Low Bandwidth Offload for Mobile AR

Puneet Jain
Hewlett Packard Labs
Palo Alto, CA
puneet.jain@hpe.com

Justin Manweiler
IBM Research
Yorktown Heights, NY
jmanweiler@us.ibm.com

Romit Roy Choudhury
University of Illinois
Urbana-Champaign, IL
croy@illinois.edu

ABSTRACT

Environmental fingerprinting has been proposed as a key enabler to immersive, highly contextualized mobile computing applications, especially augmented reality. While fingerprints can be constructed in many domains (e.g., wireless RF, magnetic field, and motion patterns), visual fingerprinting is especially appealing due to the inherent heterogeneity in many indoor spaces. This visual diversity, however, is also its Achilles' heel – matching a unique visual signature against a database of millions requires either impractical computation for a mobile device, or to upload large quantities of visual data for cloud offload. Further, most visual "features" tend to be low entropy – e.g., homogeneous repetitions of floor and ceiling tiles. Our system VisualPrint, proposes a means to offload only the *most distinctive* visual data, that is, only those visual signatures which stand a good chance to yield a unique match. VisualPrint enables cloud-offloaded visual fingerprinting with efficacy comparable to using whole images, but with an order reduction in network transfer.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software

Keywords

Augmented Reality, Offloading, Bandwidth, Latency

1. INTRODUCTION

Real-time augmented reality (AR) has remained an open problem in mobile computing. The problem is challenging because one's camera preview needs to be matched against a pre-existing image database in real-time. This operation is computationally intensive and our smartphones are resource constrained. Therefore, cloud-offloading is considered as an obvious alternative. Not surprising that most research focus toward reducing computation latency. Several designs of faster server-side pipeline [16, 31, 41], light-weight vision algorithms [36], and reducing visual search space [22] have been proposed. Despite many attempts, substantial drawbacks remain: in deployment practicality, in efficacy, or in robustness. Unpredictable end-to-end network latency is one such drawback. Several factors including the distance between the device and cloud, network bandwidth and channel, and sheer data quantity contribute to it. While edge computing [39], data compression [42], and better wireless deployment [35] can alleviate this issue to some extent, due to large volume, have not proven sufficient. This paper identifies a unique fingerprinting opportunity to substantially reduce the visual data transfer, therefore fixing the network latency issue for mobile vision applications.

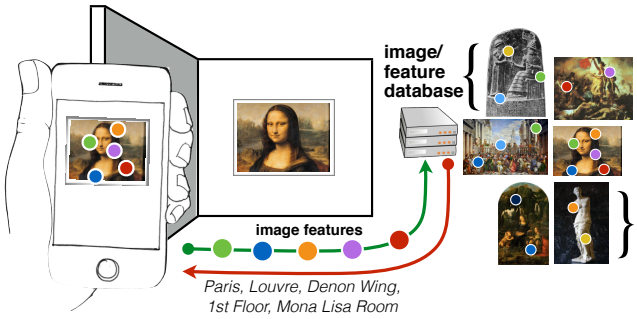

Figure 1: User views scene of interest on smartphone display, image or features shipped to cloud server, cloud returns with 3D AR location context.

Our solution to network latency lies in the observation that indoor spaces naturally lend themselves to high visual diversity. Patterns of color in paintings and photographs on walls, stylized light fixtures, corners on furniture, and even minuscule imperfections in drywall

CoNEXT '16, December 12-15, 2016, Irvine, CA, USA
© 2016 ACM. ISBN 978-1-4503-4292-6/16/12…$15.00
DOI: http://dx.doi.org/10.1145/2999572.2999587

and baseboards collectively create a unique visual "signature" or "fingerprint" of a particular space. Due to the combinatorics, it is highly unlikely that any two places would have a very similar pattern of visual features. Our goal is to create a lightweight mechanism to extract these visual fingerprints at scale. As shown in Figure 1, we want to build a lookup service by which it is possible to instantly map a photo to a unique point in the space; all by using only the essence of the scene – the visual fingerprint. Importantly, the mechanism must be such that the fingerprint size is substantially less than the corresponding visual data.

Practical Challenges of Cloud Offload

While existing approaches for mobile AR might be sufficient for identifying a user's location from a photograph, they would quickly become impractical for continuous, real-time use. A stream of images or video implies a high sustained bandwidth, especially if we take full advantage of modern high-resolution smartphone cameras. As shown in Figure 2, at these resolutions, even 10 frames per second (FPS) requires 2 Mbps using state-of-the-art H264 compression. Worse, such levels of compression result in an almost unusable reduction in the quantity of extractable keypoints (Figure 3). Thus, we would really need lossless compressed frames, such as PNG – at much higher bitrates. On a cellular data connection, even LTE, such a stream is at worst infeasible, or at best, wasteful of the user's data quota and battery life, not to mention finite cellular bandwidth.

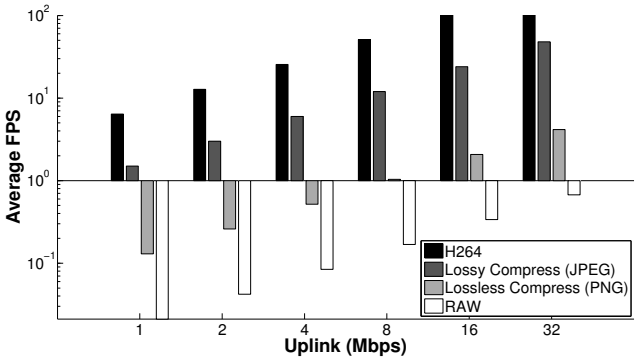

Figure 2: Uplink bandwidth versus sustainable frames per second (FPS), by encoding. Note: log-log scale plot.

It is natural to ask why do mobile AR systems need to process at high frame-rates and why subsampling frames to reduce bandwidth is not sufficient. One may argue that people do not move too fast physically, therefore subsampling should work. However, in mobile AR physical movement is not necessary – one may scan the environment by simply moving hands at fast speed. Moreover, subsampling is related to motion blur. [22] in real-world experiments found majority of frames to be blurred due to motion and shake. Since blurred frames lack ample visual features, they do not result match on the server. Therefore, subsampling can delay upload of a crisp frame for arbitrarily long time and result in perceivable latency on the screen. We discard subsampling to make system more responsive. This also avoids making assumption on the way users use an AR system. Finally, we do not infer whether a frame is blurred or not, because it is non-trivial and adds additional processing latency on the critical path.

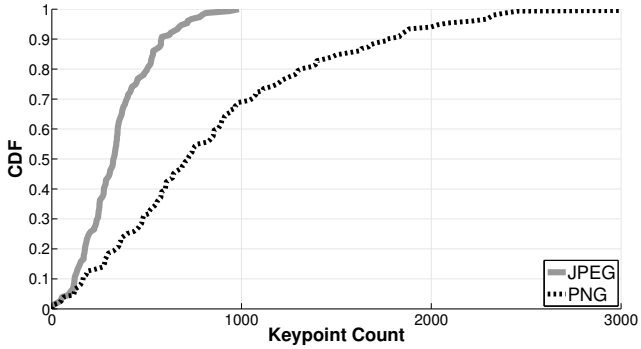

Figure 3: CDF of number of SIFT keypoints, PNG versus JPEG compression (same compression ratio as Figure 2). Under compression, SIFT feature extraction efficacy drops substantially. Peak performance is achieved using lossless compression, such as PNG.

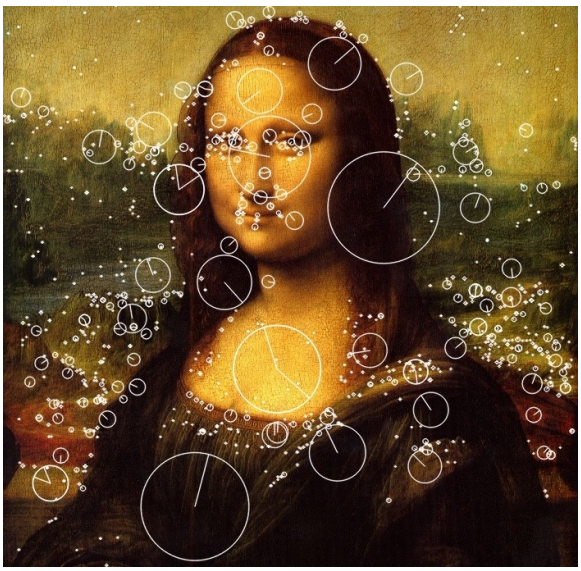

Figure 4: Visualization of SIFT keypoints. For each keypoint, center of circle represents location, radius represents detection scale, and line segment represents direction of orientation.

