ALPS: Accurate Landmark Positioning at City Scales

Yitao Hu⋆, Xiaochen Liu⋆, Suman Nath†, Ramesh Govindan⋆
†Microsoft Research **⋆University of Southern California**

Abstract
Context awareness is crucial for ubiquitous computing, and position is an important aspect of context. In an ideal world, every stationary object or entity in the built environment would be associated with position, so that applications can have precise spatial context about the environment surrounding a human. In this paper, we take a step towards this ideal: by analyzing images from Google Street View that cover different perspectives of a given object and triangulating the location of the object, our system, ALPS, can discover and localize common landmarks at the scale of a city accurately and with high coverage. ALPS contains several novel techniques that help improve the accuracy, coverage, and scalability of localization. Evaluations of ALPS on many cities in the United States show that it can localize storefronts with a coverage higher than 90% and a median error of 5 meters.

Author Keywords
Context-aware computing; Landmark localization system; Machine/Deep learning.

ACM Classification Keywords
I.5.4 Pattern Recognition: Applications; I.4.9 Image Processing and Computer Vision: Applications.

INTRODUCTION

Context awareness is essential for ubiquitous computing, and prior work [62, 65] has studied automated methods to detect objects in the environment or determine their precise position. One type of object that has received relatively limited attention is the *common landmark*, an easily recognizable outdoor object which can provide contextual cues. Examples of common landmarks include retail storefronts, signposts (stop signs, speed limits), and other structures (hydrants, street lights, light poles). These can help improve targeted advertising, vehicular safety, and the efficiency of city governments.

In this paper, we explore the following problem: How can we *automatically* collect an accurate database of the precise positions of common landmarks, at the scale of a large city or metropolitan area? The context aware applications described above require an accurate database that also has high coverage: imprecise locations, or spotty coverage, can diminish the utility of such applications.

In this paper, we discuss the design of a system called ALPS (Accurate Landmark Positioning at city Scales), which, given a set of landmark types of interest (*e.g.*, Subway restaurant, stop sign, hydrant), and a geographic region, can enumerate and find the precise position of all instances of each landmark type within the geographic region. ALPS uses a novel combination of two key ideas. First, it uses *image analysis* to find the position of a landmark, given a small number of images of the landmark from different perspectives. Second, it leverages recent efforts, like Google Street View [26], that augment maps with visual documentation of street-side views, to obtain images of such landmarks. At a high-level, ALPS scours Google Street View for images, applies a state-of-the-art off-the-shelf object detector [49] to detect landmarks in images, then triangulates the position of the landmarks using a standard least-squares formulation. On top of this approach, ALPS adds novel techniques that help the system *scale* and improve its *accuracy and coverage*.

Contributions. Our first contribution is techniques for scaling landmark positioning to large cities. Even a moderately sized city can have several million Street View images. If ALPS were to retrieving all images, it would incur two costs, both of which are scaling bottlenecks in ALPS: (1) the latency, network and server load cost of retrieving the images, and (2) the computational latency of applying object detection to the entire collection. ALPS optimizes these costs, without sacrificing coverage, using two key ideas. First, we observe that Street View has a finite resolution of a few meters, so it suffices to sample the geographic region at this resolution. Second, at each sampling point, we retrieve a small set of images, *lazily* retrieving additional images for positioning only when a landmark has been detected in the retrieved set. In addition, the ALPS system can take location *hints* to improve scalability: these hints specify where landmarks are likely to be found (*e.g.*, at street corners), which helps narrow down the search space.

Our second contribution is techniques that improve accuracy and coverage. Object detectors can have false positives and false negatives[1]. ALPS can reduce false negatives by using multiple perspectives: if a landmark is not detected at a sampling point either because it is occluded or because of poor lighting conditions, ALPS tries

[1]For an object detector, a false positive means that the detector detected a landmark in an image that doesn't actually have the landmark. A false negative is when the detector didn't detect the landmark in the image that actually does contain the landmark.

to detect it in images retrieved at neighboring sampling points. To avoid false positives, when ALPS detects a landmark in an image, it retrieves zoomed in versions of that image and runs the object detector on them, using majority voting to increase detection confidence. Once ALPS has detected landmarks in images, it must resolve aliases (multiple images containing the same landmark). Resolving aliases is especially difficult for densely deployed landmarks like fire hydrants, since images from geographically nearby sampling points might contain different instances of hydrants. ALPS clusters images by position, then uses the relative bearing to the landmark to refine these clusters. Finally, ALPS uses least squares regression to estimate the position of the landmark; this enables it to be robust to position and orientation errors, as well as errors in the position of the landmark within the image as estimated by the object detector.

Our final contribution is an exploration of ALPS' performance at the scale of a zip-code, and across several major cities. ALPS can cover over 92% of Subway restaurants in several large cities and over 95% of hydrants in one zip-code, and localize 93% of Subways and 87% of hydrants with an error less than 10 meters. Its localization accuracy is better than Google Places [25] for over 85% of the Subways in large cities. ALPS's scaling optimizations can reduce the number of retrieved images by over a factor of 20, while sacrificing coverage only by 1-2%. Its accuracy improvements are significant: for example, removing the bearing-based refinement (discussed above) can reduce coverage by half.

MOTIVATION AND CHALLENGES

Positioning Common Landmarks. Context awareness [48, 59] is essential for ubiquitous computing since it can enable computing devices to reason about the built and natural environment surrounding a human, and provide appropriate services and capabilities. Much research has focused on automatically identifying various aspects of context [62, 65, 34], such as places and locations where a human is or has been, the objects or people within the vicinity of the human and so forth.

One form of context that can be useful for several kinds of outdoor ubiquitous computing applications is the landmark, an easily recognizable feature or object in the built environment. In colloquial usage, a landmark refers to a famous building or structure which is easily identifiable and can be used to give directions. In this paper, we focus on common landmarks, which are objects that frequently occur in the environment, yet can provide contextual cues for ubiquitous computing applications. Examples of common landmarks include storefronts (*e.g.*, fast food stores, convenience stores), signposts such as speed limits and stop signs, traffic lights, fire hydrants, and so forth.

Potential Applications. Knowing the type of a common landmark (henceforth, landmark) and its precise position (GPS coordinates), and augmenting maps with this information, can enable several applications.

Autonomous cars [54] and drones [13] both rely on visual imagery. Using cameras, they can detect command landmarks in their vicinity, and use the positions of those landmarks to improve estimates of their own position. Drones can also use the positions of common landmarks, like storefronts, for precise delivery.

Signposts can provide context for vehicular control or driver alerts. For example, using a vehicle's position and a database of the position of speed limit signs [43], a car's control system can either automatically regulate vehicle speed to within the speed limit, or warn drivers when they exceed the speed limit. Similarly, a vehicular control systems can use a stop sign position database to slow down a vehicle approaching a stop sign, or to warn drivers in danger of missing the stop sign.

A database of automatically generated landmark positions can be an important component of a smart city [6]. Firefighters can respond faster to fires using a database of positions of fire hydrants [19]. Cities can maintain inventories of their assets (street lights, hydrants, trees [61], and signs [3] are example of city assets) [30, 18]; today, these inventories are generated and maintained manually. Finally, drivers can use a database of parking meter positions, or parking sign positions to augment the search for parking spaces [42], especially in places where in-situ parking place occupancy sensors have not been installed [17].

