WhereAmI: Energy Efficient Positioning using Partial Textual Signatures

Quoc Duy Vo Darius Coelho Klaus Mueller Pradipta De
Department of Computer Science
The State University of New York, Korea
Email: {rayvo, dcoelho, mueller, pradipta.de}@sunykorea.ac.kr

***Abstract*—Positioning systems can use signatures hidden in a user's environment to identify a location. Images are often used to locate a place by identifying landmarks. In this work, we present the use of texts in an image to identify a user's location. The key intuition behind this work is that a collection of names of business appearing in an image forms a bag-of-words that provides a unique signature for a location. We use Optical Character Recognition (OCR) to detect the texts from an image. However, use of OCR in outdoor settings is resource intensive, and text detection is often error prone in uncontrolled settings. We develop an algorithm that can handle partial errors in the collection of business names to locate the user. Partial errors in text detection are handled by using similarity scores based on approximate text matching. We also limit the resource usage by partitioning the application between the smartphone and a cloud based web service to save energy. We have implemented the positioning system, called WhereAmI, on Android based smartphone. The experimental results show that WhereAmI can be an alternative positioning technique for GPS in terms of accuracy, precision, energy efficiency and positioning latency.**

***Keywords*—Positioning System, System Design, Computer Vision, OCR, Android, Mobile HCI, Web Service, Energy Efficiency**

I. Introduction

Smartphones equipped with GPS receivers have enabled many personal positioning systems and numerous location based services which depend on the location information. GPS is the most well-known technique for outdoor positioning with high accuracy. However it has a high battery drain [1], [2], moreover GPS availability is limited in many scenarios, like indoors, urban canyons, and unclear view of the sky. For many applications where accuracy can be compromised, other energy efficient approaches are preferred. Cellular network positioning is low in accuracy, but is more energy efficient [3]. WiFi positioning is more accurate than cellular network positioning, but it is not suitable for outdoor localization due to the lack of access points.

Presence of low cost sensors (e.g. accelerometer, gyroscope, camera) on smartphones has motivated research on positioning systems that can sense environment signatures for localization. Although these positioning systems may be less accurate than GPS, these are often more energy efficient [4]–[8]. Nowadays a camera is present in all smartphones. Images captured by a user can be analyzed to detect landmarks that identify a location. But such image processing techniques are compute intensive and slow, which leads to battery drain.

In this paper we present WhereAmI, a lightweight approach to enable image based positioning using smartphone cameras. We argue that the collection of names of business places, buildings, prominent points of interests, and other fixed textual markers appearing together in a single image forms a unique signature for a location. Our goal is to collect these textual signatures from a picture to generate a unique bag-of-words. The keywords are then matched against texts extracted from geotagged business names to determine the location. The text from the user generated image can either be determined automatically using Optical Character Recognition (OCR) or a user can aid the OCR process by identifying the text. Even if the user cannot read the language, it is easy to determine the textual signs. For example, a user trying to find her location on a map can take a picture of the neighborhood, mark the texts, and get the location. The process of matching is triggered by uploading only the texts from the phone, thereby reducing the bandwidth requirement compared to uploading an image. Even a short message (SMS) will be adequate to upload the texts. Matching texts is also faster compared to matching images.

There are two major challenges in implementing WhereAmI. First, OCR libraries for smartphone platforms are resource hungry and drain battery [9]. Therefore, performing complete text detection process using OCR on outdoor images on a smartphone is not an energy efficient solution. We propose a practical solution to easily identify text areas in an image captured by the user instead of using complex automatic text detection techniques. We also partition the text detection operation between a smartphone and a cloud based web service to reduce energy consumption. Secondly, text detection from natural scenes is a difficult problem and OCR libraries often return texts with errors. Therefore, our design must be able to work with partially correct text signatures. While matching the bag-of-words against the database of keywords, instead of searching for an exact match, we use results that approximately match and use the similarity score to derive the final result. We have implemented WhereAmI on Android based smartphones and evaluated its performance in different outdoor settings.

In summary, the main contributions of this work are:

- We design a text-based positioning system, that combines OCR on images captured by users using smartphone cameras, and matching collection of texts from the images to geotagged texts for positioning.
- We show how to overcome the limitations of OCR on outdoor images by identifying texts using user guidance. In order to overcome the errors in OCR based text recognition, we present a keyword matching algorithm that can handle partial errors in the query keywords.

- Based on an implementation of the system on an Android based smartphone, we evaluate the performance of WhereAmI in terms of positioning accuracy and precision, power consumption, and latency of positioning.

The remainder of this paper is organized as follows. Section II presents the system design of WhereAmI, followed by the implementation in Section III. Section IV reports experiment evaluation of WhereAmI. Prior researches on positioning relevant to our system are presented in Section V. We conclude in Section VI.

II. WhereAmI System Design

In this section, we present the design challenges of WhereAmI system. We also introduce the workflow and the detailed implementation of each component in the system.

A. Design Challenges

The key function of the front-end process is to extract texts from an image clicked by a user using a smartphone. Off-the-shelf OCR software works well for text extraction from scanned documents and for reading labels [10]. Recently, OCR have been also used for indoor localization, such as CheckInside proposed by M.Elhamshary et al, which leverages the crowd-sensed data collected by users' mobile devices to extract indoor business names for localization [11]. In outdoor scenes, however, the OCR performance degrades due to various uncontrolled parameters, like uneven lighting, complex backgrounds, varying fonts, and moving objects.