An alternate approach to solve bandwidth issue could be to just send the visual features. In computer vision, interesting points such as corners are often referred as keypoint. A keypoint is typically represented using

2D pixel coordinate and a multi-dimensional feature description vector. These keypoints can then be used to identify similarity between the two visual scenes. SIFT [32] is a widely popular algorithm used to detect and describe keypoints. Figure 4 shows a visualization of SIFT keypoints. However in our case, sending SIFT keypoints only is also not practical: extracted keypoints typically require *at least* as much space as the image itself. Even after heavy GZIP compression, keypoints require comparable space for most images, and five times more uncompressed (Figure 5). We envision an alternative. What if, instead of shipping the entire image data (or all image keypoints), we ship only those parts which define a unique fingerprint?

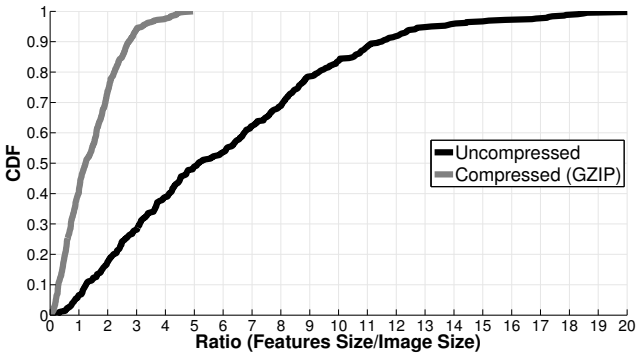

Figure 5: CDF of SIFT feature size (in bytes) ratio to image size. Even after heavy compression, features require comparable space to the original image.

VisualPrint: A Uniqueness "Oracle"

Imagine waking up in a maze of blank, white walls. It is hard to find your way through, given that each area looks the same as the one preceding it. Now imagine you have a marker. By drawing a unique symbol on each wall – equivalent to a few bits of information – you could quickly create a signature to identify where you are at any time. This paper is about finding those few bits of information that naturally occur in real-life environments.

To construct a visual fingerprint, not all parts of an image are equally useful. To reduce cloud-offload bandwidth, we need to distill to only those high-entropy bits. This is analogous to delta (difference) encoding for video compression: all data is thrown away, except the information gain. The challenge here is that the *entropy of any pixel cannot be deduced from strictly local information*. For example, imagine an art gallery. The one-of-a-kind paintings are likely to contain good candidate, highly-unique keypoints. The corners of a checkerboard floor or the regular pattern of ceiling tiles are less so. Many keypoints could be quickly discarded – for example, if two very similar keypoints appear in the same image. However, some might be unique in a particular image, yet common across several. For example, a door knob or light switch might be unique in a room, but repeated in every room of a building. To recognize this global repetition, we need something of a uniqueness "oracle."

We want to create a visual lookup mechanism whereby a smartphone app can capture a photograph or stream of video and quickly identify a handful of keypoints that are highly unique, to define a short fingerprint description. Instead of 2,000 keypoints from a single photograph (not an unusual result from a high-resolution photo using SIFT), we wish to capture the same essence in only 200 – an order of magnitude fewer. The app can then upload this short description ($\approx$ 30KB) of the scene to a cloud service. On that service, we can apply existing techniques of image-based content retrieval (like Google Reverse Image Search or TinEye.com). Instead of similar images, the service replies with metadata, an estimate of location and camera pose.

We call our approach *VisualPrint*. VisualPrint uses only local information (on the smartphone) to leverage a global awareness of which visual data is worthwhile (curated on the server and downloaded to the client in advance). Since this table can be aggressively probabilistic – false positives create a minimal performance penalty – the representation can be extremely compact. Our design ensures that lookups are cheap – constant time per image keypoint – so we can filter all captured image data to only the most essential (the "compact" fingerprint).

In Figure 6, we present our intuition for the opportunity. Most of the 128 dimensions of a SIFT descriptor contain minimal information to differentiate it from all others. Generally, only a few dimensions are required to isolate a descriptor from its closest neighbor. Locality-sensitive hashing (LSH) is a natural choice to re-project a minority of valuable dimensions into a more manageable low-dimensional space – enabling efficient high-dimensional nearest neighbor search.

With feature descriptors distilled using LSH, our key technique applies counting Bloom filters to build small, probabilistic lookup tables. These tables (10s of MBs) are downloaded to a user's device in advance to summarize vast quantities of visual data (1000s of MBs). An image keypoint can be quickly tested against these tables to quantify its uniqueness – how often it appears in other images, globally. Quickly, thousands of keypoints can be sorted according to these uniqueness. The several (200) most unique are then chosen for matching on the cloud – others immediately discarded.

Beyond the core intuition, our end-to-end implemented VisualPrint prototype consists of three major system components: (1) a wardriving app based on Google Project Tango [6] using Simultaneous Localization and Mapping (SLAM) techniques to quickly construct a 3D database of visual features, applicable to indoor environments; (2) a cloud storage and compute service

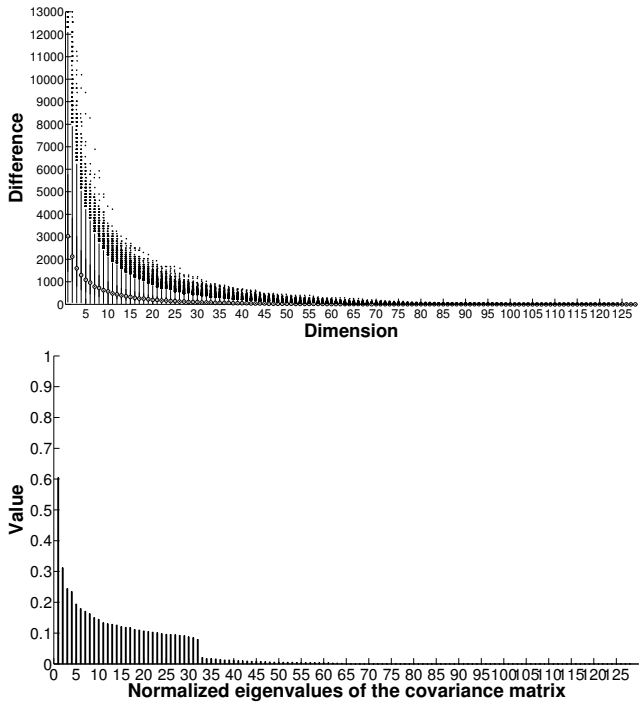

Figure 6: (a) For 500 images, each feature descriptor $\vec{A}$ matched to nearest neighbor in the database $\vec{B}$. Boxplots show `sort_reversed`$[(\vec{A}-\vec{B})^2]$. Few dimensions provide most of the Euclidean distance between $\vec{A}$ and $\vec{B}$. (b) Principal component analysis (PCA) confirms this intuition, as only a few PCA dimensions (far less than 128) are enough to account for the majority of covariance.

to curate this database, by identifying the most unique visual features, and respond to fingerprint queries with a location estimate; and (3) a smartphone app to capture photographs and extract visual features, subselect features by estimated uniqueness to a concise "fingerprint," and perform location lookups on the cloud service.

VisualPrint makes the following contribution:

VisualPrint enables mobile devices to filter visual data by global uniqueness — allowing only the most important bits to be used in a query — and reducing network offload by an order of magnitude.

2. RELATED WORK

VisualPrint builds upon a large body of prior art, especially in the areas of image-based content retrieval, cloud offload support for mobile systems, visual approaches for localization and mapping, and multi-sensory indoor localization. Additionally, the VisualPrint approach can enhance several emerging mobile applications beyond localization.

Image-based Retrieval: Image-based content retrieval has been extensively studied, including consideration for resource constraints applicable to mobile devices. [45] proposes a method to match two uncalibrated stereo images robustly. [13, 32] identify robust features in an image to achieve high accuracy. [9, 37] are binary variants of image features which can be used to perform faster image matching based on Hamming distances. While better feature design improves matching latency between a pair of images – this is insufficient to scale to a database of a large size. Approximate matching schemes such as [15, 41] address this issue to some extent, but memory and computation overheads of indexing hinder their applicability to resource-constrained smartphones. Similarly, running matching algorithms on GPUs [17] is feasible on a server but not readily on smartphones. [20, 25] design locality-sensitive Bloom-filters to address memory limitations of approximate high-dimensional matching. We extend these approaches for VisualPrint, but in a novel application to identify unique image features.

Cloud Offloading for Mobile Applications: Computation offloading to a remote server has been used to overcome computational, memory, and energy limitations of a smartphone [14, 24, 30]. However, real-time offloaded image processing remains challenging for several reasons: upload throughput between the phone and the cloud is often limited (especially over cellular, even LTE uplinks); wireless network latencies between the phone and cloud are unpredictable and can not guarantee a consistent user experience [27]; and high power consumption in continuous uploading defeats a primary incentive for offloading [22, 26]. Hybrid architectures leveraging the phone to do some local computation to (e.g., to pick frames to be uploaded on the cloud) partially address this issue [19]. [39] proposes an alternate architecture of bringing computation power in the proximity of the mobile device.