Landmark locations can also improve context-aware customer behavior analysis [36]. Landmark locations can augment place determination techniques [15, 4]. Indeed, a database of locations of retail storefronts can directly associate place names with locations. Furthermore, landmark locations, together with camera images taken by a user, can be used to more accurately localize the user itself than is possible with existing location services. This can be used in several ways. For example, merchants can use more precise position tracks of users to understand the customer shopping behavior. They can also use this positioning to target users more accurately with advertisements or coupons, enriching the shopping experience.

Finally, landmark location databases can help provide navigation and context for visually impaired persons [10, 28]. This pre-computed database can be used by smart devices (*e.g.* Google Glass) to narrate descriptions of surroundings (*e.g.*, "You are facing a post office and your destination is on its right, and there is a barbershop on its left.") to visually impaired users.

Challenges and Alternative Approaches. An *accurate* database which has high coverage of common landmark locations can enable these applications. High coverage is important because, for example, a missing stop sign can result in a missed warning. Moreover, if the database is complete for one part of a city, but non-existent for another, then it is not useful because applications cannot rely on this information being available.

To our knowledge, no such comprehensive public database exists today, and existing techniques for compiling the database can be inaccurate or have low coverage. Online maps (*e.g.*, Google Places [25] or Bing Maps [8]) contain approximate locations of some retail storefronts (discussed below). Each city, individually, is likely to have reasonably accurate databases of stores within the city, or city assets. In some cases, this information is public. For example, the city of Los Angeles has a list of fire hydrant locations [40], but not many other cities make such information available. Collecting this information from cities can be logistically difficult. For some common landmarks, like franchise storefronts, their franchiser makes available a list of franchisee addresses: for example, the list of Subway restaurants in a city can be obtained from `subway.com`. From this list, we can potentially derive locations through reverse geo-coding, but this approach doesn't generalize to the other landmarks discussed above. Prior work has explored two other approaches to collecting this database: crowdsourcing [46, 66], and image analysis [21]. The former approach relies on users to either explicitly (by uploading stop signs to OpenStreetMaps) or implicitly (by checking in on a social network) tag landmarks, but can be inaccurate due to user error, or have low coverage because not all common landmarks may be visited. Image analysis, using geotagged images from photo sharing sites, can also result in an incomplete database.

In this paper, we ask the following question: *is it possible to design a system to automatically compile, at the scale of a large metropolis, an accurate and high coverage database of landmark positions*? Such a system should, in addition to being accurate and having high coverage, be extensible to different types of landmarks, and scalable in its use of computing resources. In the rest of the paper, we describe the design of a system called ALPS that satisfies these properties.

THE DESIGN OF ALPS

Approach and Overview

The input to ALPS is a landmark *type* (a chain restaurant, a stop sign, *etc.*) and a geographical region expressed either using a zip code or a city name.[2] The output of ALPS is a list of GPS coordinates (or *positions*) at which the specified type of landmark may be found in the specified region. Users of ALPS can specify other optional inputs, discussed later.

ALPS localizes landmarks by *analyzing images* using the following idea. To localize a fire hydrant, for example, suppose we are given three images of the same fire hydrant, taken from three different perspectives, and the position and orientation of the camera when each image was taken is also known. Then, if we can detect the hydrant in each image using an *object detector*, then we can establish the bearing of the hydrant relative to each image. From the three bearings, we can *triangulate* the location of the hydrant. ALPS uses more complex variants of this idea to achieve *accuracy*, as discussed below.

To obtain such images, ALPS piggybacks on map-based visual documentation of city streets [27, 9]. Specifically, ALPS uses the imagery captured by Google's Street View. The vehicles that capture Street View images have positioning and bearing sensors [53], and the Street View API permits a user to request an image taken at a given position and with a specified bearing. ALPS's *coverage* is dictated in part by Street View's coverage, and its *completeness* is a combination of its coverage, and the efficacy of its detection and localization algorithms.

Street View (and similar efforts) have large databases, and downloading and processing all images in a specified geographic region can take time, computing power, and network bandwidth. To *scale* to large geographic regions (*e.g.*, an entire zipcode or larger), ALPS employs novel techniques that (a) retrieve just sufficient images to ensure high coverage, (b) robustly detect the likely presence of the specified landmark, then (c) drill down and retrieve additional images in the vicinity to localize the landmarks.

Finally, users can easily *extend* ALPS to new landmark types, and specify additional scaling hints.

ALPS comprises two high-level capabilities (Figure 1): *Seed location generation* takes a landmark type specified by user as input, and generates a list of *seed locations* where the landmarks *might* be located; and *Landmark localization* takes seed locations as input and generates landmark positions in the specified geographic region as output.

In turn, seed location generation requires three conceptual capabilities: (1) *base image retrieval* which downloads a subset of all Street View images; (2) *landmark detection* that uses the state-of-the-art computer vision object detection [49] to detect and localize[3] landmarks retrieved by base image retrieval, and applies additional filters to improve the accuracy of detection; (3) *image clustering* groups detected images that likely contain the same instance of the landmark. The result of these three steps is a small set of seed locations where the landmark is likely to be positioned, derived with minimal resources without compromising coverage.

Landmark localization reuses the landmark detection capability, but requires two additional capabilities: (1) *adaptive image retrieval*, which drills down at each seed location to retrieve as many images as necessary for localizing the object; (2) and a *landmark positioning* capability that uses least squares regression to triangulate the landmark position.

[2]For now, we have only experimented with cities and regions in North America, but ALPS should be extensible to other parts of the globe as well.

[3]In the context of object detection, to localize means to find the position of the object in the image. ALPS uses this capability to localize the object in the global (GPS) coordinate frame.

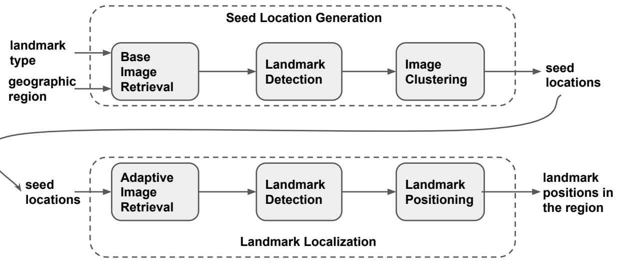


Figure 1: ALPS Components

Base Image Retrieval

ALPS retrieves images from Street View, but does not retrieve *all* Street View images within the input geographic region. This brute-force retrieval does not scale, since even a small city like Mountain View can have more than 10 million images. Moreover, this approach is wasteful, since Street View's resolution is finite: in Figure 2(a), a Street View query for an image anywhere within the dotted circle will return the image taken from one of the points within that circle.

ALPS scales better by retrieving as small a set of images as possible, without compromising coverage (Figure 2(b)). It only retrieves two Street View images in two opposing directions perpendicular to the street, at intervals of $2r$ meters, where $2r$ is Street View's resolution (from experiments, r is around 4 meters). By using nominal lane [55] and sidewalk [56] widths, Street View's default angle of view of 60°, it is easy to show using geometric calculations that successive 8 meter samples of Street View images have overlapping views, thereby ensuring visual coverage of the entire geographic region.

