VirtualTag: Enabling Indoor Augmented Reality with Marker-Facilitated Localization

Anirudh C. Mohan[*]
Duke University
Advisors: Dr. Romit Roy Choudhury[†]and Dr. John Board[‡]

Submitted for Graduation with Departmental Distinction
in Electrical and Computer Engineering

April 16, 2014

[*]Can be reached at anirudh.mohan@duke.edu
[†]University of Illinois at Urbana Champaign
[‡]Duke University

Contents

1 Abstract

Indoor augmented reality (AR) has become increasingly important in both research and software development communities – various consumer and enterprise applications could be built on an indoor AR platform. However, many existing strategies that have been used to develop outdoor augmented reality platforms do not translate well to indoor environments. This is for two reasons: (1) typical localization approaches such as GPS work poorly indoors and (2) objects are distributed and arranged differently indoors.

I here propose VirtualTag, an indoor AR system that incorporates a new method of indoor localization along with a computer-vision based approach. I demonstrate the efficacy of VirtualTag on both the Android and iOS mobile platforms and describe areas in which the system can be further developed in order to be deployed robustly to consumers in the future.

2 Introduction

2.1 Augmented Reality: Overview

Augmented reality has been hailed as one of the next major platforms that will revolutionize how we interact with software applications. AR offers users a view into the physical world augmented by computer-generated data such as sound, images, or GPS data. AR will enable the virtual and physical worlds to interact in a heretofore unprecedented way – users can directly associate digital content with physical objects. Relevant applications of AR include immersive commercial applications (*e.g.* interactive coupons in shopping malls), self-guided museum and real estate tours, and virtual Post-It notes.

2.2 Outdoor AR

Existing work in AR has focused primarily on outdoor applications [1, 2, 3]. ARQuake, for example, uniquely represents user-generated "virtual" tags for subsequent retrieval using feature extraction tuples, namely two-dimensional vectors of the following form: <features, user content>, where features denote the key visual identifiers of the object being tagged [1]. The popular Google Googles application that enables users to identify outdoor landmarks and buildings utilizes this tagging scheme [4]. On the other hand, Reitmayr and Drummond propose a different tuple storage-based system: <location, user content>, where location denotes the physical coordinates (*e.g.* GPS coordinates) at which the object lies [2].

2.3 Indoor AR

Little progress has been made on extending AR to indoor environments, primarily because the distribution of "noteworthy" or "taggable" objects differs greatly indoors versus outdoors. For example, relevant objects might be located far closer together indoors versus outdoors (*e.g.* two adjacent chairs are closer together than two adjacent buildings). Thus, simple image feature extraction is no longer sufficient, as objects may be too densely compacted to produce a "features vector" that remains free from visual noise. Additionally, GPS no longer serves as a reliable means of obtaining object locations. At worst case, GPS provides only a 8 meter resolution with 98% confidence – objects are typically located more closely together indoors [5]. Choosing the appropriate tuple representation of a tagged object is thus no longer a simple task.

2.3.1 Geometric Approach

Some indoor AR systems, such as the one proposed by Jang [6], adopt a purely geometric approach. Jang suggests localizing users within an indoor environment using a known reference point, such as a QR code. As seen in Figure 1, user A may scan (via a mobile device) a QR code containing metadata about its own location. The user may then calculate his position according to how large or small the code appears on his screen.

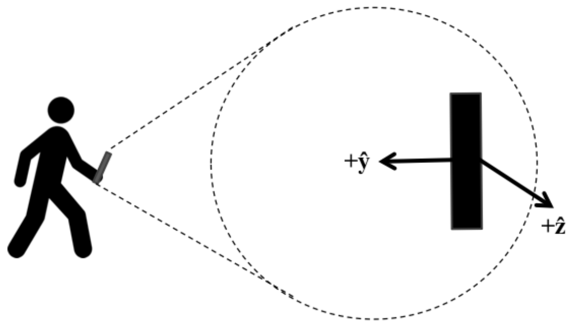

Figure 1: User localizing himself by scanning marker

Next, user A might choose to tag a particular object in his surrounding environment, as depicted in Figure 2. Given that the AR system knows the user's location, the system can represent the tagged object as a line segment (of some finite length l set to the same length scale as the room's size) with one endpoint defined as the user's location – $(x_{coordinate}, y_{coordinate})$ – and the other endpoint is l away in the direction being faced by the user. Thus, the object's tuple identifier would be of the form <tag line segment, user content>, where "tag line segment" consists of the 3-dimensional vector $< x_{coordinate}, y_{coordinate},$ direction angle>.