Visual Simultaneous Localization and Mapping (VSLAM): VSLAM refers to the problem of simultaneously constructing/updating a map while tracking the location of a device, using a camera and other onboard sensors [23]. The Project Tango hardware, used in VisualPrint, is essentially VSLAM put into practice. VSLAM heavily relies on statistical techniques such as Kalman or particles filter to smooth and correct dead reckoning errors of noisy sensor data. Bundle adjustment [43] is used to generate a 3D model of the scene, while simultaneously estimating and correcting optical parameters of the camera. Later, image registration [33] and inertial tracking is used to find an instantaneous localization of the device [38]. VSLAM is implicitly applied in VisualPrint, through our reliance on Google Tango during the initial 3D wardriving phase.

Indoor Localization in Other Domains: WiFi and inertial sensing have been applied for indoor localization [11, 44]. Our approach to visual fingerprinting is complementary to, and can be further enhanced by, these works: location context can be used to reduce image search on the server. However, this context is not ubiquitously available. Thus, we do not assume such a location service, to the benefit of generalizability.

Applications: Crowdsourced video is rising in importance, as evident in the popularity of Youtube, Periscope [8], and Meerkat [7]. Several applications built atop demand real-time understanding of location context in the video streams [2], as can be provided by VisualPrint. Image-derived location context can also immediately benefit applications for augmented reality [3, 5], mobile object recognition [1], movable-camera surveillance [4], life-logging [18], sports video analytics [21], mood-sensing [29], and reaction sensing [12]. We believe that VisualPrint can be productively applied to each of these domains.

3. SYSTEM DESIGN

Figure 7 illustrates VisualPrint's simplified system architecture, consisting of three major components: (1) a client app for commodity off-the-shelf Android smartphones, providing visual fingerprinting and localization; (2) a cloud server module, providing visual fingerprint pre-processing and fingerprint-to-location lookup services; and (3) a wardriving app for Google Tango app, to populate the server with visual fingerprint data, each time a new building is added to the database.

The smartphone client "app" is really a proof of concept. It exists only to demonstrate and test the VisualPrint client library, which could be easily incorporated into any Android app requiring fingerprinting and/or localization services. Or, it might be plausibly incorporated as a cross-app system-level service – enabling camera-based localization applicable indoors when GPS is untenable.

When the library loads the first time, it contacts the VisualPrint cloud service to download up-to-date feature-uniqueness tables (to be refreshed periodically). These are structured as a set of counting Bloom filters, indexed using locality-sensitive hashing. In the next subsection, we describe this in details.

Uniqueness "Oracle" Construction

Primer – Bloom Filters: A Bloom filter is a probabilistic data structure, designed for efficient (constant time and memory) set membership queries. That is, one can look for the presence of a particular (exact) element: range or nearest-neighbor queries are not supported. There is some uncertainty to a lookup, hence considered probabilistic. The filter can determine that an element is either *definitely not* in the set or *may be* in the set. Bloom filters are typically implemented using a fixed-size bit vector. To insert, an element is hashed multiple times using a non-cryptographic hash (typically, a hash is selected for execution speed over cryptographic guarantees, such as Murmur-3). The hash output is taken modulo the size of the bit vector as an index – setting these positions to true. To test set membership, the query element is simply hashed again and the corresponding bits are checked. If all the hashed bits set to true, the element is probabilistically assumed to be present. If a single hashed bit is set to false, the element is definitely not present.

A *counting* Bloom filter is a variant that maintains counters (rather than binary bit flips), enabling accumulation for multiple occurrences of an element. VisualPrint leverages counting Bloom filter with a low saturation point (beyond which additional insertions of the same value have no effect). Once saturated for a particular feature, we can be confident that the corresponding keypoint is fairly common, not useful to our purpose.

Primer – Locality Sensitive Hashing (LSH): LSH techniques widely apply to efficient searching in a high dimensional dataset. The intuition is to reduce dimensions of the data in such a way that similar items map to similar or nearby buckets with high-probability. Assuming the number of buckets is much smaller compared to the original data, an efficient nearest-neighbor lookup can be performed. For VisualPrint, we wish to enable an efficient lookup mechanism of image descriptors. The closeness of two descriptors is measured by the Euclidean distance between them. Due to this requirement, from a variety of LSH variants, E^2LSH [10] is most natural for VisualPrint.

E^2LSH belongs to a family of random projection-based LSH schemes. Specifically, a point to be indexed is projected randomly on several hyperplanes. The expectation is that for two nearby points, most projections will also yield nearby scalar values for each. Therefore, once discretized, most nearby points will be indexed to the same LSH "bucket." What distinguishes E^2LSH from related schemes is in how it selects coefficients to each of the random projection hyperplanes. Critically, it has been shown that when one constructs hyperplanes with coefficients drawn as random samples of a p-stable distribution [15], the projection values preserve the L_p norm. Thus, under a 2-stable distribution, the L_2 norm – the metric of Euclidean distances – is preserved. The Gaussian distribution is 2-stable, and thus an appropriate choice. Hence, E^2LSH (under Gaussian coefficient selection) supports nearest neighbor lookup by Euclidean distance, appropriate for SIFT keypoints.

Locality-Sensitive Bloom Filters

VisualPrint blends the advantages of LSH and Bloom filters. LSH enables imprecise queries based on Euclidean distances; Bloom filters provide a compact representa-

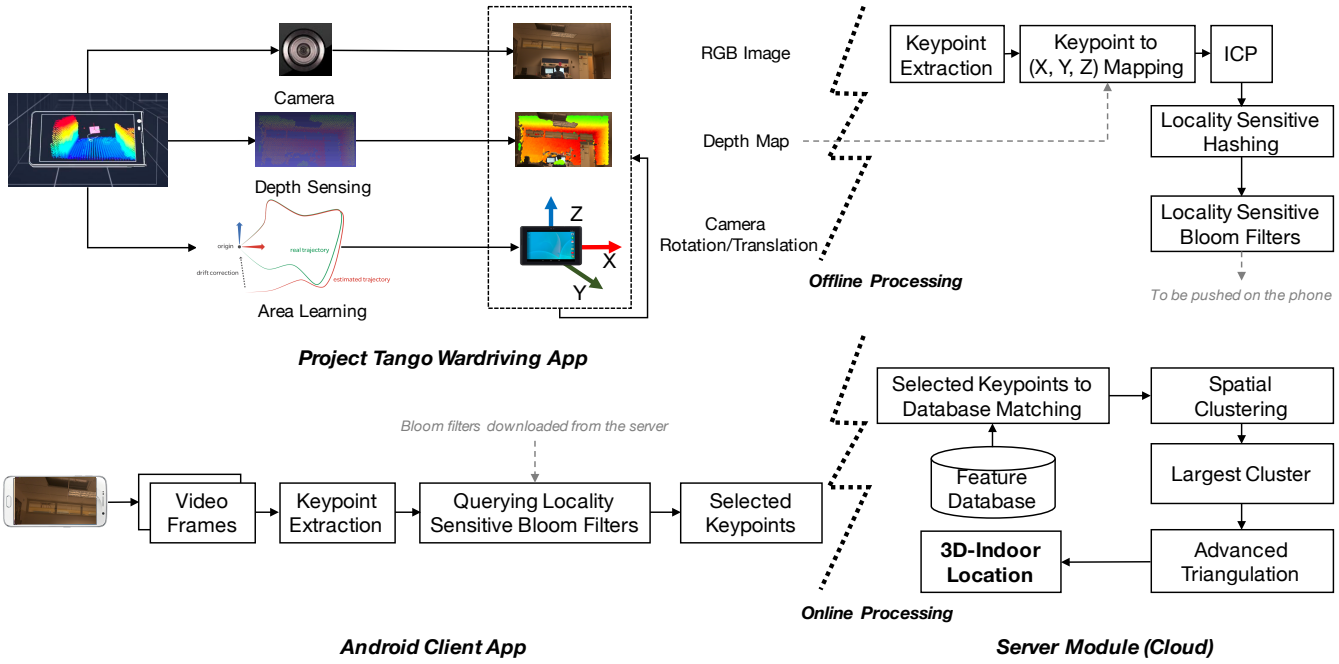

Figure 7: System overview. Top left: wardriving-app based on Google Tango. The output of wardriving app consists of 2D image keypoints with corresponding 3D metadata. Top right: server processes wardriven image 2D/3D data and constructs concise representation in the form of locality-sensitive bloom filters. Bottom left: phone (client) app captures video frames, extracts keypoints, identifies a subset of highly-unique keypoints, and queries to server. Bottom right: server processes and infers 3D location from a client query (2D, comprised of several highly-unique keypoints).

tion. Figure 8 illustrates our locality-sensitive Bloom filter construction. From top to bottom, we begin with a single keypoint descriptor. A SIFT descriptor exists in 128-dimensional space (each dimension being a one-byte integer value). The first step is to take M random projections of the 128-dimension descriptor. Each singular projection yields a single scalar value, with M projections yielding a M-dimensional vector. Each projection is quantized into a discrete space based on width parameter W. We construct L such M-dimensional vectors to create L LSH "buckets." Each of the $M \times L$ randomly-chosen projections is held constant for the life of the data structure. For each of the L buckets, we apply one of L cryptographic hash functions g_i from the same family (Murmur-3) on the M-dimensional vector. The hash output provides K indices into a counting Bloom filter (K for each of the L LSH buckets). For each index, we increase the corresponding count by one. Each bit index counter is represented in 10 bits, for a count saturation (maximum count that can be represented) of 1024. Beyond 1024, we treat a keypoint as not unique enough for consideration. We empirically optimize the various parameters to LSH and Bloom filters from a generic image keypoint dataset: $L = 10$, $M = 7$, $W = 500$, and $K = 8$.