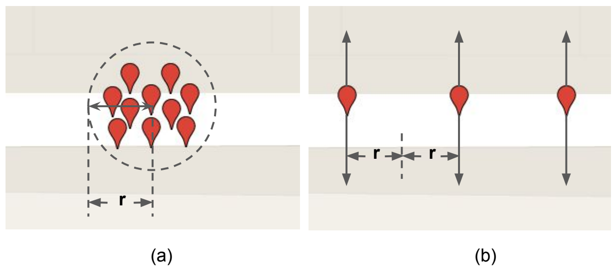


Figure 2: Base Image Retrieval

Landmark Detection

Given an image, this capability detects and localizes the landmark within the image. This is useful both for seed location generation, as well as for landmark localization, discussed earlier. Recent advances [35, 23] in deep learning techniques have enabled fast and accurate object detection and localization. We use a state-of-the-art object detector, called YOLO [49]. YOLO uses a neural network to determine whether an object is present in an image, and also draws a *bounding box* around the part of the image where it believes the object to be (*i.e.*, *localizes* the object in the image). YOLO needs to be trained, with a large number of training samples, to detect objects. We have trained YOLO to detect logos of several chain restaurants or national banks, as well as stop signs and fire hydrants. Users wishing to extend ALPS functionality to other landmark types can simply provide a neural network trained for that landmark.

Even the best object detection algorithms can have false positives and negatives [32]. False positives occur when the detector mistakes other objects for the target landmark due to lighting conditions, or other reasons. False negatives can decrease the coverage and false positives can reduce positioning accuracy. In our experience, false negatives arise because YOLO cannot detect objects smaller than 50×50 pixels or objects that are blurred, partially obscured or in shadow, or visually indistinguishable from the background.

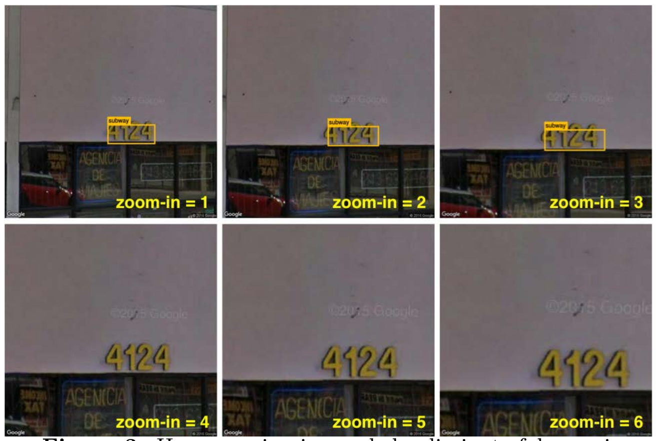
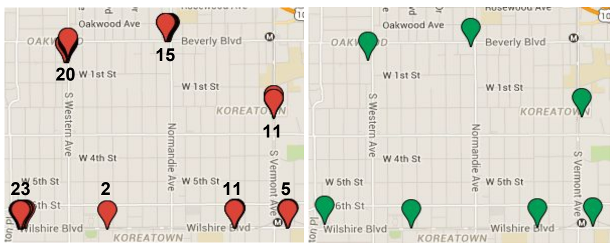


Figure 3: How zooming-in can help eliminate false positives

ALPS reduces false positives by using Street View's support for retrieving images at different *zoom levels*. Recall that base image retrieval downloads two images at each sampling point. ALPS applies the landmark detector to each image: if the landmark is detected, ALPS retrieves six different versions of the corresponding Street View image each at different zoom levels. It determines the tilt and bearing for each of these zoomed images based on the detected landmark. ALPS then uses two criteria to mark the detection as a true positive: that YOLO should detect a landmark in a majority of the zoom levels, and that the size of the bounding box generated by YOLO is proportional to the zoom level. For example, in Figure 3, YOLO incorrectly detected a residence number, when detecting a Subway logo, in the first three zoom levels (the first zoom level corresponds to the base image). After zooming in further, YOLO was unable to detect the Subway logo in the last 3 zoomed-in images. In this case, ALPS declares that the image does not contain a Subway logo, because the majority vote failed. We address false negatives in later steps.

Image Clustering

To generate seed locations, ALPS performs landmark detection on each image obtained by base image retrieval. However, two different images might contain the same landmark: the *clustering* step uses image position and orientation to cluster such images together. In some cases, this clustering can reduce the number of seed locations dramatically: in Figure 4(a), 87 landmarks are detected in the geographic region shown, but a much smaller fraction of them represent unique landmarks (Figure 4(b)).



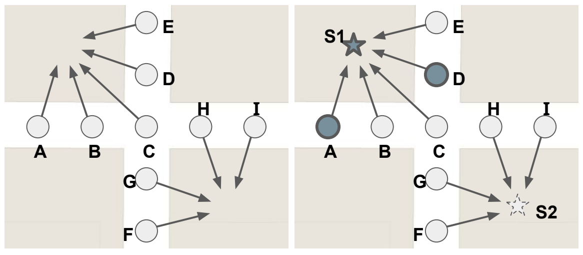
(a) (b)

Figure 4: Clustering can help determine which images contain the same landmark

The input to clustering is the set of images from the base set in which a landmark has been detected. ALPS clusters this set by using the position and bearing associated with the image, in two steps: first, it clusters by position, then, within each cluster it distinguishes pairs of images whose bearing is inconsistent.

To cluster by position, we use mean shift clustering [22]: (1) put all images into a candidate pool; (2) select a random image in the candidate pool as the center of a new cluster; (3) find all images within R meters (R=50 in our implementation) of the cluster center, put these images into the cluster, and remove them from the candidate pool; (4) calculate the mean shift of the center of all nodes within the cluster, and if the center is not stable, go to step (3), otherwise go to step (2).

Clustering by position works well for landmarks likely to be geographically separated (*e.g.*, a Subway restaurant), but not for landmarks (*e.g.*, a fire hydrant) that can be physically close. In the latter case, clustering by position can reduce accuracy and coverage.

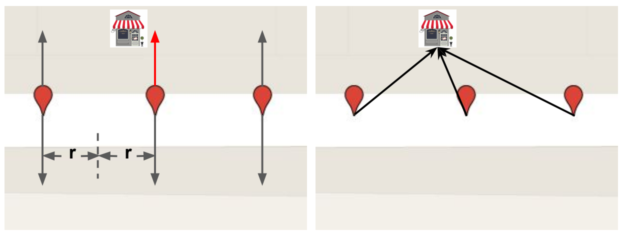


(a) (b)

Figure 5: Clustering by bearing is necessary to distinguish between two nearby landmarks

To improve the accuracy of clustering, we use bearing information in the Street View images to refine clusters generated by position-based clustering. Our algorithm is inspired by the RANSAC [20] algorithm for outlier detection, and is best explained using an example. Figure 5(a) shows an example where ALPS's position-based clustering returns a cluster with 9 images A-I. In Figure 5(b), images A-E and images F-I see different landmarks. ALPS randomly picks two images A and D, adds them to a new *proto-cluster*, and uses its positioning algorithm (described below) to find the *approximate* position of the landmark ($S1$) as determined from these images. It then determines which other images have a bearing consistent with the estimated position of $S1$. H's bearing is inconsistent with $S1$, so it doesn't belong to the new proto-cluster, but B's bearing is. ALPS computes all possible proto-clusters in the original cluster, then picks the lowest-error large proto-cluster, outputs this as a refinement of the original cluster, removes these images from the original cluster, and repeats the process. In this way, images A-E are first output as one cluster, and images F-I as another.