Many OCR-based text recognition techniques for outdoor environment have been proposed with the aids of image processing techniques [12]. Despite these advancements, the text extraction process using OCR is error prone. Therefore, a keyword matching algorithm must be designed to handle partial errors in keywords. The matching process must not depend only on exact match, and must use similarity scores denoting how closely an incorrect keyword matches the correct one in the database.

The success of the localization scheme hinges on the principle that a combination of names generates a unique signature of a place. However, if we consider popular store names around the world, then many of the names can occur in a group in the same location. For example, it is not unusual to find multiple coffee shops, like `Starbucks` and `Dunkin Donuts` in the same area. Although increasing the number of keywords in the combination can improve the accuracy, still a better option is to narrow down the search space with respect to the matching performance. We need an approximate location of the user which can be used as the anchor point to guide the matching process. This anchor point can be computed using approximate coarse localization available on most mobile devices, such as via cell tower triangulation or the location of the current cell that the mobile device is connecting to.

B. WhereAmI Workflow

WhereAmI consists of two main components: the front-end of WhereAmI is an Android application that runs on the user's devices (i.e. smartphone), and the backend component designed as a web service that executes the text matching service. The workflow begins with a user starting the WhereAmI app on her Android smartphone. The app provides the user an interface to click a picture of her surrounding area. Once the picture is saved, the text recognition process triggers.

In the next phase, the collection of texts extracted from the text recognition process is sent to the text matching service for positioning. There are two key steps in the text matching service: (a) cell tower based zoning to narrow down the search space, (b) matching texts against a database of geotagged texts within the zone to pinpoint a location. Fig. 1 shows the sequence of steps in the workflow.

C. Text Recognition Process

Three key steps in the text recognition process are, user guided image fragment determination, cleansing the fragments using a combination of image processing techniques, and text recognition using OCR.

1) User guided Image Fragment Extraction: Automatic text recognition using smartphones is computationally expensive and energy hungry, as well as, low on accuracy for natural scenes. To address this limitations, we introduce a human-in-the-loop step, which requires the user to highlight the relevant texts. User-in-the-loop approach helps in picking permanent signs that uniquely identify a location since a user taking a picture will be better at eliminating the temporary signs, like signs on moving vehicles, while highlighting the texts in the image. The highlights make it simpler to locate and construct a bounding box around the marked text area.

We use the flood fill algorithm to compute the extreme points of the highlight. The flood fill algorithm works by searching for a seed pixel (start point) for each highlight and testing whether its neighboring pixels belong to the highlight until it reaches the edges of the highlight. In our method we store the point on the screen that the user touches at the start of every highlight, and use the points as the seed pixels for each highlight. This eliminates the need to search for the seed pixel thereby reducing the processing time of the flood fill algorithm. We also handle overlapping highlights by merging them as they are considered as one continuous highlight. We use the computed extreme points to create a bounding box with additional 10 pixel padding. The bounding box areas are cropped from the original image and form the image fragments containing keywords chosen by user.

2) Image Fragment Cleansing: The cleansing step extracts the text from the fragments and removes irrelevant content, such as the background and other non-text objects, from the image segment. First we compensate for the uneven lighting that occurs in natural images by performing contrast enhancement on the fragments. Next, we apply Stroke Width Transform (SWT) to compute the text strokes in the fragment. SWT is a local image operator which can efficiently extract connected components during text detection [13]. Finally we binarize the stroke width image such that we are left with black text on white background. This process of binarizing the image improves the performance of the OCR tool.

3) Text Recognition using OCR: The OCR engine operates on binary image fragments containing a keyword. There are many standard libraries for OCR on smartphones and servers. The OCR engine should also support multiple languages. The two operations, viz. image fragment cleansing, and text recognition using OCR, are compute intensive. Hence performing the steps on a server can save smartphone battery at the cost

