS (128 bytes), but achieved comparable performance on our dataset. Our method also faster than ACS as reported on benchmark datasets. Moreover, our method is faster than ACS. Note that camera is calibrated in this experiment. About 10% of images are completely failed ($\geq$ 50 (m)) due to image retrieval, the confusion of similar buildings, or reflection of building facades. Our proposed system achieves encouraging results using GSV images: the median error of is about 4(m), and 70% of queries have errors less than 6(m). Those results of overall system demonstrates the efficiency and effectiveness of our system design on real-world dataset.

C. Mobile implementation

1) Image retrieval: Our vocabulary size of T-embedding is $k_{temb} = 32$, thus its T-embedding feature dimension is 4096. The fixed consumption of embedding and aggregating parameters is 67.14 (MB). The indexing step of PQ needs approximately 129.44 (MB) to encode $N = 227K$ images, where the number of sub-vectors is $g = 256$ and the number of sub-quantizer per sub-vector is 256. The total memory is 129.44 (MB), which can easily fit to modern devices with RAM $\geq 1GB$. As increasing the size of dataset to 1M images, the total memory consumption is 327.34 (MB) is still processable on the RAM.

2) 2D-3D correspondence search: Our (Setting 3) is implemented for mobile implementation as it is fast and requires less memory. The method requires 128-bit (16 bytes) hash code to encode a SIFT descriptor. Using $N_l = 8$ look-up tables, each one is comprised $K_b = 8 \times 2^{16}$ buckets. Each bucket needs a 4-byte pointer referring to one point-id list. Let N_p is the point number of the 3D model, if N_p is large enough, the overhead memory is small can be ignored. N_l tables refer to N_l point-id of total N_p points. One point-id can be represented by a 4-byte integer number. N_p 3d point coordinates consume $N_p \times 12$ bytes. Our model needs the total of $N_p \times 60$ bytes, which is nearly $\sim 2.5\times$ compressed than the original model of $N_p \times 144$ bytes as Table IV (ignoring the indexing structures of other methods that may require more memory). Our 227K images of approximately 15km road distance coverage consumes about 40MB of total memory. By some calculations, it is feasible to extend to 1 millions images which cover approximately about 70 (km), while consuming less than 2GB memory. We can extend further if storing 3D models on modern SD cards. It is worth noting that the overall performance for such extensions would only affect to the accuracy of image retrieval, not 2D-3D correspondence search.

	Our model	**Original model**
Look-up tables	$8 \times 2^{16} \times 4$	-
Point id	$8 \times N_p \times 4$	$N_p \times 4$
Point coordinates	$N_p \times 12$	$N_p \times 12$
Descriptors	$N_p \times 16$	$N_p \times 128$
TOTAL MEMORY	$\approx N_p \times 60$	$N_p \times 144$

TABLE IV
MEMORY REQUIREMENTS (#BYTES) FOR OUR MODEL VS. ORIGINAL MODEL.

3) On-device running time: Our system is implemented on Android device: Nvidia Tablet Shield K1, 2.2 GHz ARM Cortex A15 CPU with 2 GB RAM, NVIDIA Tegra K1 192 core Kepler GPU, 16GB storage. Our camera solution is 1920×1080. Table V reports the running time for each individual steps: feature extraction, image retrieval, 2D-3D matching and RANSAC. Since SIFT extraction is time-consuming, it is implemented in GPU. Image retrieval is also accelerated by GPU, whereas two other components are on CPU. Processing time of image retrieval is acceptable and consistent with dataset size. Running time of 2D-3D matching is reported for only one model. In practice, a few of models (≤ 4) are manipulated at a time and the latency of loading one model is low, about 0.04 (s). Therefore, it takes average about 10(s) in total to localize one query (including hard examples), but for good cases, it takes average only 3-4(s) per query. The localization and pose estimation parts are based on single CPU core, the speed of our system can be further optimized/improved with multi-core CPU and GPU in future work. Note that we calculate the codebook size $K_{qc} = \frac{N_p}{10}$ as training ACS on our own models and other parameters as the same reported in [40].

V. CONCLUSION

We present complete design of an entire on-device system for large-scale urban localization, by combining compact image retrieval and fast 2D-3D correspondence search. The

Step	Time (s)
Feature extraction (GPU)	0.67
Image retrieval (GPU)	0.82
Cascade search	0.55
Pose estimation	1.15

TABLE V

AVERAGE RUNNING TIME FOR EACH INDIVIDUAL STEP ON OUR DEVICE.

proposed system is demonstrated via the dataset of 227K GSV iamges (with approximately 15km road segment). The scale of system can be readily extended with our design. Experiment results show that our system can localize mobile queries with high accuracy. The processing time is less than 10s on device. It demonstrates the potential of developing a practical city-scale localization system using the abundant GSV dataset.

We propose a efficient and effective 2D-3D correspondence search for localization by combining prioritized hashing technique and RANSAC. Our RANSAC can handle a large number of matches to achieve higher accuracy, while maintaining the same execution time as traditional RANSAC. Our matching method requires $\sim 2.5\times$ less memory footprint than using original models. Our matching method achieved competitive accuracy as compared to state-of-the-art methods on benchmark datasets, especially we obtained the best performance of both processing time and registration rate on Aachen and Vienna datasets.

REFERENCES

[1] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua, "Worldwide pose estimation using 3d point clouds," in *ECCV*, 2012.