Each cluster contains images that, modulo errors in position, bearing and location, contain the same landmark. ALPS next uses its positioning algorithm (discussed below) to generate a *seed location* for the landmark.



(a) (b)

Figure 6: Adaptive Image Retrieval

Adaptive Image Retrieval

A seed location may not be precise, because the images used to compute it are taken perpendicular to the street (Figure 6(a)). If the landmark is offset from the center of the image, errors in bearing and location can increase the error of the positioning algorithm. Moreover, the landmark detector may not be able to accurately draw the bounding box around a landmark that is a little off-center. Location accuracy can be improved by retrieving images whose bearing matches the heading from the sampling point to the seed location (so, the landmark is likely to be closer to the center of the image, Figure 6(b)). (A seed location may not be precise also because a cluster may have too few images to triangulate the landmark position. We discuss below how we address this.)

To this end, we use an idea we call *adaptive image retrieval*: for each image in the cluster, we retrieve one additional image with the same position, but with a bearing directed towards the seed location (Figure 6(b)). At this stage, we also deal with false negatives. If a cluster has fewer than k images ($k = 4$ in our implementation), we retrieve one image each from neighboring sampling points with a heading towards the seed location, *even if these points are not in the cluster.* In these cases, the landmark detector may have missed the landmark because it was in a corner of the image; retrieving an image with a bearing towards the seed location may enable the detector to detect the landmark, and enable higher positioning accuracy because we have more perspectives.

Landmark Positioning

Precisely positioning a landmark from the images obtained using adaptive image retrieval is a central capability in ALPS. Prior work in robotics reconstructs the 3-D position of an object from multiple locations using three key steps [51, 14]: (1) camera calibration with intrinsic and extrinsic parameters (camera 3-D coordinates, bearing direction, tilt angle, field of view, focus length, *etc.*); (2) feature matching with features extracted by algorithm like SIFT [41]; (3) triangulation from multiple images using a method like singular value decomposition [24].

In our setting, these approaches don't work too well: (1) Street View does not expose all the extrinsic and intrinsic camera parameters; (2) some of available parameters (GPS as 3-D coordinates, camera bearing) are noisy and erroneous, which may confound feature matching; (3) Street View images of a landmark are taken from different directions and may have differing light intensity, which can reduce feature matching accuracy; (4) panoramic views in Street View can potentially increase accuracy, but there can be distortion at places in the panoramic views where images have been stitched together [53].

Instead, ALPS (1) projects the landmark (*e.g.*, a logo) onto a 2-dimensional plane to compute the relative bearing of the landmark and the camera, then (2) uses least squares regression to estimate the landmark position.

Estimating Relative Bearing. ALPS projects the viewing directions onto a 2-D horizontal plane as shown in Figure 7(a). O represents the landmark in 3-dimensions, and $O^{'}$ represents the projected landmark on a 2-D horizontal plane. C_i and its corresponding $C_i^{'}$ represent the camera locations in 3-D and 2-D respectively. Thus, $\vec{C_iO}$ is the relative bearing from camera i to landmark O, and $\vec{C_i^{'}O}$ is its projection.

The landmark detector draws a bounding box around the pixels representing the landmark, and for positioning, we need to be able to estimate the relative bearing of the center of this bounding box relative to the bearing of the camera itself. In Figure 7(b), line $\vec{AB}$ demarcates the (unknown) depth of the image and vector $\vec{C^{'}H}$ represents the bearing direction of camera, so $O^{''}$ is the image of $O^{'}$ on $\vec{AB}$. Our goal is to estimate $\angle O^{'}C^{'}X$ or $\angle 4$, which is the bearing of the landmark relative to x-axis.

To do this, we need to estimate the following three variables: (1) the camera angle of view $\angle AC^{'}B$ or $\angle 1$, which is the maximum viewing angle of the camera; (2) the camera bearing direction $\angle HC^{'}X$ or $\angle 2$, which is the bearing direction of the camera when the image was taken; (3) the relative bearing direction of the landmark $\angle O^{''}C^{'}D$ or $\angle 3$, which is the angle between the bearing direction of the camera and the bearing direction of the landmark.

$\angle 1$ and $\angle 2$ can be directly obtained from image metadata returned by Street View. Figure 7(b) illustrates how to calculate $\angle 3 = \arctan(|\vec{DO''}|/|\vec{DC'}|)$. Landmark detection returns the image width in pixels and the pixel coordinates of the landmark bounding box. Thus, $|\vec{DO''}| = |\vec{AO''}| - \frac{1}{2}|\vec{AB}|$. Since $\tan(\frac{1}{2}\angle 1) = |\vec{AD}|/|\vec{DC'}|$, we can calculate $|\vec{DC'}| = \frac{1}{2}|\vec{AB}|\tan(\frac{1}{2}\angle 1)$. Then we derive $\angle 3$ as $\arctan(|\vec{DO''}|/|\vec{DC'}|)$. Finally, we can calculate the bearing direction of the landmark: $\angle 4 = \angle 2 - \angle 3$.

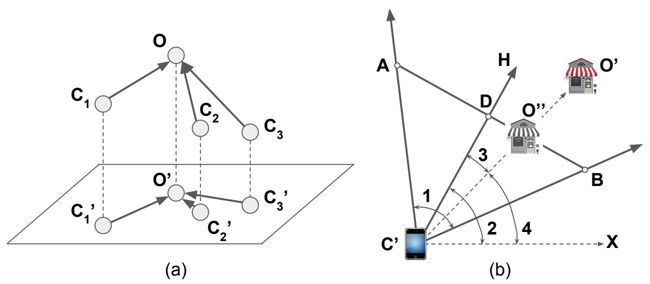


Figure 7: Landmark Positioning

Positioning using Least Squares Regression. For each cluster, using adaptive image retrieval, ALPS retrieves N images for a landmark and can calculate the relative bearing of the landmark to the camera by executing landmark detection on each image. Positioning the landmark then becomes an instance of the landmark localization problem [58, 33], where we have to find the landmark location $P = [x_o, y_o]$ given N distinct viewing locations $p_i = [x_i, y_i], i = 1, 2, \ldots, N$ with corresponding bearing $\theta_i, i = 1, 2, \ldots, N$, where $[x_i, y_i]$ is point p_i's GPS coordinates in x-y plane. From first principles, we can write θ (or $\angle 4$) as follows:

$$\tan(\theta_i) = \frac{\sin(\theta_i)}{\cos(\theta_i)} = \frac{y_o - y_i}{x_o - x_i}. \tag{1}$$

Simplifying this equation and combining the equations for all images, we can write the following system of linear equations:

$$G\beta = h, \tag{2}$$

where $\beta = [x_o, y_o]^T$ represents the landmark location, $G = [g_1, g_2, \ldots, g_N]^T$, $g_i = [\sin(\theta_i), -\cos(\theta_i)]$, $h = [h_1, h_2, \ldots, h_N]^T$, $h_i = [\sin(\theta_i)x_i - \cos(\theta_i)y_i]$.