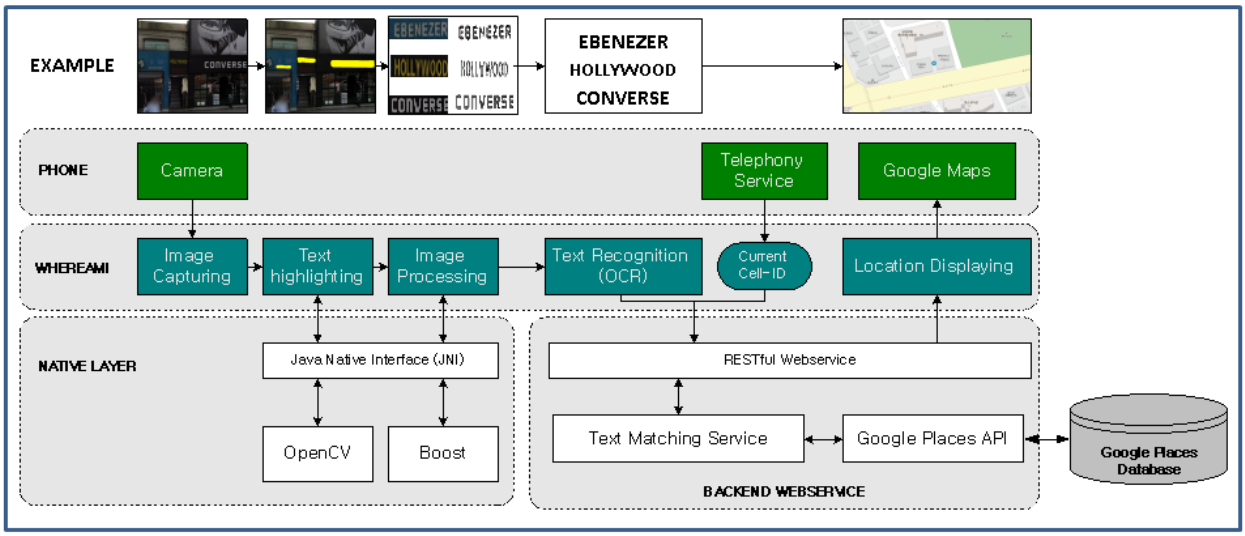


Fig. 1: WhereAmI system workflow showing the progression of steps in time. WhereAmI system architecture showing the steps in the system. The picture shows the correlation between the steps and its implementation.

of spending relatively less network card energy to upload few KiloByte sized segments to a web service.

D. Text Matching Service

There are two key steps in the text matching service - cell tower based zoning and the keyword matching algorithm.

1) Cell tower based Zoning: In our system, we use the location of the cell tower in order to limit the search space. This location can be obtained by sending a request to Google Maps GeoLocation API [1] with the telephony service information on the smartphone. Using this coarse location, the matching algorithm is guided to search only the business names whose coordinates are within a fixed radius of the cell tower.

2) Keyword Matching Algorithm: The keyword matching algorithm takes as input the set of keywords returned by the text recognition phase, and the coordinates of the cell tower. It also requires two parameters, *SearchRadius* and *NeighborRadius*. The *SearchRadius* determines the distance from the cell tower within which we want to search for the combination of texts. It is usually the coverage radius of the cell tower set in kilometer range. The *NeighborRadius* is used to find the names which are clustered near each other and can lead to a solution. Hence it is chosen in the range of meters. The algorithm, as shown in Algorithm 1, begins by generating vectors of strings, P_i, for each keyword, K_i (Line 2). The *GText* function in Line 2 returns a set of business names matching a keyword within *searchRadius* distance around the anchor point. It performs an approximate match, and the results are ordered according to the closeness of match. Thus, P_i contains the names of all businesses within the *SearchRadius* that matches K_i. Since OCR may not detect all keywords correctly, an attempt to match K_i exactly to the names of the businesses can fail. If there are multiple business with similar names, K_i, then all of them are returned as separate strings in P_i. Each of these strings denotes a candidate location around which the other keywords could be found, and lead to a match.

[1]Google-Geolocation: https://developers.google.com/maps/

Since there can be partially correct keywords, we approximately match the text where partially matching keywords or business names are returned along with a similarity score. As the number of the business names returned by matching one keyword increases, the algorithm will take longer to terminate, increasing system response time. Therefore, we reduce the size of the result set by selecting only businesses which have the same or similar names to the keyword K_i using a *SimilarityFilter* function (Line 36). The selection is governed by a threshold, called $FilterThreshold$. Higher value of $FilterThreshold$ blocks more keywords returned by *GText*, and the algorithm terminates faster. However, aggressive blocking may eliminate the actual business name we are interested in since an erroneous keyword may match some other keyword with higher similarity value and degrade the positioning accuracy. After the filtering process based on *SimilarityFilter* function, the set of remaining business names is sorted according to distance from the anchor (Line 5). It is now possible to combine this location coordinates to infer the user's location. Note that, smaller cardinality of P_i indicates that K_i is a less common business name in the area. Therefore, beginning the search in the neighborhood of a least common K_i will converge faster. Hence we sort the P_is based on the cardinality of the P_is (Line 8).

In the procedure *searchNeighbor*, first a business name, say ϕ, from P_i is picked in the order they were retrieved. The *GNeighbor* function takes as input the location attribute of ϕ, and retrieves all the business names around ϕ within *neighborRadius* distance (Line 20). If ϕ is in the area where the user is, then the neighborhood business names vector, N_j, should contain the rest of the keywords. Line 22 searches for this match across $n-1$ P_is corresponding to the remaining K_is. If all the keywords are matched, the search terminates and the *ResultSet* contains all the businesses which the user marked and is in her surrounding. We compute a bounding box using all the locations in the *ResultSet* and assign the midpoint as the user's location on the map.

However, if any one keyword is inaccurate, then *GText*

Algorithm 1 *WhereAmI Keyword Matching Algorithm*

Require: A list of n keywords $K = K_1, K_2, ...K_n$
Require: Coordinates of the cell tower: (Coord(T) = T_{lat}, T_{lon}
Require: $SearchRadius$, $NeigborRadius$, $FilterThreshold$

1: **for** i = 1 to n **do**
2: P_i = *GText*(K_i, Coord(T), SearchRadius)
3: P_i = SimilarityFilter(P_i, K_i)
4: /* sort elements in P_i in increasing distance to the anchor */
5: $P_i = \pi_1^m(P_i^j)$ /* m is the number of elements in P_i
6: **end for**
7: /* sort P_is in increasing count of elements per set */
8: $P = \pi_1^n(P_i)$
9: ResultSet = {} /*ResultSet will be used to compute the location*/
10: **for** i = 1 to n **do**
11: Set S = P_i
12: located = searchNeighbor(S, i)
13: **if** located == TRUE **then**
14: break;
15: **end if**
16: **end for**