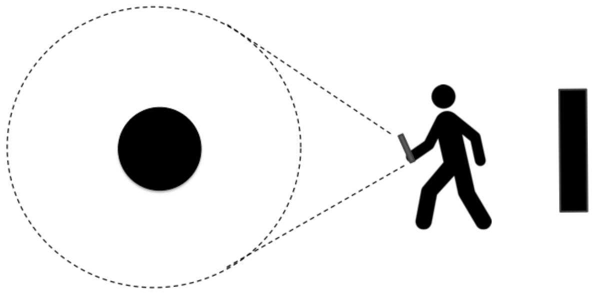

Figure 2: User tagging circular object after self-localization

Subsequently, a new user B could enter the room, and after undergoing a similar localization process to the one carried out by user A in Figure 1, could scan the room searching for tagged objects. The AR system generates a location line segment for user B at a given sampling rate. If user B's line segment intersects any existing line segments corresponding to object tags, user B successfully retrieves a tagged object, and user A's content is displayed.

A key problem that arises, however, is the scenario in which multiple objects might lie along the same axis within the room (*i.e.* along user A's location line segment, as in Figure 3). As a result, when user B attempts to retrieve an object tag, the AR system might yield a false positive when user B's location line segment intersects any point along user A's line segment. This problem (what I call the infinite intersection problem) exists since the object's distance from the user's mobile device is not being stored. Manweiler, *et. al.* propose an object positioning system that determines distances between a mobile device and arbitrary objects in the environment [8], but the details of the system are not sufficiently refined for implementation. As a result, the geometric approach alone is insufficient for robust indoor AR.

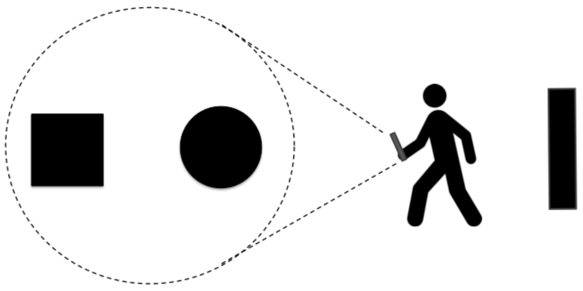

Figure 3: Infinite intersection problem

2.3.2 Computer Vision Approach

Tenmoku, *et. al.* propose an alternative indoor AR system that primarily employs computer vision techniques to represent tagged objects [8]. The system bears remarkable similarity to the one proposed by Reitmayr and Drummond for outdoor environments, with the sole difference being the use of a feature detection filter that is less sensitive to ambient lightning conditions. Since objects have unique edges, corners, colors, and other distinguishing feature points, Tenmoku attempts to characterize all object tags through these features.

However, unlike in the outdoor case, it is common to have identical copies of a single object in indoor environments (*e.g.* two identical adjacent chairs alongside a table). Given that both objects will return highly similar sets of detected features, a solely vision-based approach will not guarantee the system's ability to differentiate between distinct objects.

3 System Design

My proposal with VirtualTag is to merge the geometric and vision-based approaches to allow for: (1) computational scalability as the number of tags in a given room increases while (2) minimizing the number of false positive "tag retrievals" (*i.e.* ensuring that tag messages are only displayed for the appropriate objects). The infinite intersection problem posed by a purely geometric approach can be resolved with feature detection techniques, assuming that along a given location line segment there lie sufficiently dissimilar objects. Thus, although a room may contain multiple copies of the same object, I am making the assumption here that the likelihood of duplicate objects lying on user A's location line segment is low.

VirtualTag consists of multiple stages, as seen in Figure 4. I will here describe the typical operational flow of the VirtualTag system for two users A (the object tagger) and B (object retriever). I will then delve deeper into three key focus areas of the system.