Challenge, False Positives: Three sources of false positives exist in our scheme: (1) Bloom filters, (2) LSH, and (3) the interplay of Bloom filters and LSH. The probability of Bloom filter false positive is tunable – trading off memory consumption. We tune Bloom filters to support up to 2.5M unique feature vectors with less than 1% false positives. LSH false positives are mitigated through empirical tuning of L, M, and W. After tuning, we observe few false positives from LSH alone. However, the interplay of LSH and Bloom is more problematic. The primary cause is due to the quantization of features (as tuned using LSH parameter W). Quantization must be coarse enough to capture similarity of two nearby feature descriptors. However, when W is selected to be more coarse, there are fewer unique "bins." Thus, the Bloom filter is used less uniformly – bits are only flipped to represent this smaller number of bins. After many insertions, "hotspots" in the Bloom filter may result. We address these in two ways. First, we choose a relatively higher saturation point for counting Bloom filters: we use 10 bits per Bloom filter index, for saturation at 1024, when fewer would otherwise be required. Secondly, we leverage an established technique from prior art – a *verification* Bloom filter. For each insertion to a primary Bloom filter, a second insertion is performed into the verification Bloom filter. However, instead of hashing the original data, we hash the bit positions of the insertions to the primary Bloom filter. During the query, a positive result is returned if and

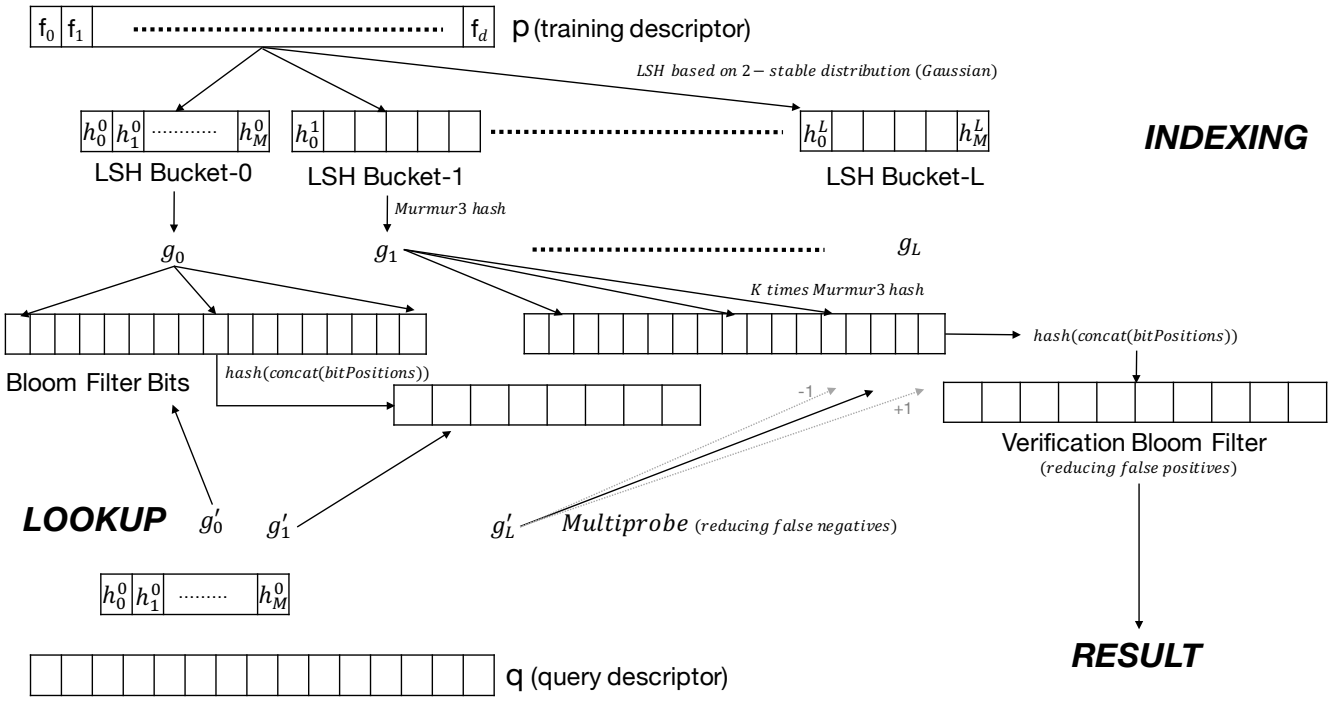

Figure 8: Locality-sensitive Bloom filter "oracle" construction. From top (indexing): image keypoint descriptor passes first through a locality-sensitive, then secondly, a cryptographic hash, yielding several bit indices into counting Bloom filters (middle). From bottom left (lookup): keypoint descriptor checked against Bloom filters. "Multiprobe" protects against off-by-one false negatives. Non-counting verification Bloom filter (far right) protects against false positives.

only if a positive match is found in both the primary and verification Bloom filters.

Challenge, False Negatives: False negatives may occur in LSH, due to the quantization boundaries. Irrespective of the choice of quantization parameter W, it is possible that two nearby feature descriptors would be mapped to different quantization buckets (if the quantization boundary happens to fall in between). Fortunately, the error can be at most a single quantization bucket. Borrowing a technique from [34], we perform "multi-probe" checks into adjacent quantization buckets to reduce these cases. During retrieval, we consider off-by-one partial matches in the Bloom filter – those with $K-1$ of K bits matching. This requires a total of $2K+1$ Bloom filter checks – easily accommodated, as each Bloom filter test is extremely fast. If such a partial match is found, we declare this as a *potential* false negative case – accepting it a "pass" of the primary Bloom filter. The verification Bloom filter provides the final check. Given that multi-probes increase the probability of false positives, the verification Bloom filter becomes all the more crucial to our scheme.

Keypoint-to-3D Position Wardriving

The core data structure of the VisualPrint cloud service is a large-scale lookup table – mapping image features to their corresponding 3D positions. Before VisualPrint can be used to *retrieve* visual fingerprints in an indoor space, the environment must be wardriven – populating this table. We build a visual-wardriving app for Google Tango [6] hardware. While the user walks around a building, the app captures photographs (from a traditional RGB sensor), a depth map (from an integrated IR depth sensor), and the location of the user (relative to the start position, tracked through VSLAM).

During the wardriving phase, the VisualPrint app for Tango is used to collect a series of snapshots with corresponding metadata: the six degree of freedom (6-DoF) pose of the device, an RGB image (from which keypoints will be extracted), and a lower resolution depth map of the corresponding view (from an embedded IR-based depth sensor). The 6-DoF pose is relative to the initial position (where the app was initialized). It includes three dimensions of translation in (x, y, z) and three dimensions of device rotation/orientation (yaw, pitch, roll). To wardrive a venue, a user needs to walk throughout the indoor space to be captured. This process is manual (the user must physically

walk), but could be automated using area-covering robots like Roomba. During collection, the Tango app pushes images, depth context, and positioning metadata to the VisualPrint cloud service. As described in the next subsection, the VisualPrint cloud service pre-processes this data to establish mappings between unique image keypoints and their corresponding 3D locations. The number of such mappings in a building can be exceedingly large from hundreds of thousands to millions.

Figure 9: Tango connected with phone using double-backed tape and foam board.