17: **procedure** Boolean searchNeighbor(Set S, int keyId)
18: **for** j = 1 to $|S|$ **do**
19: curResultSet = S[j]
20: N_j = *GNeighbor*(Coord(S[j]), NeighborRadius)
21: **for** (z = 1 to n) && (z $\neq$ keyId) **do**
22: **if** $(N_j \bigcap P_z) \neq$ NULL **then**
23: curResultSet = curResultSet $\bigcup (N_j \bigcap P_z)$
24: **end if**
25: **end for**
26: **if** $|curResultSet|$ == n **then**
27: ResultSet = curResultSet;
28: RETURN TRUE;
29: **end if**
30: **if** $|curResultSet| > |ResultSet|$ **then**
31: ResultSet = curResultSet;
32: **end if**
33: **end for**
34: RETURN FALSE
35: **end Procedure**

36: **procedure** Set SimilarityFilter(Set P, keyword)
37: perfectSet = {} ; filterSet = {}
38: **for** k = 1 to $|P|$ **do**
39: *Similarity*(P[k], keyword)
40: **if** $P[k].Similarity == 1$ **then**
41: perfectSet.add(P[k])
42: **end if**
43: **if** $P[k].Similarity >= FilterThreshold$ **then**
44: filterSet.add(P[k])
45: **end if**
46: **end for**
47: **if** $|perfectSet| > 0$ **then**
48: RETURN perfectSet
49: **else**
50: RETURN filterSet
51: **end if**
52: **end Procedure**

53: *GText*(**keyword, anchor, SearchRadius**):
54: returns a set of locations matching keyword K_i within *SearchRadius* from *anchor*.

55: *GNeighbor*(**anchor, radius**):
56: returns a set of locations within *radius* distance from *anchor*

would not retrieve any business name that exactly matches the keyword. Instead it will return all the business names closely matching the keyword. This leads to Line 22 returning null at least once implying that a keyword is not in the cluster, although the rest of the keywords may have matched. We ensure that in such a scenario we take the largest cluster (Line 30). However, if the cluster size is less than $n/2$, then we report failure to localize. In case of a failure, the user is sent a feedback to take another picture, and the process is repeated.

Note that although the complexity of the algorithm is $O(n^2\,|S|)$, we found empirically that typically 3 keywords are sufficient for the algorithm to locate a user's location.

III. WhereAmI System Implementation

In this section, we present the implementation of WhereAmI, as shown in Fig. 1.

A. Text Recognition Process

The text recognition process including image fragment extraction, image fragment cleansing, and text recognition, can be implemented entirely on the mobile Android platform. Since this implementation requires C++ source libraries, we use the Android Native Development Kit (NDK) to compile the libraries. In order to implement the highlighting feature, we use the OpenCV library which provides all of the image processing functionalities. The implementation of the image fragment cleansing uses the SWT source code from Kumar and Perrault [14]. Kumar's implementation additionally requires the Boost C++ libraries[2], along with OpenCV. The OCR functionality uses the open-source Tesseract OCR engine[3], which has multi-platform support, including Android, and provides multi-language capability.

In the alternative implementation, the OCR process is performed as a Web service with respect to energy efficiency. The client application uploads image fragments that are extracted based on keywords highlighted by a user in an image. At the Web service, the texts are extracted from the fragments using the same OCR engine (Tesseract OCR). The client application uploads image fragments that are extracted based on keywords highlighted by a user in an image.

B. Location Determination

The implementation of the matching algorithm requires access to a database with names of business along with their geolocation. Google Places provides the API to access a large database of business names along with their location coordinates. We implement the *GText* and *GNeighbor* functions mentioned in the algorithm using Google Places API. *GText* allows to search for a specific business name within a given radius from the anchor location. *GNeighbor* allows to search for all business within a given radius for each specific business name. The maximum radius is limited to 20km.

A limitation of the Google API used to implement *GText* function is that one query sent to the API can get a maximum result set of 60 business names. Due to this limitation, the *NeighborRadius* parameter while searching the neighborhood must be chosen carefully. If the radius is too small, then the business names from the image may not be considered in the result. We uses 20m as *NeighborRadius* in our algorithm.

[2]Boost C++ libraries - http://www.boost.org/
[3]Tesseract - https://code.google.com/p/tesseract-ocr/

IV. System Evaluation

In this section, we first present the performance of the text recognition process proposed in our system. Next, we evaluate WhereAmI with respect to positioning accuracy, precision, energy efficiency, and latency. The experiments were done at different outdoor areas in Seoul downtown and suburban areas under different external settings, like areas with high density of buildings, different weather conditions, and time of day.

A. Text Recognition Performance

We evaluate the accuracy of Tesseract OCR library and show the effectiveness of the cleansing step to improve text detection accuracy. We also generate an error model to represent the extent of error in the text recognition process.