In this system of linear equations, there are two unknowns x_o and y_o, but as many equations as images, resulting in an overdetermined system. However, many of the θ_is may be inaccurate because of *errors in camera bearing, location, or landmark detection.* To determine the most likely position, ALPS approximates $\hat{\beta}$ using least squares regression, which minimizes the squared residuals $S(\beta) =$

$||G\beta - h||^2$, as output. If G is full rank, the least squares solution of Equation 2 is:

$$\hat{\beta} = \arg\min(S(\beta)) = (G^T G)^{-1} G^T h. \tag{3}$$

Putting it All Together

Given a landmark type and a geographic region, ALPS first retrieves a base set of images for the complete region, which ensures coverage. On each image in this set, it applies landmark detection, retrieving zoomed-in versions of the image if necessary to obtain higher confidence in the detection and reduce false positives. It applies position and bearing based clustering on the images where a landmark was detected. Each resulting cluster defines a *seed location*, where the landmark might be.

At each seed location, ALPS adaptively retrieves additional images, runs landmark detection on each image again to find the bearing of the landmark relative to the camera for each image, and uses these bearings to formulate a system of linear equations whose least squares approximation represents the position of the landmark.

Flexibility

ALPS is flexible enough to support extensions that add to its functionality, or improve its scalability.

New landmark types. Users can add to ALPS's library of landmark types by simply training a neural network to detect that type of landmark. No other component of the system needs to be modified.

Seed location hints. To scale ALPS better, users can specify seed location hints in two forms. ALPS can take *a list of addresses* and generate seed locations from this using reverse geo-coding. ALPS also takes *spatial constraints* that restrict base image retrieval to sampling points satisfying these constraints. For example, fire hydrants usually can be seen at or near street corners, or on a street midway between two cross-streets. Therefore, to specify such constraints, ALPS provides users with a simple language with 4 primitives: `at_corner` (only at street corners), `midway` (at the midpoint between two cross-streets, `searching_radius` (search within a radius of the points specified by other constraints), and `lower_image` (the landmark like a fire hydrant only appears in the lower part of the image). More spatial constraints may be required for other landmarks: we have left this to future work.

ALPS EVALUATION

In this section, we evaluate the coverage, accuracy, scalability and flexibility of ALPS on two different types of landmarks: Subway restaurants, and fire hydrants.

Methodology

Implementation and Experiments. We implemented ALPS in C++ and Python and accessed Street View images using Google's API [27]. Our implementation is 2708 lines of code.[4] All experiments described in the paper are run on a single server with an Intel Xeon CPU at 2.70GHz, 32GB RAM, and one Nvidia GTX Titan X GPU inside. Below, we discuss the feasibility of parallelizing ALPS's computations across multiple servers.

Dataset. We evaluate ALPS using images for several geographic regions across five cities of the United States. In some of our experiments, we use seed location hints to understand the coverage and accuracy at larger scales.

Ground Truth. For both landmark types we evaluate, getting ground truth locations[5] is not easy because no accurate position databases exist for these. So, we *manually* collected ground truth locations for these as follows. For Subway restaurants, we obtained street addresses for each restaurant within the geographic region from the chain's website [50]. For fire hydrants, there exists an ArcGIS visualization of the fire hydrants in 2 zipcodes [40] (as an aside, such visualizations are not broadly available for other geographic locations, and even these do not reveal exact position of the landmark). From these, we obtained the approximate location for each instance of the landmark. Using this approximate location, we first manually viewed the landmark on Street View, added a pinpoint on Google Maps at the location where we observed the landmark to be, then extracted the GPS coordinate of that pinpoint. This GPS coordinate represents the ground truth location for that instance.

We validated of this method by collecting measurements at 30 of these landmarks using a high accuracy GPS receiver [52]. The 90th percentile error between our manual labeling and the GPS receiver is 6 meters.[6]

Metrics. To measure the performance of ALPS, we use three metrics. *Coverage* is measured as the fraction of landmarks discovered by ALPS from the ground truth (this measures *recall* of our algorithms). Where appropriate, we also discuss the false positive rate of ALPS, which can be used to determine ALPS's *precision.* The *accuracy* of ALPS is measured by its *positioning error*, the distance between ALPS's position and ground truth. For *scalability*, we quantify the processing speed of each module in ALPS and the number of retrieved images. (We use the latter as a proxy for the time to retrieve images, which can depend on various factors like the Street View image download quota (25,000 images per day per user [27]) and access bandwidth that can vary significantly).

Coverage and Accuracy

To understand ALPS's coverage and accuracy, we applied ALPS to the zip-code 90004 whose area is 4 sq. km., to localize both Subway restaurants and fire hydrants. To understand ALPS's performance at larger scales, we

[4]Available at https://github.com/USC-NSL/ALPS

[5]We define the precise physical GPS address as ground truth location

[6]In the three cases where the error was high, we noticed that a sunshade obstructed our view of the sky, so the GPS receiver is likely to have obtained an incorrect position fix.

used seed location hints to run ALPS at the scale of large cities in the US.

Zip-code 90004. Table 1 shows the coverage of the two landmark types across the entire zip-code. There are seven Subways in this region and ALPS discovers all of them, with no false positives. Table 3 shows that ALPS localizes all Subways within 6 meters, with a median error of 4.7 meters. By contrast, the error of the GPS coordinates obtained from Google Places is over 10 meters for each Subway and nearly 60 meters in one case. Thus, at the scale of a single zip-code, ALPS can have high coverage and accuracy for this type of landmark.

Fire hydrants are much harder to cover because they are smaller in shape, lower in position so can be occluded, can blend into the background or be conflated with other objects. As Table 1 shows, ALPS finds 262 out of 330 fire hydrants for an 79.4% coverage. Of the ones that ALPS missed, about 16 were *not* visible to the naked eye in any Street View image, so no image analysis technique could have detected these. In 12 of these 16, the hydrant was occluded by a parked vehicle (which is illegal, Figure 8 b)) in the Street View image, and the remaining 4 simply did not exist in the locations indicated in [40]. Excluding these, ALPS's coverage increases to about 83.4%. ALPS can also position these hydrants accurately. Figure 9 shows the cumulative distribution function (CDF) of errors of ALPS for fire hydrants in 90004. It can localize 87% of the hydrants within 10 meters, and its median error is 4.96 meters.

We then manually inspected the remaining 52 fire hydrants visible to the human eye but not discovered by ALPS. In 6 of these cases, the fire hydrant was occluded by a car in the image downloaded by base image retrieval: a brute-force image retrieval technique would have discovered these (see below). The remaining 46 missed hydrants fell roughly evenly into two categories. First, 24 of them were missed because of shortcomings of the object detector we use. In these cases, even though the base image retrieval downloaded images with hydrants in them, the detector did not recognize the hydrant in any of the images either because of lighting conditions (*e.g.*, hydrant under the shade of a tree, Figure 8 a)), or the hydrant was blurred in the image. The remaining 22 false negatives occurred because of failures in the positioning algorithm. This requires multiple perspectives (multiple images) to triangulate the hydrant, but in these cases, ALPS couldn't obtain enough perspectives either because of detector failures or occlusions. Finally, the 21 false positives were caused by the object detector misidentifying other objects (such as a bollard, Figure 8 c)) as hydrants. Future improvements to object detection, or better training and parametrization of the object detector, can reduce both false positives and false negatives. We have left this to future work.

Finally, both false positives and negatives in ALPS can be eliminated by using competing services like Bing Streetside [9] which may capture images when a landmark is not occluded, or under different lighting conditions, or from perspectives that eliminate false positive detections. To evaluate this idea, we downloaded images from Bing Streetside near fire hydrants that were not detected using Google Street View. By combining both image sources, ALPS detected 300 out of 314 visible fire hydrants, resulting in 95.5% coverage in this area.