The Tango prototype does not immediately provide all of the required data for VisualPrint. Practical limitations, due to Tango's unpolished beta status, complicate our implementation. Principally, when leveraging Tango visual-SLAM and depth mapping functionality, all visual sensing hardware is locked, and cannot be otherwise accessed by third party Tango (a form of Android) apps. We leverage area learning and depth perception to achieve an understanding of where a depth map exists in relation to others in the same indoor space. Unfortunately, Tango blocks access to the RGB camera when using these services. So, it is less straightforward to map a 3D depth estimate, projected into a shared indoor space via area learning, to a portion of an RGB image frame. Note that much of Tango exists in closed source, so we were not able to circumvent the hardware protections. While it is possible to grab the RGB screen buffer, this is too low resolution for our needs. Instead, we used a low-tech workaround: we taped a second Android device to the back of our Tango and created a simple network API to synchronously grab image frames in concert with Tango-recorded depth maps (see Figure 9). Of course, limited RGB access is not fundamental, and we expect this shortcoming to be addressed in the future. Figure 10 shows how we merge Tango-acquired depth data into RGB.

Challenge, Positioning Error and Uniqueness: Unfortunately, but unsurprisingly, there are inaccuracies in Tango's visual-SLAM positioning. As the user walks around, location estimates naturally reflect some amount of drift from true positions. While small

Figure 10: Tango RGB+depth: (a) original RGB image; (b) heat map of depth from observer, red is farther away.

amounts of positioning error might not cause undue harm to many potential Tango applications, for VisualPrint, this can be problematic. If two photographs appear to be of different objects (due to erroneous location estimates), truly-unique keypoints may appear to be repeated – a false negative for our uniqueness tracking. We resolve this issue by post-processing Tango's depth map output – a 3D *point cloud*. We apply iterative closest point (ICP) heuristics to merge Tango 3D depth maps (from separate snapshots) into a single coherent point cloud for the entire indoor space. Only from this converged, comprehensive depth map we can be sure that two keypoints reflect truly independent locations. This post processing also has a secondary advantage of reducing some error in our location estimates.

Cloud Processing and 3D Positioning

VisualPrint cloud service accepts keypoint-to-3D position mappings from the Tango wardriving app. As they are received, the service updates two data structures: (1) the keypoint-to-3D position lookup tables and (2) LSH-indexed Boom filters for keypoint uniqueness.

The lookup table is really a large-scale image-based content retrieval table, as one might apply for reverse image search. For our prototype, we apply standard LSH. Rather than a reference to an image or other content, the LSH table stores the 3D position of the keypoint. The service accepts queries to this table in the form of keypoint-plus-2D coordinate pairs (the x, y coordinate of the keypoint in the camera frame). The service performs an LSH lookup on each keypoint to retrieve its 3D position. It then applies a localization step, described next, and returns the estimated client location and pose.

The second data structure, the LSH-indexed Bloom filter, is the same exact data structure downloaded by VisualPrint clients as the uniqueness oracle. As new keypoint-to-3D mapping arrives at the VisualPrint cloud service, we simply perform an LSH lookup followed by insertion to the primary and verification Bloom filters. With this simple design, new keypoint-to-location mappings can be incorporated continuously, in constant time and memory.

VisualPrint Application: Localization

For each client localization query, the cloud service retrieves the 3D position associated with each image keypoint. This is a straightforward query to a standard LSH-indexed lookup table. However, the retrieved 3D positions are only the first step towards localizing the client user. When combined with the 2D data of how the 3D position is perceived by the client device, it is possible to infer the clients 6-DoF camera pose, including translation (x, y, z) and orientation (yaw, pitch, roll). In the final effect, we achieve a similar positioning fidelity as Google Tango, but *with only a standard, 2D, RGB camera*. Unlike several approaches in prior art [28, 40, 44], VisualPrint achieves instantaneous localization – continuous tracking not required.

From a single image I, a client queries VisualPrint with a set of keypoints K. From the LSH-indexed lookup table, VisualPrint finds $|K| \cdot n$ nearest-neighbor matches (since there is uncertainty in the match), and accordingly $|K| \cdot n$ 3D points. On these points, VisualPrint applies spatial clustering to filter down to only those 3D points in the largest cluster P, discarding others. $K' \subseteq K$ keypoints remain, for $|P|$ 3D points, where $|K'| \leq |P| \leq |K'| \cdot n$. For each 3D point $p \in P$, there is a single corresponding keypoint $k^p \in K'$. We also know that k has a corresponding 2D pixel coordinate (x_p, y_p) in the original client query image I.

Figure 11 illustrates the spatial relationship between a VisualPrint client camera capturing an image I and the known 3D points corresponding to keypoints found in I. Assume the client's camera exists in 3D space at point A. B is at the center-right (along the edge) and C is at the center of the image. Therefore, $\angle CAB$ is half of the horizontal field-of-view of the camera. The diagram shows how one can calculate angles from A, w.r.t to the center C. Let (p_x, p_y) be the pixel coordinates of a keypoint P and γ_x be the angle $\angle CAP$ (P's projection on the x-axis). Using known 2D pixel coordinates $B = (width, height/2)$ and $C = (width/2, height/2)$, and the camera's field of view (FoV_h, FoV_v), we may estimate γ_x and γ_y.

Similarly, for a pair of keypoints P_0 and P_1, we may compute x-axis angles $\gamma_x(P_0)$, $\gamma_x(P_1)$, and y-axis angles $\gamma_y(P_0)$, $\gamma_y(P_1)$. Now, the x-axis angle between P_0 and P_1 is $\gamma_x(P_0) + \gamma_x(P_1)$, if P_0 and P_1 fall on opposite sides of C, or $|\gamma_x(P_0) - \gamma_x(P_1)|$ if they are on the same side. The same method applies for the y-axis angle.

From several such point pairs (in the same image), VisualPrint estimates the client's location in 3D. VisualPrint models this geometry as a nonlinear optimization. Figure 12 presents the objective function, constraints, variables, and parameters of the optimization.

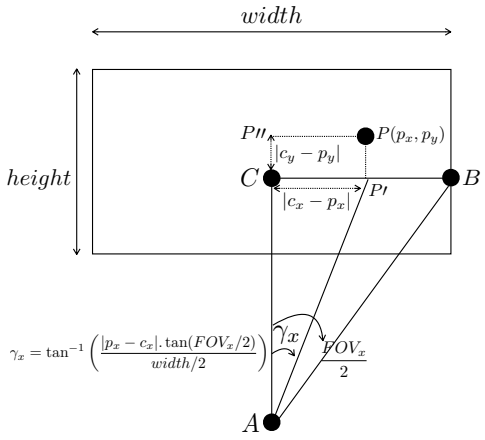

Figure 11: Geometry of angular separation from a keypoint P and the center of the screen C, respective to the 3D location of the client's camera at A. The angle $\angle CAP$ (called γ_x) is of interest – the angle from center to a projection of P along the x-axis. All expressions treat the distance of line segments $\overline{AC}$, $\overline{AP}$, and $\overline{AB}$ as unknown.

Client Android App

The client app serves as a proof of concept for our client library, intended to bring visual fingerprint matching to any application. The app demonstrates systems capabilities for realtime localization, based on visual features.

When launched for the first time, the app retrieves the current state of the uniqueness LSH Bloom filter ("oracle") from the cloud server. In our testing, this download is approximately 10MB, but may be more or less, varying with the number of unique keypoints on the server. Although the Bloom filters are of fixed size, we compress them with GZIP for efficient retrieval, and compressibility reduces as the Bloom filter becomes more saturated. The app periodically refreshes its copy of the Bloom filter to stay current with the server. We could reduce data transfer by sending only a compressed bitmask representing the diff between versions (not yet implemented).

Once ready, the app activates the smartphone camera. The app continuously extracts video frames. It performs a quick check on each frame to detect blur (often due to quick motion), discarding such frames. It also rejects frames when processing falls behind the realtime stream. That is, the app only processes extremely recent frames. For each frame passing all checks, the app extracts SIFT keypoints – up to several thousand per frame, depending on image content. The app then queries the uniqueness LSH Bloom filter for each keypoint. Recall that the primary Bloom filter is of the counting variety. Thus, uniqueness counts (up to the saturation point of 1024) yield a partial ordering, ranking keypoints from highly unique to common. The app then uploads several of the most unique keypoint de-

$$
\begin{aligned}
&\textbf{Minimize} && \sum_{\forall i<j \in 1\ldots K} E^x_{ij} + E^y_{ij} \\
&\textbf{Subject to} && \\
\gamma^x_{ij} &= && \gamma(p^x_i, C_x, F_h, W) - \gamma(p^x_j, C_x, F_h, W) \\
\gamma^y_{ij} &= && \gamma(p^y_i, C_y, F_v, H) - \gamma(p^y_j, C_y, F_v, H) \\
\cos(\gamma^x_{ij} + E^x_{ij}) &= && \frac{d(x,z,x_i,z_i) + d(x,z,x_j,z_j) - d(x_i,z_i,x_j,z_j)}{2\sqrt{d(x,z,x_i,z_i)}\sqrt{d(x,z,x_j,z_j)}} \\
\cos(\gamma^y_{ij} + E^y_{ij}) &= && \frac{d(y,z,y_i,z_i) + d(y,z,y_j,z_j) - d(y_i,z_i,y_j,z_j)}{2\sqrt{d(y,z,y_i,z_i)}\sqrt{d((y,z,y_j,z_j)}} \\
\gamma(p,C,F,S) &= && \tan^{-1}\left(\frac{|p-C|.\tan(F/2)}{S/2}\right) \\
d(x_1,y_1,x_2,y_2) &= && (x_1-x_2)^2 + (y_1-y_2)^2 \\
&\textbf{Solving for} && \\
\forall i<j \in 1\ldots K &: && x, y, z, E^x_{ij}, E^y_{ij} \\
&\textbf{With parameters} && \\
\forall i<j \in 1\ldots K &: && p^x_i, p^y_i, p^x_j, p^y_j, x_i, y_i, z_i, x_j, y_j, z_j
\end{aligned}
$$