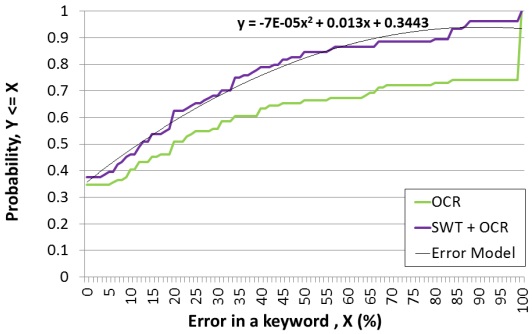


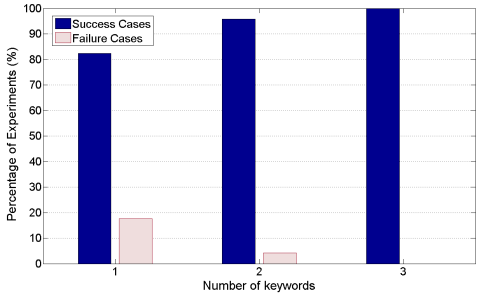
Fig. 2: Probability of the degree of error in a keyword while using OCR for text detection in two scenarios - OCR with and without image cleansing. The equation for error model of OCR with image cleansing is shown.

We create our test set by collecting images acquired using WhereAmI and images from the ICDAR datasets and Google Street View. Text fragments containing keywords are extracted from these images using WhereAmI's fragment extraction process. In order to measure the extent of errors introduced by the text detection process, we measure the number of characters that are recognized incorrectly, and report the percentage error. We conducted two experiments - OCR on image fragments with and without the cleansing step to determine the usefulness of the cleansing step to improve text detection accuracy.

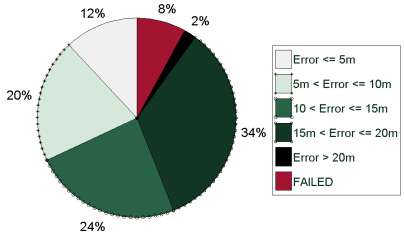
Fig. 2 shows the cumulative distribution of the percentage error in text detection for the two experiment scenarios. The figure shows that 37% of keywords in the dataset images could be recognized without any error, while 75% of the keywords could be detected with at most 34% error using the cleansing step. But without the cleansing step, 34% of keywords could be recognized without any error and 75% of the keywords could be detected with at most 85% error. We conclude that the cleansing step improves the accuracy of the text recognition process. For further experiments, we fit a curve to the CDF plot that acts as the error model ($y = -7E\text{-}05x^2 + 0.013x + 0.3443$) for the text recognition with cleansing process (SWT + OCR).

B. Positioning Accuracy

Positioning accuracy of WhereAmI is evaluated by comparing the ground truth location of the user to the one reported by WhereAmI. Due to the fact that Google Places and Google Earth share the same databases of business places, we use Google Earth to pinpoint a specific location and obtain its coordinates as the ground truth location. With the reported granularity of Google Earth images[4], we can determine the actual position of the user within an average error of 0.5 meter.

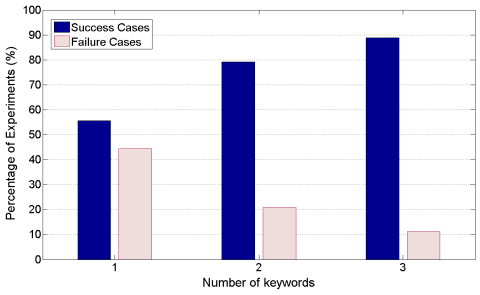


(a) Success and Failure rate of WhereAmI in percentage.

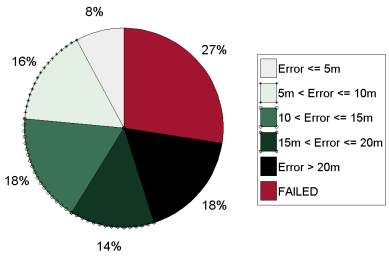


(b) Distribution of the positioning error in terms of distance from actual location.

Fig. 3: Positioning accuracy of WhereAmI when all keywords are correct.



(a) Success and Failure rate of WhereAmI in percentage.



(b) Distribution of the positioning error in terms of distance from actual location.

Fig. 4: Positioning accuracy of WhereAmI when at least one keyword is incorrect.

We conduct a real walk to collect images of signs at 71 outdoor locations across downtown and suburbs. Using keywords extracted from the collected images, we first measure the capability to identify the user's location and the positioning

[4]Google Earth: https://www.google.com/earth/media/features.html

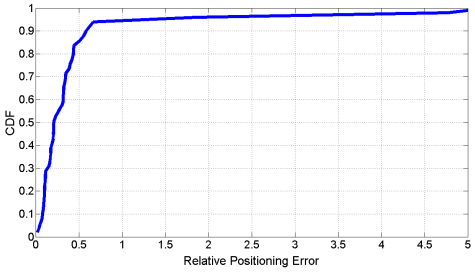


Fig. 5: Positioning error of WhereAmI relative to positioning error of GPS.

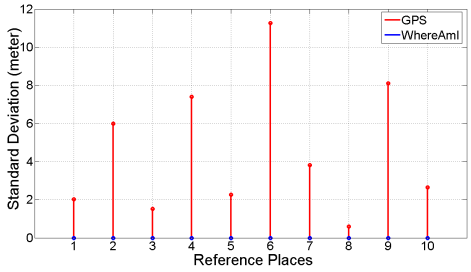


Fig. 6: Comparison of standard deviation of GPS positions with respect to WhereAmI location for each reference place.