Type	# landmark	# visible	# ALPS	Coverage
Subway	7	7	7	100%
Hydrant	330	314	262	83.4%

Table 1: Coverage of ALPS

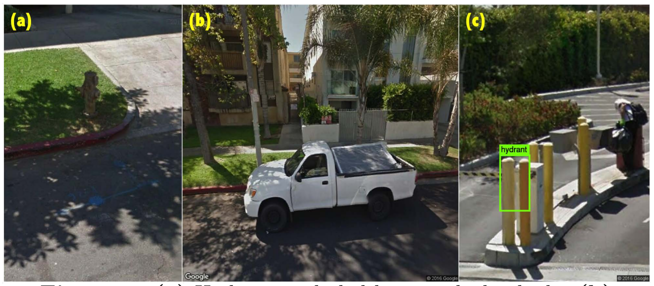


Figure 8: **(a)** Hydrant occluded by a parked vehicle. **(b)** Detection failure because hydrant is under the shade of a tree. **(c)** False positive detection of bollard as hydrant.

City-Scale Positioning. To understand the efficacy of localizing logos, like that of Subway, over larger scales, we evaluated ALPS on larger geographic areas on the scale of an entire city. At these scales, ALPS will work, but we did not wish to abuse the Street View service and download large image sets. So, we explored city-scale positioning performance by feeding seed-location hints in the form of addresses for Subway restaurants, obtained from the chain's web page.

Table 2 shows the coverage with seed locations in different areas. Across these five cities, ALPS achieves more than 92% coverage. With seed location hints, ALPS does not perform base image retrieval, so errors arise for other reasons. We manually analyzed the causes for errors in these five cities. In all cities, the invisible Subways were inside a building or plaza, so image analysis could not have located them. The missed Subways in Los Angeles, Mountain View, San Diego and Redmond were either because: (a) the logo detector failed to detect the logo in any images (because the image was partly or completely occluded), or (b) the positioning algorithm did not, in some clusters, have enough perspectives to localize.

ALPS does not exhibit false positives for Subway restaurants. For hydrants, all false positives arise because the landmark detector mis-detected other objects as hydrants. The Subway sign is distinctive enough that, even though the landmark detector did have some false positives, these were weeded out by the rest of the ALPS pipeline.

At city-scales also, the accuracy of ALPS is high. Figure 10 shows the CDF of errors of ALPS and Google Places locations for all of the Subways (we exclude the Subways that are not visible in any Street View image). ALPS can localize 93% of the Subways within 10 meters, and its median error is 4.95 meters while the median error from Google places is 10.17 meters. Moreover, for 87% of the Subways, Google Places has a higher error than ALPS in positioning. These differences might be important for high-precision applications like drone-based delivery.

Scalability: Bottlenecks and Optimizations

Processing Time. To understand scaling bottlenecks in ALPS, Table 4 breaks down the time taken by each component for the 90004 zip-code experiment (for both Subways and hydrants). In this experiment, base image retrieval, which retrieved nearly 150 thousand images, was performed only once (since that component is agnostic to the type of landmark being detected). Every other component was invoked once for each type of landmark.

Of the various components, clustering and positioning are extremely cheap. ALPS thus has two bottlenecks. The first is image retrieval, which justifies our optimization of this component (we discuss this more below). The second bottleneck is running the landmark detector. On average, it takes *59 milliseconds* for the landmark detector to run detection on an image, regardless of whether the image contains the landmark or not. However, because we process over 150 thousand images, these times become significant. (Landmark detection is performed both on the base images to determine clusters, and on adaptively retrieved images for positioning, hence the two numbers in the table). Faster GPUs can reduce this time.

Fortunately, ALPS can be scaled to larger regions by parallelizing its computations across multiple servers. Many of its components are trivially parallelizable, including base image retrieval which can be parallelized by partitioning the geographic region, and adaptive image retrieval and positioning which can be partitioned across seed location. Only clustering might not be amenable to parallelization, but clustering is very fast. We have left an implementation of this parallelization to future work.

The Benefit of Adaptive Retrieval. Instead of ALPS's two phase (basic and adaptive) retrieval strategy, we could have adopted two other strategies: (a) a *naive* strategy which downloads images at very fine spatial scales of 1 meter, (b) a *one phase* strategy which downloads 6 images, each with a 60° viewing angle so ALPS can have high visual coverage. For the 90004 zip-code experiment, the naive strategy retrieves 24× more images than ALPS's two-phase strategy, while one-phase retrieves about 3× as many. The retrieval times are roughly proportional to the number of images retrieved, so ALPS's optimizations provide significant gains. These gains come at a very small loss in coverage: one-phase has 1.91% higher coverage than two-phase for hydrants[7] mostly because the former has more perspectives: for example, hydrants that were occluded in the base images can be seen in one-phase images.

[7]We have not evaluated the coverage gains for naive, given the large number of retrievals (3 million) required.

Seed Location Hints. We have already seen that seed location hints helped us scale ALPS to large cities. These hints provide similar trade-offs as adaptive retrieval: significantly fewer images to download at the expense of slightly lower coverage. For hydrants in 90004, using hints that tell ALPS to look at street corners or mid-way between intersections and in the lower half of the image enabled ALPS to retrieve 3× fewer images, while only detecting 5% fewer hydrants.

Accuracy and Coverage Optimizations

Object Detection Techniques. The accuracy of the object detector is central to ALPS. We evaluated the recall, precision, and processing time of several different object detection approaches: YOLO, HoG+SVM, and keypoint matching [16] with SIFT [41] features. For HoG+SVM, we trained LIBSVM [38] with HoG [12] features and a linear kernel. Table 5 shows that YOLO outperforms the other two approaches in both recall and precision for recognizing the Subway logo. YOLO also has the fastest processing time due to GPU acceleration.

Street View Zoom. ALPS uses zoomed Street View images to increase detection accuracy. To quantify this, after using zoomed in images the landmark detector had a precision of 96.2% and a recall of 86.8%. In comparison, using only Yolo without zoomed in images had a precision of 85.1% and recall of 87.4%.

To understand how the object detector affects the accuracy of ALPS, we manually labeled the position of the Subway logo in all the images in the dataset of Subways in LA. We thus emulated an object detector with 100% precision and recall. This ideal detector finds the 3 missing Subways (by design), but with position accuracy comparable to YOLO (Figure 11).

Importance of Bearing-based Clustering. We used the fire hydrant dataset to understand the incremental benefit of bearing-based cluster refinement. Without this refinement, ALPS can only localize 141 fire hydrants of 314 visible ones, while the refinement increases coverage by nearly 2× to 262 hydrants. Moreover, without bearing-based refinement, position errors can be large (in one case, as large as 80 meters) because different hydrants can be grouped into one cluster.