Left Side	Interpretation
i, j	Keypoint indices, $i < j$.
γ^x_{ij}	Angle from keypoint i to j, projected on the x-axis.
$\cos(\gamma^x_{ij} + E^x_{ij})$	Geometric constraint derived from law of cosines.
$\gamma(p, C, F, S)$	Angle from client at A to keypoint, as a function of pixel location, image center, field of view, and side length (width or height).
$d(x_1, y_1, x_2, y_2)$	Square of Euclidean distance $(x_1, y_1) \leftrightarrow (x_2, y_2)$.
(x, y, z)	Optimized 3D coordinate of client (position A).
$E^x_{ij} + E^y_{ij}$	Minimized angular error between keypoints i, j.
(x_i, y_i, z_i)	Known 3D position of i.
(p^x_i, p^y_i)	Viewed 2D position of i.

Figure 12: Nonlinear optimization to estimate client camera position (x, y, z), as labeled "A" in the geometry shown by Figure 11. The optimization objective finds x, y, z where perceived angles $\gamma^x_{i,j}$ (on the X/Z plane) and $\gamma^y_{i,j}$ (Y/Z plane) between any keypoint pair i, j are most consistent with their corresponding known 3D locations at $x_i, y_i, z_i, x_j, y_j, z_j$. The objective minimizes summed residual error for all keypoint pairs.

scriptors (experiments with 200 and 500 are presented in the next section).

Upon receiving these unique keypoint descriptors, the cloud service performs the keypoint-to-3D client localization, returning the 3D camera pose estimate to the client. The app then displays the estimate on screen.

4. EVALUATION

Our evaluation considers these research questions:

1. *How does VisualPrint's accuracy compare with other image/keypoint matching schemes?* In particular, we compare VisualPrint to a BruteForce Euclidean distance match (implemented on GPU), a traditional LSH implementation (as would be typical of a large-scale reverse image search), and Random where we uniformly subsample image keypoints.

2. *Are client-side overheads acceptable?* How does VisualPrint reduce data upload compared to conventional cloud offload? Memory? Computation? and Energy?

3. *How effective is VisualPrint in enabling our example application: client localization?*

Matching Accuracy

We photographed 100 non-overlapping scenes across the three floors of our research facility, the Coordinated Science Lab (CSL). Each CSL floor is 50m $\times$ 10m in dimension. We captured each scene with a single image. We also capture 400 additional distractor images. These images consist of ceiling, floor, name-plates, furniture, etc. in the building. Since these images naturally contain repeated patterns, they are likely to have repeated visual features as well. For each of the 500 images, we use OpenCV's default SIFT implementation to extract features. Across these 500 images, the database contains a total of ≈2.5M feature descriptors (each descriptor is 128 bytes).

The query database consists of five additional photographs of each scene in the database. We systematically captured these five photograph from substantially different angles. As SIFT matching efficacy degrades substantially with angular separation, these diverse angles are intended to challenge all matching schemes.

The average number of features per query image was 3,500. From these keypoints, we evaluate VisualPrint across five different system regimes. Random picks 500 random keypoints from the query image and uploads them to the server for matching. Random can be understood as lower-bound on VisualPrint's performance (one with no intelligence in feature subselection). For VisualPrint-200 and VisualPrint-500, our complete system selects the 200 and 500 most unique keypoints. LSH applies the reference E^2LSH locality-sensitive hashing implementation for nearest-neighbor search. BruteForce finds the "optimal" nearest neighbor match. Note that unlike Random and VisualPrint-200/500, LSH and BruteForce use all image keypoints. Note that while it may be tempting to consider BruteForce an upper

bound on accuracy, it can be misled by homogeneous keypoints across images.

Metrics: Let Y be the set of all processed query frames. Let $V \subset Y$ be the set of frames containing scene k. Let $P \subset Y$ be the set of frames identified by our system as capturing k. Then, $V \setminus P$ denotes our system's false negative predictions, $P \setminus V$ denotes false positive predictions, and $Y \setminus V$ denotes the set of frames which do not contain scene k. *We evaluate VisualPrint's prediction efficacy by the following metrics of information retrieval:*

Precision$_k$ $= |V \cap P|/|P|$
i.e., among all frames that VisualPrint identifies as capturing scene k, what fraction truly captures k.

Recall$_k$ $|V \cap P|/|V|$
i.e., among all frames that truly have capture scene k, what fraction was identified by VisualPrint.

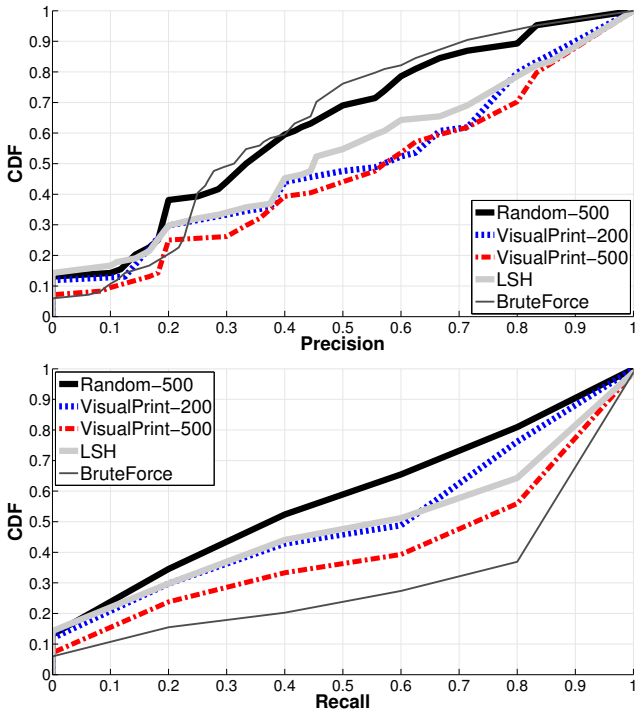

Figure 13: For precision/recall, our goal was for VisualPrint to be roughly comparable with a BruteForce approach — *true gains are measured in substantial network transfer savings*. Incidentally, VisualPrint's precision improves slightly over LSH and BruteForce, as VisualPrint automatically eliminates useless, distracting non-unique keypoints (homogeneous repetitions can lead to false positive matches). Recall suffers slightly against BruteForce (some additional false negatives due to some discarded-but-useful keypoints), but still surpasses LSH (the most realistic server-side comparison) and random keypoint subselection (strawman baseline).

Figure 13 presents CDF of (a) precision and (b) recall results. The precision and recall values are calculated per scene using the above definition. For example, 50% of the test scenes achieve precision $>$ 58% and recall $>$ 61% incase of VisualPrint-200. Unsurprisingly, Random performs poorly in both precision and recall. The results for BruteForce are more nuanced. BruteForce precision is poor, likely due to confusion among homogeneous keypoints. BruteForce recall is most favorable, likely some valuable keypoints are missed by VisualPrint, or it is able to differentiate even among homogeneous keypoints based on their relative quantities (the entire distribution is scored in the BruteForce heuristic). LSH performs comparably to VisualPrint-200, but is inferior to VisualPrint-500 – likely due to the same confusion-under-homogeneity afflicting BruteForce. Note that sole purpose of the precision and recall results is to make sure that VisualPrint does not compromise accuracy to gain bandwidth savings.

Client Overheads

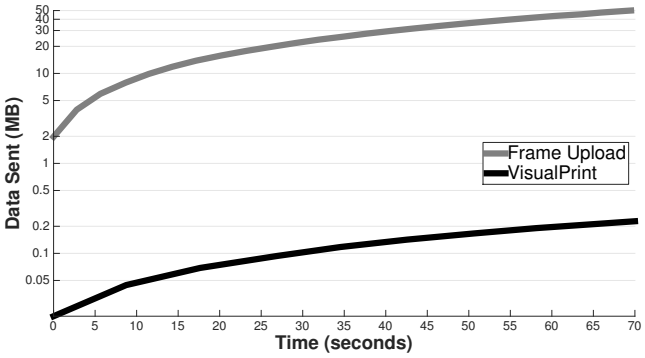

Figure 14: Cumulative data upload by execution time. VisualPrint reduces data consumption by at least one order of magnitude compared to raw frames (sending all raw keypoints would be more than frames, as shown in Figure 5).