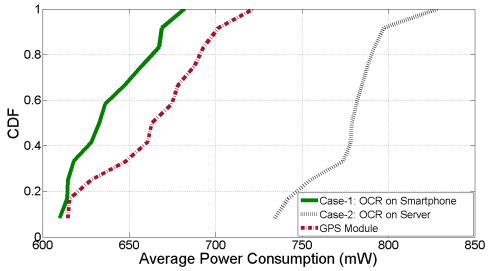


Fig. 7: Power consumption of WhereAmI over 3G network, compared to that of GPS.

error of WhereAmI, assuming OCR works perfect. Next, we show how the matching algorithm can handle incorrect or partial keywords, assuming there are some errors in OCR. Finally, we compare the positioning error of WhereAmI with respect to GPS. For the purpose of experiments, we also implement a text input interface in WhereAmI such that a user can input the names of the business places for localization. It enables controlled experiments with or without OCR errors.

1) Measuring positioning accuracy using correct keywords: The success/failure rate of WhereAmI is shown in Fig. 3a. As the number of keywords increases in one request, success rate of the matching algorithm improves. Fig. 3b summarizes the statistics of positioning error. The figure shows that WhereAmI can locate a position within 20m accuracy in 90% cases, while positioning failed in 8% cases. On average, WhereAmI achieves positioning error of about 11.8 meters. When a business name is not registered in Google Places, WhereAmI fails. In 2% cases positioning error was greater than 20 meters. When only one keyword is used, and it is not unique in the neighborhood, multiple locations are returned by Google Place API. It is not possible to identify which location is correct. The location, which may not be the correct location but closest to the cell tower, will be selected.

2) Measuring positioning accuracy using incorrect keywords or partial keyword: Using the error model of the text recognition process, as explained in Section IV-A, errors are introduced in the keywords. We randomly pick a value between 0 and 1, and calculated the percentage of error in one keyword. Using the percentage of error, we changed a few alphabets to add an error to the keyword before sending it to the keyword matching algorithm. For example, given a business name, `JNM Fashion`, we choose a random number 0.6, which gives the error rate as 22%. We change two characters, which is 20% of the length of the string, viz. `JMM Fashian`.

Fig. 4a and Fig. 4b show the success/failure rate and the positioning error of WhereAmI, respectively. The success rate increases when the number of keywords increases. However, the failure rate is higher than that of cases where only correct keywords are used. Two factors contribute to the high failure rate. First, a business name may not be registered with Google Places database. Second, the Google Places API fails to retrieve any business name due to errors in the keywords.

3) Comparison of Positioning Error: We compare the positioning error of WhereAmI with respect to GPS using the following metric, *Relative Positioning Error* (RPE).

$$RPE = \frac{Positioning\ Error_{WhereAmI}}{Positioning\ Error_{GPS}}$$

The location returned by GPS is obtained using the Location Manager module of the Android Framework. In each experiment, we record the location using both WhereAmI and GPS.

Fig. 5 shows the relative positioning error between WhereAmI and GPS. When RPE is less than 1, the positioning error of WhereAmI is smaller than that of GPS. The figure shows that in about 95% of the cases WhereAmI returns lower positioning error compared to GPS. This is because positioning using WhereAmI is based on getting the keywords and successfully matching them against Google Places. Unlike GPS, WhereAmI is not affected by environmental conditions (e.g. high density of buildings, bad weather condition).

C. Positioning Precision

In this section, we evaluate the positioning precision of WhereAmI, compared to that of GPS. We experiment WhereAmI at 10 reference places which have different building densities. At each reference place, we measured the locations returned by both GPS and WhereAmI multiple times using the same set of business names, but under different settings (e.g. weather conditions, times of a day, different days).

Fig. 6 shows the standard deviation of GPS positions compared to that of WhereAmI location for each reference place. The geo-coordinates returned by WhereAmI at each place do not change regardless of different settings. This is due to the fact that WhereAmI positioning technique is based on getting the keywords and successfully matching them. Therefore positioning using WhereAmI is precise as long as the business names in the area remain unchanged. However, the geo-coordinates obtained by the GPS module at each place varies every time the settings change. This is because GPS signals are distorted by the environmental conditions.

D. Power Consumption

We measure the power consumption of WhereAmI in two different cases: *Case-1* where the OCR process is performed on

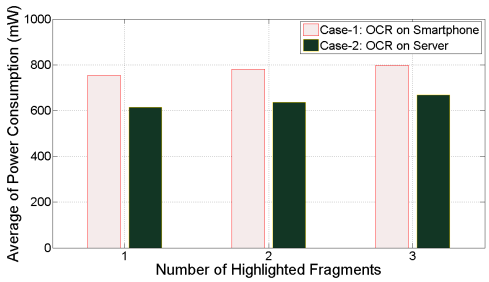


Fig. 8: Power consumption of WhereAmI in two cases (*Case-1* OCR on the smartphone and *Case-2* OCR on the server).

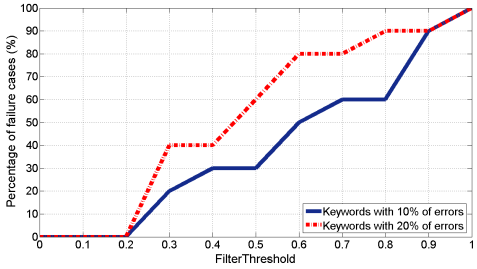


Fig. 9: Comparison of the failure rate in two scenarios: (i) keywords with 10% error, and (ii) keywords with 20% error.

the smartphone and the keywords are uploaded for positioning, *Case-2* where highlighted fragments are uploaded to the server, and the OCR process is performed on the server. We also compare the power consumption of WhereAmI in both cases to GPS. Finally, we evaluate the effect of increasing number of highlighted fragments on power consumption.