[2] L. Svrm, O. Enqvist, M. Oskarsson, and F. Kahl, "Accurate localization and pose estimation for large 3d models," in *CVPR*, 2014.

[3] T. Sattler, B. Leibe, and L. Kobbelt, "Efficient effective prioritized matching for large-scale image-based localization," *TPAMI*, no. 99, pp. 1–1, 2016.

[4] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3d," in *SIGGRAPH*, 2006.

[5] H. Lim, S. N. Sinha, M. F. Cohen, and M. Uyttendaele, "Real-time image-based 6-dof localization in large-scale environments," in *CVPR*, 2012.

[6] C. Arth, M. Klopschitz, G. Reitmayr, and D. Schmalstieg, "Real-Time Self-Localization from Panoramic Images on Mobile Devices," in *ISMAR*, 2011.

[7] J. Ventura, C. Arth, G. Reitmayr, and D. Schmalstieg, "Global localization from monocular SLAM on a mobile phone," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 4, pp. 531–539, 2014.

[8] S. Middelberg, T. Sattler, O. Untzelmann, and L. Kobbelt, "Scalable 6-dof localization on mobile devices," in *ECCV*, 2014.

[9] Y. Li, N. Snavely, and D. P. Huttenlocher, "Location recognition using prioritized feature matching," in *ECCV*, 2010.

[10] S. Cao and N. Snavely, "Minimal scene descriptions from structure from motion models," in *CVPR*, 2014.

[11] S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart, "Get out of my lab: Large-scale, real-time visual-inertial localization," in *Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 13-17, 2015*, 2015.

[12] C. Arth, D. Wagner, M. Klopschitz, A. Irschara, and D. Schmalstieg, "Wide Area Localization on Mobile Phones," in *ISMAR*, 2009.

[13] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[14] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, 2004.

[15] T. Sattler, B. Leibe, and L. Kobbelt, "Improving image-based localization by active correspondence search," in *ECCV*, 2012.

[16] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver, "Google street view: Capturing the world at street level," *Computer*, vol. 43, no. 6, pp. 32–38, 2010.

[17] A. L. Majdik, Y. Albers-Schoenberg, and D. Scaramuzza, "Mav urban localization from google street view data," in *IROS*, 2013.

[18] S. L. H. Liu, T. Mei, J. Luo, H. Li, and S. Li, "Finding perfect rendezvous on the go: Accurate mobile visual localization and its applications to routing," in *ACM Multimedia*, 2012.

[19] A. Taneja, L. Ballan, and M. Pollefeys, "Never get lost again: Vision based navigation using streetview images," in *ACCV*, 2014.

[20] P. Agarwal, W. Burgard, and L. Spinello, "Metric localization using google street view," in *IROS*, 2015.

[21] W. Zhang and J. Kosecka, "Image based localization in urban environments," in *International Symposium on 3D Data Processing, Visualization and Transmission*, 2006.

[22] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *CVPR*, 2006.

[23] A. R. Zamir and M. Shah, "Accurate image localization based on google maps street view," in *ECCV*, 2010.

[24] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof, "From structure-from-motion point clouds to fast location recognition," in *CVPR*, 2009.

[25] T. Sattler, B. Leibe, and L. Kobbelt, "Fast image-based localization using direct 2d-to-3d matching," in *ICCV*, 2011.

[26] A. Babenko and V. S. Lempitsky, "The inverted multi-index," in *CVPR*, 2012.

[27] D. M. Chen, G. Baatz, K. Koser, S. S. Tsai, R. Vedantham, T. Pylvanainen, K. Roimela, X. Chen, J. Bach, M. Pollefeys, B. Girod, and R. Grzeszczuk, "City-scale landmark identification on mobile devices," in *CVPR*, 2011.

[28] H. Jégou and A. Zisserman, "Triangulation embedding and democratic aggregation for image search," in *CVPR*, 2014.

[29] C. Wu, "Towards linear-time incremental structure from motion," in *3DTV*, 2013.

[30] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE TPAMI*, vol. 33, no. 1, pp. 117–128, 2011.

[31] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast search in hamming space with multi-index hashing," in *CVPR*, 2012, pp. 3108–3115.

[32] J. Cheng, C. Leng, J. Wu, H. Cui, and H. Lu, "Fast and accurate image matching with cascade hashing for 3d reconstruction," in *CVPR*, 2014.

[33] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *CVPR*, 2011.

[34] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*, ser. STOC '02, 2002.

[35] B. Zeisl, T. Sattler, and M. Pollefeys, "Camera pose voting for large-scale image-based localization," in *ICCV*, 2015.

[36] O. Chum and J. Matas, "Optimal randomized ransac," *IEEE TPAMI*, vol. 30, no. 8, pp. 1472–1482, 2008.

[37] T. Sattler, T. Weyand, B. Leibe, and L. Kobbelt, "Image retrieval for image-based localization revisited." in *BMVC*, 2012.

[38] Y. Feng, L. Fan, and Y. Wu, "Fast localization in large-scale environments using supervised indexing of binary features," *IEEE Trans. Image Processing*, vol. 25, no. 1, pp. 343–358, 2016.

[39] S. Cao and N. Snavely, "Graph-based discriminative learning for location recognition," in *CVPR*, 2013.

[40] T. Sattler, M. Havlena, F. Radenovic, K. Schindler, and M. Pollefeys, "Hyperpoints and fine vocabularies for large-scale location recognition," in *ICCV*, 2015.