Data Upload Savings: Figure 14 compares upload data for VisualPrint versus whole frame upload. Note that, as shown in Figure 5, uploading keypoints (even compressed) does not save any upload bandwidth since their size is comparable (often more) to that of the original frame.

Client Storage and Memory Overhead: Figure 15 compares the client-side storage/memory footprints of Random, VisualPrint, LSH, and BruteForce. Random requires effectively no memory (there is no index). VisualPrint has a substantial memory footprint, due to the Bloom filters. LSH has an extremely large memory footprint, much larger than the input data, due to multiple replications supporting multiple projections. BruteForce has a memory footprint comparable to the database size – here this is implemented on GPU as a SIMD matching

(CPU matching would be nonsensical due to extreme latency). It would be possible to reduce BruteForce memory footprint by loading the database in parts, but with a trade-off of excessive loading latency, rather than a one-time cost.

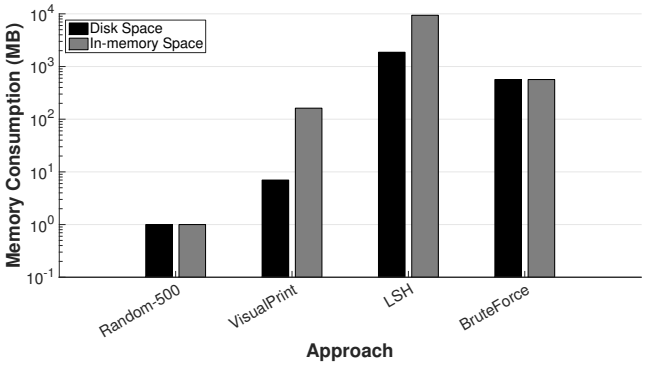

Figure 15: Client disk/memory consumption. VisualPrint uses substantially less memory than conventional LSH, which caches the entire image database. VisualPrint requires somewhat more memory than random keypoint subselection to maintain Bloom filters in memory. BruteForce memory consumption reflects loading all database keypoints into memory for a GPU SIMD parallel execution.

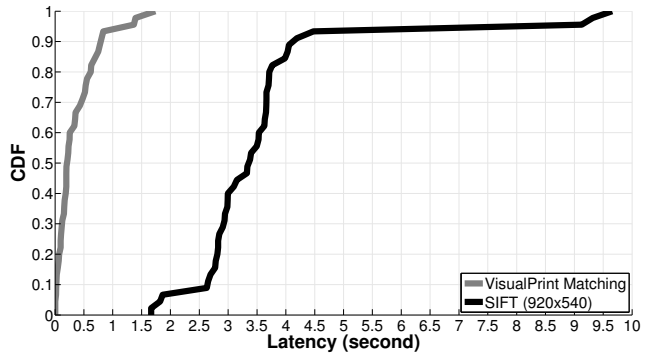

Figure 16: Computational overheads on Galaxy S6. VisualPrint requires an order of magnitude less computation than SIFT feature extraction. Ample room for end-to-end improvement by running SIFT on GPU or a coprocessor.

Client Computational Latency: Figure 16 presents a CDF of VisualPrint computational latency, as compared to SIFT feature extraction. Principally, VisualPrint latency is due to the LSH Bloom filter lookups for each keypoint descriptor, plus sorting. SIFT delay dominates over VisualPrint computation overhead.

Energy: All non-energy experiments were performed on the latest-generation Samsung Galaxy S6 phone. However, as the battery compartment of the S6 is not removable, we revert to the prior generation S5 phone for energy measurements (only). The measurement setup is shown in Figure 17. We use a Monsoon power monitor to measure the average power consumption of the phone. No additional applications apart from VisualPrint and required background services were running during the experiment. The phone was put on airplane mode to avoid cellular communications and WiFi was turned on for VisualPrint tests (only). As energy consumption is impacted by the number of extracted keypoints, lighting conditions and scene were kept same across all schemes. See Figure 18.

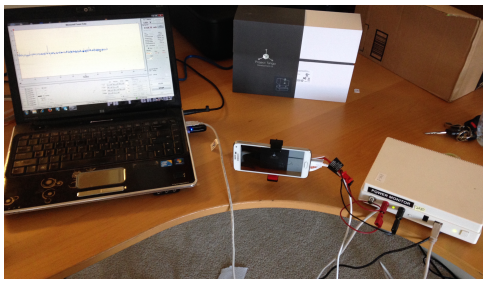

Figure 17: Experimental setup for energy measurement. Monsoon power meter provides current to Galaxy S5 phone in place of battery. Measurement at 5,000 Hz.

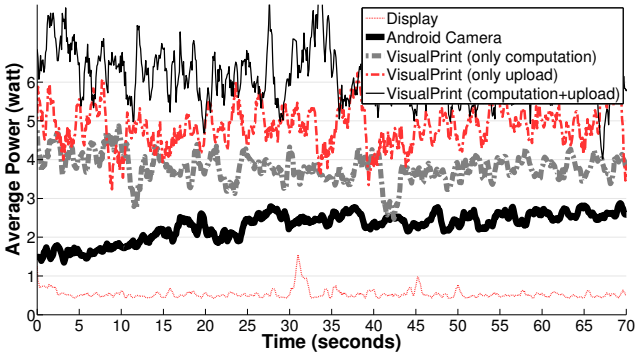

Figure 18: Energy. Including display and camera, complete VisualPrint ≈ 6.5 W. Not shown: whole-frame cloud offload ≈ 4.9 W. Energy consumption of our current prototype is unacceptable but avenues for substantial improvement exist. SIFT and Bloom filter operations can be highly-parallelized using GPU. When using GPU, the processor is occupied for short time, can lead to significant energy savings [36].

Localization

We test VisualPrint's localization performance in three indoor environments: an office space, employee cafeteria, and grocery store. The office environment consisted of similar looking cubicles, a kitchen area, and lounge. In the office, we wardrive a rectangular space

of approximate dimensions 50m $\times$ 20m. The cafeteria consists of many identical chairs and tables, food menu boards, foodservice and checkout counters, and registers in a 50m $\times$ 15m room. The grocery store (80m $\times$ 50m) had standard aisle based layout, filled with household items.

During wardriving, the Tango was mounted with a phone (Galaxy Note 3) as shown in Figure 9. During query phase, a different phone (Galaxy S6) was used to take photographs, in arbitrary orientations. We use Tango to collect a "ground-truth" reference to compare query results. Wardriving and query data were collected along similar routes.

Figure 19 shows cumulative localization errors. Figure 20 isolates those errors to X, Y, and, Z directions. While localization is generally quite accurate, some failure cases do occur. We believe that most cases of substandard performance may be attributed to suboptimality at local minima (we solve the localization optimization using a time-bounded differential evolution).

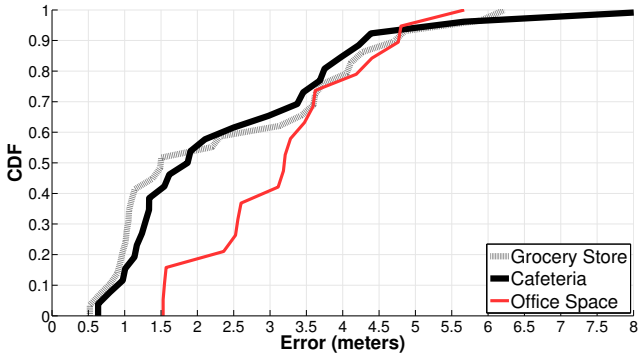

Figure 19: CDF of 3D localization error. Graph shows 3D distance from "ground truth" (as estimated from Google Tango), to VisualPrint's estimate on Galaxy S6.

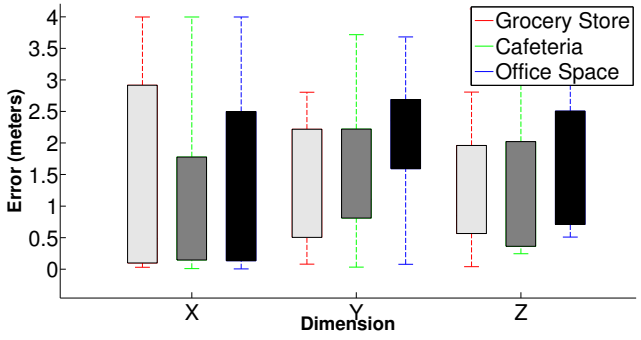

Figure 20: Localization accuracy by dimension. Localization on the horizontal X/Y plane (parallel to floor and ceiling) is more accurate than vertically (changes in elevation). This is expected, as the wardriving motion is also along the X/Y plane.