We use Monsoon Power Monitor[5] to measure the power consumption of WhereAmI over 3G network. We experiment WhereAmI using images consisting of multiple business names at 16 different places. At each place, we also profile the power consumption of GPS on the same test phone. After the GPS module is called, it starts receiving GPS signals to compute the location. Due to the environmental conditions, the GPS module waits for the signals until it receives enough information to calculate the location. Since the power consumption of the test phone fluctuates during the positioning process and remains stable after the positioning is successful, we can easily isolate the power consumption of the GPS module.

Fig. 7 shows the comparison of the power consumption of WhereAmI in both cases versus GPS. On average GPS consumes 665mW of power and WhereAmI consumes 778mW of power in *Case-1*, while there is only 639mW consumed in *Case-2*. The experimental results show that the average power consumption of GPS is less than that of WhereAmI in *Case-2*, but higher than that of WhereAmI in *Case-1*. In other words, WhereAmI with the OCR process on the server could be used as an energy efficient positioning technique, compared to GPS.

To evaluate the effect of increasing number of highlighted fragments on power consumption, we experiment WhereAmI with an increasing number of highlighted fragments in each experiment. Fig. 8 shows that positioning using *Case-1* consumes more power than positioning using *Case-2*. Although sending a group of texts consumes much less power than sending crops of fragments, the power consumption required for the text detection process is much higher than the power required to send the crops of fragments.

E. Response Time

In this section, we evaluate the response time of matching algorithm, compared to that of GPS. The response time is computed with respect to the increasing number of keywords.

Table I shows that the number of keywords increases, the response time increases. In the worst case, it could take close to 8 seconds to locate the user. Since none of the keywords are unique, the algorithm needs to search for matches in multiple neighborhoods, hence taking longer. On the other hand, some queries could run fast even if the number of keywords in the query increases. This is because some keywords in the query can be unique or less common keywords, and the algorithm converged quickly in the first few iterations. On average, the matching algorithm takes about 2.9 seconds for positioning.

	Mean (s)	Standard Deviation (s)	Min (s)	Max (s)
1 keyword	1.7	0.2	1.4	2.3
2 keywords	3.0	1.1	1.8	5.9
3 keywords	3.9	1.2	2.3	7.4
GPS	12.5	1.8	10.3	15

TABLE I: Numerical summary of the response time of the matching algorithm compared to GPS.

We also measure how long GPS takes to identify the location. The experimental results show that the GPS module needs around 12.5 seconds on average to complete the positioning. In comparison with WhereAmI, GPS takes longer to find the location due to the fact that it has to wait until it receives the satellite's signal before computing the location. It takes more time to complete the positioning in cases where the satellite's signal is blocked by an obstruction or interference.

F. Sensitivity Analysis

We verify the effect of $FilterThreshold$ on the failure rate with increasing errors in keywords. As mentioned in Section II-D2, selecting the value of $FilterThreshold$ affects the performance of the keyword matching algorithm. When the input keywords are partially incorrect, increasing the value of $FilterThreshold$ can terminate the algorithm faster. However, this may degrade the positioning accuracy of the algorithm.

We used 14 test images which consist of multiple business names. From each image, we extracted a collection of the business names appearing in the image. Since keywords with high percentage of errors may not be identified by Google Places, we only introduced 10% and 20% error to the keywords to generate two sets of partially incorrect keywords.

Fig. 9 shows how increasing the value of $FilterThreshold$ impacts positioning performance as the error rate in keywords goes up. We evaluate two error rates: (i) keywords with 10% error, and (ii) keywords with 20% error. In both scenarios, the failure rate increases when the threshold increases. Since there is an error in the input keyword, its similarity index is always less than 1. Therefore, as the threshold increases, the keyword with lower similarity index is blocked.

[5]Monsoon Power Monitor http://www.msoon.com

The failure rate in the first scenario is less than in the second scenario. As the keyword error rate increases, the failure rate increases. With higher errors in a keyword, the similarity index decreases and the algorithm discards keywords that do not match closely. Therefore, as the threshold increases, the keywords with higher errors are blocked. However, since we start with partially correct keywords, the filtering process may discard the actual business name we are interested in. We observe that the matching algorithm successfully identifies the location when the threshold is less than or equal to 0.2 in all cases. Therefore, we set the default value of $FilterThreshold$ to be 0.2 in the prototype.

V. Related Work

Presence of multiple sensors in smartphones has opened up the design space of fingerprint-based localization. The key idea behind is to discover signatures of a location, and match the observed signatures against a database of geotagged signatures. Several works have been explored using signatures in different forms for both indoor and outdoor positioning [4], [15]–[17]. SurroundSense explores the presence of signatures in logical labels of a place in the form of sound, light, color [4]. It is targeted towards indoor environments for precise positioning. Wireless signals also can be used as fingerprints as signatures for localization in recent decades [16]. Other works using images of a location for matching come close to our work [15], [17]. The idea is to match an image of landmarks against a database of geotagged images to find the place. Schroth et al. proposed a localization technique for large scale image retrieval by segmenting a large database of geotagged images into overlapping cells to narrow down the search space, with respect to the localization performance [15]. These approaches which work on the whole image is compute intensive, and were not designed as a lightweight system suitable for mobile devices. WhereAmI is proposed as a lightweight localization technique which uses only texts extracted from taken images.