RELATED WORK

Prior work in ubiquitous and pervasive computing deals with localizing objects within the environment, or humans. Some of these have explored localizing users using low-cost energy-efficient techniques on mobile devices: magnetic sensors [63], inertial and light sensors [65], inertial sensors together with wireless fingerprints [29, 31], RF signals [45, 7, 47], mobility traces [34] and other activity fingerprints [37]. Other work has explored localizing a network of devices using low-cost RF powered cameras [44]. Many of these techniques are largely complementary

City	# Subway	# Visible	# ALPS	Coverage	Median error(m)
Los Angeles	123	118	115	97%	4.8
Mountain View	38	26	24	92%	5.1
San Francisco	49	39	39	100%	4.2
San Diego	57	44	41	93%	5.0
Redmond	31	25	24	96%	4.8

Table 2: Coverage with Seed Locations

Subway #	1	2	3	4	5	6	7
Error of Google (m)	10.06	11.53	14.10	30.38	59.48	16.60	14.90
Error of ALPS (m)	2.03	4.53	6.78	2.93	7.39	5.94	3.33

Table 3: Error of ALPS and Google for localizing Subways

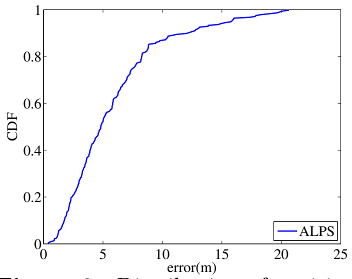


Figure 9: Distribution of position errors for hydrants in 90004 zip-code

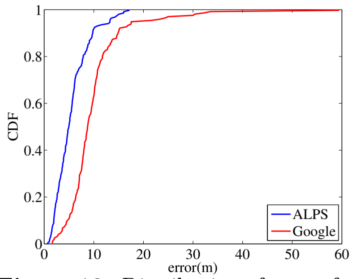


Figure 10: Distribution of errors for Subways in five cities

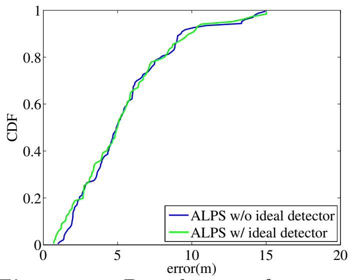


Figure 11: Distribution of position errors for ALPS on Subway w/ and w/o ideal detector in Los Angeles

Module	Base Retrieval	Base Detection	Cluster
Time (s)	3528	8741	0.749
Module	Adaptive Retrieval	Adaptive Detection	Positioning
Time (s)	715	1771	0.095

Table 4: Processing time of each module

	YOLO	HoG+SVM	SIFT
Precision	85.1%	74.2%	63.7%
Recall	87.4%	80.5%	40.6%
Speed (sec/img)	0.059	0.32	0.65

Table 5: Evaluation of different object detection methods

to ALPS, which relies on Street View images to localize common landmarks. Perhaps closest to our work is Argus [64], which complements WiFi fingerprints with visual cues from crowd-sourced photos to improve indoor localization. This work builds a 3-D model of an indoor setting using advanced computer vision techniques, and uses this to derive geometric constraints. By contrast, ALPS derives geometric constraints by detecting common landmarks using object detection techniques. Finally, several pieces of work have explored augmenting maps with place names and semantic meaning associated with places [15, 4]. ALPS derives place name to location mappings for common places with recognizable logos.

Computer vision research has considered variants of the following problem: given a GPS-tagged database of images, and a query image, how to estimate for the GPS position of the given image. This requires matching the image to the image(s) in the database, then deriving position from the geo-tags of the matched images. Work in this area has used Street View images [67, 57], GIS databases [2]), or images from Flickr [11]). The general approach is to match features, such as SIFT in the query image with features in the database of images. ALPS is complementary to this line of work, since it focuses on enumerating common landmarks of a given type. Because these landmarks have distinctive structure, we are able to use object detectors, rather than feature matching.

Research has also considered another problem variant: given a set of images taken by a camera, finding the position of the camera itself. This line of work [1, 39] attempts to match features in the images to a database of geo-tagged images, or to a 3-D model derived from the image database. This is the inverse of the our problem: given a set of geotagged images, to find the position of an object in these images. Finally, Baro *et al.* [5] propose efficient retrieval of images from an image database matching a given object. ALPS goes one step further and actually positions the common landmark.

Recently, Convolutional Neural Network (CNN) based object detection methods have begun to outperform earlier methods (*e.g.*, cascade classifiers [60]) in both speed and coverage. YOLO [49], a CNN-based detector, is able to process images very fast and is crucial for scaling ALPS.

CONCLUSION

ALPS achieves accurate, high coverage, positioning of common landmarks at city-scales. It uses novel techniques for scaling (adaptive image retrieval) and accuracy (increasing confidence using zooming, disambiguating landmarks using clustering, and least-squares regression to deal with sensor error). ALPS discovers over 92% of Subway restaurants in several large cities and over 95% of hydrants in a single zip-code, while localizing 93% of Subways and 87% of hydrants with an error less than 10 meters. Future work includes documenting large cities, and extending ALPS to common landmarks that may be set back from the street yet visible in Street View, such as transmission or radio towers, and integrating Bing Streetside to increase coverage and accuracy.

Acknowledgements. This research was partially sponsored by the US Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053.

Bibliography

[1] Pratik Agarwal, Wolfram Burgard, and Luciano Spinello. "Metric localization using Google Street View". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. 2015, pp. 3111–3118.

[2] Shervin Ardeshir et al. "GIS-assisted Object Detection and Geospatial Localization". In: *Computer Vision–ECCV 2014*. Springer, 2014, pp. 602–617.

[3] Vahid Balali, Elizabeth Depwe, and Mani Golparvar-Fard. "Multi-class traffic sign detection and classification using google street view images". In: *Transportation Research Board 94th Annual Meeting, Transportation Research Board, Washington, DC*. 2015.

[4] Xuan Bao et al. "PinPlace: Associate Semantic Meanings with Indoor Locations Without Active Fingerprinting". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 921–925.

[5] Xavier Baró et al. "Generic Object Recognition in Urban Image Databases". In: *Artificial Intelligence Research and Development, Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence, CCIA 2009, October 21-23, 2009, Vilar Rural de Cardona (El Bages), Cardona, Spain*. 2009, pp. 27–34.

[6] Michael Batty et al. "Smart cities of the future". In: *The European Physical Journal Special Topics* 214.1 (2012), pp. 481–518.

[7] Jacob T. Biehl et al. "LoCo: A Ready-to-deploy Framework for Efficient Room Localization Using Wi-Fi". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington: ACM, 2014, pp. 183–187.

[8] *Bing Maps API*. https://msdn.microsoft.com/en-us/library/dd877180.aspx. 2016.

[9] *Bing Streetside*. https://www.bing.com/mapspreview. 2016.

[10] *Cities Unlocked*. http://www.citiesunlocked.org.uk/. 2016.

[11] David J. Crandall et al. "Mapping the World's Photos". In: *Proceedings of the 18th International Conference on World Wide Web*. WWW '09. Madrid, Spain: ACM, 2009, pp. 761–770.

[12] Navneet Dalal and Bill Triggs. "Histograms of Oriented Gradients for Human Detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*. 2005, pp. 886–893.

[13] Pasquale Daponte et al. "Metrology for drone and drone for metrology: measurement systems on small civilian drones". In: *Proc. of 2nd Int. Workshop on Metrology for Aerospace*. Benevento, Italy, 2015, pp. 316–321. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7180673.

[14] Hugh F. Durrant-Whyte and Tim Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robot. Automat. Mag.* 13.2 (2006), pp. 99–110.

[15] Moustafa Elhamshary and Moustafa Youssef. "CheckInside: A Fine-grained Indoor Location-based Social Network". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington: ACM, 2014, pp. 607–618.