Evaluation Takeaways

The key findings of our evaluation are as follows:

1. VisualPrint precision and recall are roughly comparable to LSH, the parent scheme on which it is based (Figure 13).
2. VisualPrint requires $1/10^{th}$ bandwidth of whole frame uploads, only 51.2 KB oppose to 523 KB (Figure 14).
3. Based on our 2.5M descriptor database, the VisualPrint client uses 10.5 MB disk space for Bloom filters, stored compressed – $1/124^{th}$ of the server side LSH indices compressed (1.3 GB).
4. Based on our 2.5M descriptor database, the VisualPrint client requires 162 MB in RAM uncompressed – $1/58^{th}$ of the LSH indices uncompressed (9.4 GB), cached in RAM by the server.
5. Client compute latency consists of SIFT extraction (3300 ms at median) and Bloom filter lookups (217 ms at median) for a total median compute latency of 3547 ms, dominated by SIFT (Figure 16). However, using [36], upto 7x latency reduction is possible by running SIFT extraction on a mobile GPU or a coprocessor.
6. Client energy consumption is substantial (6.5 W), principally due to camera and local computation, especially SIFT feature extraction (Figure 18). Again, using [36], upto 87% of the SIFT computation energy can be saved.
7. Results show VisualPrint has a median 3D localization error of 2.5 m (Figure 19).

5. LIMITATIONS AND DISCUSSION

Several practical limitations are apparent from our evaluation of VisualPrint. Principally, SIFT keypoint extraction incurs substantial compute overhead, yielding a large energy impact. Secondly, any visual fingerprinting scheme will incur a large energy cost, simply from turning on the camera. Fundamentally, VisualPrint could never be energy-competitive with lighter-weight sensory approaches.

VisualPrint should also not be taken as a replacement for traditional localization schemes. Rather it is an attempt to leverage visual data when it is anyways available (i.e., the camera is already turned on). Due to its visual nature, VisualPrint cannot always return precise user location. It can fail due to: (1) a lack of ample features in the query image, such as hallway with white walls; (2) insufficient wardriving – the environment at a location may not be well fingerprinted, causing image matching to fail; (3) false positives in keypoint matching – some environmental repetition might not be captured during wardriving; and (4) dead reckoning errors during wardriving – Tango uses IR depth sensing

and IMU to track user location, which are prone to errors (especially, in environments with natural sunlight or electromagnetic machinery).

From a cursory look, VisualPrint appears to be SIFT specific; mostly due to the techniques based on LSH and Bloom filter. However, this is not a fundamental systems limitation. Keypoint detection and description are two separate stages in computer vision. One can use any keypoint detection algorithm (e.g., SURF [13]) with another integer keypoint description algorithm without modification in the system pipeline. In the cases where floating point descriptors are desired, one can scale floating point descriptor elements to the integers by multiplying with an order of required decimal precision.

6. CONCLUSION

Environmental fingerprint is a powerful primitive for mobile vision and augmented reality applications. However, the identification of visual fingerprints is challenging from a perspective of data storage, indexing, and retrieval. Cloud offload presents an obvious-if-challenging solution. VisualPrint enables selective cloud offload of only those visual fingerprints that carry the greatest global entropy – those that are highly unique, particular to the user's true location. Beyond visual fingerprints, we believe that the VisualPrint approach can be productively reapplied in other high-dimensional sensory domains, such as wireless RF, auditory, and hyperspectral signatures. We consider these in our ongoing research.

7. ACKNOWLEDGMENTS

We sincerely thank our many volunteers, Dr. Ganesh Ananthanarayanan our shepherd, as well the anonymous reviewers for their invaluable feedback. We are also grateful to Intel, Google, and NSF for partially funding this research through the grant NSF 1430064.

8. REFERENCES

[1] Amazon fire phone. https://developer.amazon.com/public/solutions/devices/fire-phone.

[2] Dextro stream. https://www.dextro.co/.

[3] Microsoft hololens. https://www.microsoft.com/microsoft-hololens/en-us.

[4] Nest cam. https://nest.com/camera/meet-nest-cam/.

[5] Oculus vr. https://www.oculus.com/en-us/.

[6] Project tango. https://www.google.com/atap/projecttango/.

[7] Meerkat: Live stream video. http://meerkatapp.co/.

[8] Periscope: Explore the world through someone else's eyes. https://www.periscope.tv/.

[9] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. Kaze features. In *ECCV*. Springer, 2012.

[10] A. Andoni and P. Indyk. E2lsh: Exact euclidean locality-sensitive hashing, 2004.

[11] P. Bahl and V. N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM*. IEEE, 2000.

[12] X. Bao, S. Fan, A. Varshavsky, K. Li, and R. Roy Choudhury. Your reactions suggest you liked the movie: Automatic content rating via reaction sensing. In *UbiComp*. ACM, 2013.

[13] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[14] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: making smartphones last longer with code offload. In *MobiSys*. ACM, 2010.

[15] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004.

[16] J. Fung and S. Mann. Openvidia: parallel gpu computer vision. In *MM*. ACM, 2005.

[17] M. Gong and Y.-H. Yang. Near real-time reliable stereo matching using programmable graphics hardware. In *CVPR*. IEEE, 2005.

[18] S. Hodges, L. Williams, E. Berry, S. Izadi, J. Srinivasan, A. Butler, G. Smyth, N. Kapur, and K. Wood. Sensecam: A retrospective memory aid. In *UbiComp*. Springer, 2006.

[19] W. Hu, B. Amos, Z. Chen, K. Ha, W. Richter, P. Pillai, B. Gilbert, J. Harkes, and M. Satyanarayanan. The case for offload shaping. In *HotMobile*. ACM, 2015.

[20] Y. Hua, B. Xiao, B. Veeravalli, and D. Feng. Locality-sensitive bloom filter for approximate membership query. *Computers, IEEE Transactions on*, 61(6):817–830, 2012.

[21] P. Jain, J. Manweiler, A. Acharya, and K. Beaty. Focus: clustering crowdsourced videos by line-of-sight. In *SenSys*, page 8. ACM, 2013.

[22] P. Jain, J. Manweiler, and R. Roy Choudhury. Overlay: Practical mobile augmented reality. In *MobiSys*. ACM, 2015.

[23] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich. The vslam algorithm for robust localization and mapping. In *ICRA*. IEEE, 2005.

[24] R. Kemp, N. Palmer, T. Kielmann, and H. Bal. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*. Springer, 2012.

[25] A. Kirsch and M. Mitzenmacher. Distance-sensitive bloom filters. In *ALENEX*, volume 6. SIAM, 2006.

[26] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, (4):51–56, 2010.

[27] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong. Mobile data offloading: how much can wifi deliver? In *CoNext*, page 26. ACM, 2010.

[28] L. Li, P. Hu, C. Peng, G. Shen, and F. Zhao. Epsilon: A visible light based positioning system. In *NSDI*. USENIX Association, 2014.

[29] R. LiKamWa, Y. Liu, N. D. Lane, and L. Zhong. Moodscope: Building a mood sensor from smartphone usage patterns. In *MobiSys*. ACM, 2013.

[30] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl. Energy characterization and optimization of image sensing toward continuous mobile vision. In *MobiSys*. ACM, 2013.

[31] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang. Large-scale image classification: fast feature extraction and svm training. In *CVPR*. IEEE, 2011.

[32] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, volume 2. Ieee, 1999.

[33] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, 1981.

[34] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*. VLDB Endowment, 2007.

[35] G. Papagiannakis, G. Singh, and N. Magnenat-Thalmann. A survey of mobile and wireless technologies for augmented reality systems. *Computer Animation and Virtual Worlds*, 2008.

[36] B. Rister, G. Wang, M. Wu, and J. R. Cavallaro. A fast and efficient sift detector using the mobile gpu. In *ICASSP*. IEEE, 2013.

[37] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. In *ICCV*. IEEE, 2011.

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

[39] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[40] S. Sen, J. Lee, K.-H. Kim, and P. Congdon. Avoiding multipath to revive inbuilding wifi localization. In *MobiSys*. ACM, 2013.

[41] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*. IEEE, 2008.

[42] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *ICMR*. ACM, 2008.

[43] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment a modern synthesis. In *Vision algorithms: theory and practice*. Springer, 2000.

[44] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury. No need to war-drive: unsupervised indoor localization. In *MobiSys*. ACM, 2012.

[45] Z. Zhang, R. Deriche, O. Faugeras, and Q.-T. Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial intelligence*, 78(1):87–119, 1995.