Given the choice of different modes of outdoor positioning, there is a tradeoff between energy and accuracy in designing a system. Dejavu uses different features of road landmarks (i.e. tunnels, bumps, bridges, and even potholes) as signatures to provide both accurate and energy-efficiency localization [18]. Other works have explored fingerprint-based techniques to improve the energy efficiency of GPS. RAPS proposes a rate adaptation of GPS sensing using hints gathered by inertial sensors [5]. CAPS uses the daily trajectory of users, and cell tower signals to improve over GSM based positioning [6]. Inertial sensors are used for positioning in SmartLoc [8]. Map matching and dead reckoning algorithms are combined in APT for localization [7]. Energy optimization of GPS has also been explored by offloading the position calculation steps to the cloud in LEAP [19]. WhereAmI uses a similar idea of leveraging cloud resources to improve the energy efficiency.

VI. Conclusion

The ubiquitous presence of camera in smartphones motivates the use of images captured by users for positioning. However, visual location recognition techniques are compute intensive. In this work, we present WhereAmI, a lightweight positioning technique that uses texts appearing together in an outdoor image as a unique signature for localization. We proposed a keyword matching algorithm that can handle partial errors in the detected text by matching approximately to texts from the database of business names. We have implemented WhereAmI on Android smartphones. Compared to GPS, WhereAmI has lower response time, less energy consumption, and better position accuracy in urban areas. But it requires higher amount of user involvement than in GPS.

Acknowledgement

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the "ICT Consilience Creative Program" (IITP-2015-R0346-15-1007) supervised by the IITP(Institute for Information and Communications Technology Promotion)"

References

[1] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær, "Entracked: Energy-efficient robust position tracking for mobile devices," in *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys, 2009.

[2] Z. Zhuang, K.-H. Kim, and J. P. Singh, "Improving energy efficiency of location sensing on smartphones," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys, 2010.

[3] Y. Jie, V. Alexander, L. Hongbo, C. Yingying, and G. Marco, "Accuracy characterization of cell tower localization," in *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, ser. Ubicomp, 2010.

[4] M. Azizyan, I. Constandache, and R. Roy Choudhury, "Surroundsense: Mobile phone localization via ambience fingerprinting," in *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom, 2009.

[5] J. Paek, J. Kim, and R. Govindan, "Energy-efficient rate-adpative gps-based positioning for smartphones," in *MobiSys*, 2010.

[6] J. Paek, K.-H. Kim, J. P. Singh, and R. Govindan, "Energy-efficient positioning for smartphones using cell-id sequence matching," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys, 2011.

[7] X. Zhu, Q. Li, and G. Chen, "Apt: Accurate outdoor pedestrian tracking with smartphones." in *INFOCOM*, 2013.

[8] C. Bo, X.-Y. Li, T. Jung, X. Mao, Y. Tao, and L. Yao, "Smartloc: Push the limit of the inertial sensor based metropolitan localization using smartphone," in *Mobicom*, 2013.

[9] R. LiKamWa, E. Reyes, and L. Zhong, "Poster: Retrofitting computer vision libraries for concurrent support on mobile devices," in *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '14, 2014.

[10] K. Jung, K. I. Kim, and A. K. Jain, "Text information extraction in images and video: a survey," in *Pattern Recognition 37(5): 977-997*, 2004.

[11] M. Elhamshary and M. Youssef, "Checkinside: A fine-grained indoor location-based social network," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '14. New York, NY, USA: ACM, 2014, pp. 607–618.

[12] A. Shahab, F. Shafait, and A. Dengel, "Icdar 2011 robust reading competition challenge 2: Reading text in scene images," in *International Journal of Document Analysis and Recognition*, 2011.

[13] B. Epshtein, E. Ofek, and Y. Wexler, "Detecting text in natural scenes with stroke width transform," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[14] "Text detection on nokia n900 using stroke width transform," https://sites.google.com/site/roboticssaurav/strokewidthnokia.

[15] G. Schroth, R. Huitl, D. Chen, M. Abu-Alqumsan, A. Al-Nuaimi, and E. Steinbach, "Mobile visual location recognition," *Signal Processing Magazine, IEEE*, vol. 28, no. 4, pp. 77–89, July 2011.

[16] V. Honkavirta, T. Perl, S. Ali-Lytty, and R. Pich, "A comparative survey of wlan location fingerprinting methods." in *WPNC*, 2009.

[17] K.-H. Yap, T. Chen, Z. Li, and K. Wu, "A comparative study of mobile-based landmark recognition techniques," *Intelligent Systems, IEEE*, vol. 25, no. 1, pp. 48–57, Jan 2010.

[18] A. Heba and Y. Moustafa, "Dejavu: An accurate energy-efficient outdoor localization system," in *ACM SIGSPATIAL GIS*, 2013.

[19] H. S. Ramos, T. Zhang, J. Liu, N. B. Priyantha, and A. Kansal, "Leap: a low energy assisted gps for trajectory-based services." in *Ubicomp*, 2011.