[16] *Feature Points Matching*. http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html.

[17] *Find a Parking Space Online*. https://www.technologyreview.com/s/410505/find-a-parking-space-online/. 2008.

[18] *Fire Mapping: Building and Maintaining Datasets in ArcGIS*. http://www.esri.com/library/ebooks/fire-mapping.pdf. 2012.

[19] *Firefighters Searching for Hydrants*. http://patch.com/connecticut/danbury/firefighters-searching-for-hydrants. 2013.

[20] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (1981), pp. 381–395.

[21] *Flickr API*. https://www.flickr.com/services/api/. 2016.

[22] Keinosuke Fukunaga and Larry D. Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *IEEE Trans. Information Theory* 21.1 (1975), pp. 32–40.

[23] Ross B. Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, pp. 1440–1448.

[24] G. Golub and C. ER Reinsch. "Singular value decomposition and least squares solutions". In: *Numerische Mathematik* 14.5 (1970), pp. 403–420.

[25] *Google Place API*. https://developers.google.com/places/web-service/search. 2016.

[26] *Google Street View*. https://www.google.com/maps/streetview/.

[27] *Google Street View Image API*. https://developers.google.com/maps/documentation/streetview/. 2016.

[28] Kotaro Hara et al. "Exploring early solutions for automatically identifying inaccessible sidewalks in the physical world using google street view". In: *Human Computer Interaction Consortium* (2013).

[29] Suining He et al. "Calibration-free Fusion of Step Counter and Wireless Fingerprints for Indoor Localization". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 897–908.

[30] *High-Tech Web Mapping Helps City of New York's Fire Department Before Emergencies*. http://www.esri.com/news/arcnews/fall10articles/new-york-fire-dept.html. 2010.

[31] Sebastian Hilsenbeck et al. "Graph-based Data Fusion of Pedometer and WiFi Measurements for Mobile Indoor Positioning". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington: ACM, 2014, pp. 147–158.

[32] *ILSVRC2015 Results*. http://image-net.org/challenges/LSVRC/2015/results. 2015.

[33] Steven M Kay. *Fundamentals of Statistical Signal Processing: Practical Algorithm Development*. Vol. 3. Pearson Education, 2013.

[34] Christian Koehler et al. "Indoor-ALPS: An Adaptive Indoor Location Prediction System". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington: ACM, 2014, pp. 171–181.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[36] SangJeong Lee et al. "Understanding customer malling behavior in an urban shopping mall using smartphones". In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM. 2013, pp. 901–910.

[37] Seungwoo Lee et al. "Non-obstructive Room-level Locating System in Home Environments Using Activity Fingerprints from Smartwatch". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 939–950.

[38] *LIBSVM*. https://www.csie.ntu.edu.tw/~cjlin/libsvm/. 2016.

[39] Heng Liu et al. "Finding perfect rendezvous on the go: accurate mobile visual localization and its applications to routing". In: *Proceedings of the 20th ACM Multimedia Conference, MM '12, Nara, Japan, October 29 - November 02, 2012*. 2012, pp. 9–18.

[40] *Los Angeles County Fire Hydrant Layer*. http://egis3.lacounty.gov/dataportal/2012/05/23/los-angeles-county-fire-hydrant-layer/. 2012.

[41] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[42] Suhas Mathur et al. "Parknet: drive-by sensing of road-side parking statistics". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM. 2010, pp. 123–136.

[43] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B Moeslund. "Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey". In: *IEEE Transactions on Intelligent Transportation Systems* 13.4 (2012), pp. 1484–1497.

[44] Saman Naderiparizi et al. "Self-localizing Battery-free Cameras". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 445–449.

[45] Kazuya Ohara et al. "Transferring Positioning Model for Device-free Passive Indoor Localization". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 885–896.

[46] *OpenStreetMap*. http://www.openstreetmap.org/. 2016.

[47] Anindya S. Paul et al. "MobileRF: A Robust Device-free Tracking System Based on a Hybrid Neural Network HMM Classifier". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington: ACM, 2014, pp. 159–170.

[48] Charith Perera et al. "Context Aware Computing for The Internet of Things: A Survey". In: *IEEE Communications Surveys and Tutorials* 16.1 (2014), pp. 414–454.

[49] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). URL: http://arxiv.org/abs/1506.02640.

[50] *Subway*. http://www.subway.com/. 2016.

[51] Carlo Tomasi and Takeo Kanade. "Shape and motion from image streams under orthography: a factorization method". In: *International Journal of Computer Vision* 9.2 (1992), pp. 137–154.

[52] *u-blox GPS module*. https://www.u-blox.com/en/product/c94-m8p.

[53] *Understand Street View*. https://www.google.com/maps/streetview/understand/. 2016.

[54] Christopher Urmson et al. "Autonomous driving in urban environments: Boss and the Urban Challenge". In: *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I* 25.8 (2008). Ed. by Sanjiv Singh Martin Buehler Karl Lagnemma, pp. 425–466.

[55] *US Lane Width*. http://safety.fhwa.dot.gov/geometric/pubs/mitigationstrategies/chapter3/3_lanewidth.cfm. 2016.

[56] *US Sideway Guideline*. http://www.fhwa.dot.gov/environment/bicycle_pedestrian/publications/sidewalks/chap4a.cfm. 2016.

[57] Gonzalo Vaca-Castano, Amir Roshan Zamir, and Mubarak Shah. "City scale geo-spatial trajectory estimation of a moving camera". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1186–1193.

[58] Reza M Vaghefi, Mohammad Reza Gholami, and Erik G Ström. "Bearing-only target localization with uncertainties in observer position". In: *Personal, Indoor and Mobile Radio Communications Workshops (PIMRC Workshops), 2010 IEEE 21st International Symposium on*. IEEE. 2010, pp. 238–242.

[59] Katrien Verbert et al. "Context-Aware Recommender Systems for Learning: A Survey and Future Challenges". In: *IEEE Trans. Learn. Technol.* 5.4 (Jan. 2012), pp. 318–335.

[60] Paul Viola and Michael Jones. "Rapid Object Detection Using a Boosted Cascade of Simple Features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–511.

[61] Jan D. Wegner et al. "Cataloging Public Objects Using Aerial and Street-Level Images - Urban Trees". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[62] Muchen Wu, Parth H. Pathak, and Prasant Mohapatra. "Monitoring Building Door Events Using Barometer Sensor in Smartphones". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 319–323.

[63] Hongwei Xie et al. "MaLoc: A Practical Magnetic Fingerprinting Approach to Indoor Localization Using Smartphones". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '14. Seattle, Washington: ACM, 2014, pp. 243–253. DOI: 10.1145/2632048.2632057. URL: http://doi.acm.org/10.1145/2632048.2632057.

[64] Han Xu et al. "Enhancing Wifi-based Localization with Visual Clues". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 963–974.

[65] Qiang Xu, Rong Zheng, and Steve Hranilovic. "IDyLL: Indoor Localization Using Inertial and Light Sensors on Smartphones". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp '15. Osaka, Japan: ACM, 2015, pp. 307–318.

[66] *Yelp API*. https://www.yelp.com/developers/documentation/v2/search_api. 2016.

[67] Amir Roshan Zamir and Mubarak Shah. "Accurate Image Localization Based on Google Maps Street View". In: *Computer Vision–ECCV 2010*. Springer, 2010, pp. 255–268.