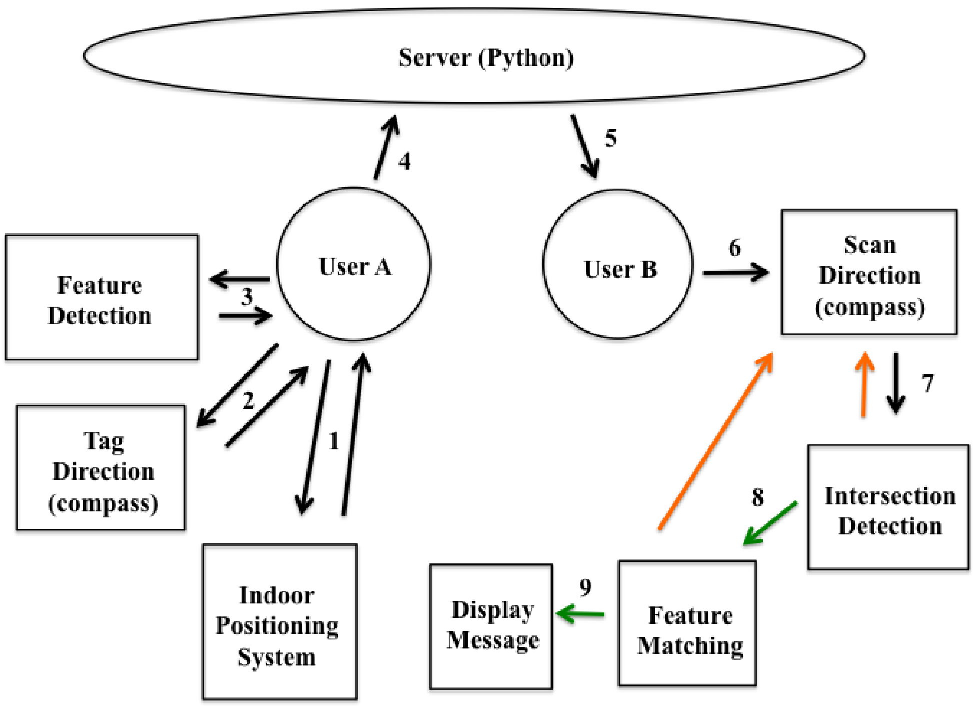

Figure 4: VirtualTag system diagram

User A begins by scanning a marker embedded on the walls or ambient environment of the room; by doing so, the system can successfully apply an indoor positioning algorithm to determine the user's location. Next, user A stands in place and rotates his mobile device around the room along a single lateral axis of rotation – VirtualTag does not currently allow the user to move once his initial location is computed. When user A views the object which he wishes to tag on his mobile device, he may submit a tag request to the VirtualTag server. This request contains user A's location, the direction he is facing, relevant image features of the object from his perspective, and any user-generated content that he might include.

User B may then enter the room and scan the embedded marker so that the system may determine his location (which may or may not be different from user A's location). Next, user B rotates his mobile device to scan the environment for object tags. As he does so, all metadata for object tags associated with the scanned marker will be downloaded to his mobile device from the VirtualTag server. At regular intervals, VirtualTag determines whether user B's line of sight intersects that of user A's when he tagged the object. If so, an image similarity recognition algorithm is applied between the downloaded image features from user A and the features of the current image being viewed by user B. User A's content is transmitted to user B if the images are sufficiently similar.

I will now describe the technical details of three different aspects of the system: (1) the indoor positioning algorithm, (2) the intersection detection algorithm deployed by user B during tag retrieval, and (3) the image similarity recognition algorithm.

3.1 Indoor Positioning

Whenever a user begins interacting with VirtualTag, he must first be localized within the room. VirtualTag's indoor positioning system is similar to Jang's [6]. In Jang's work, the user determines his location within a coordinate system defined by a known reference point – a QR code. In VirtualTag, the reference point serves as a solid black square, which could more easily be identified as a unique object by a mobile device from side angles.

Ultimately, the purpose of the positioning system is for VirtualTag to determine how far away the user is located from the black square. To do so, the pixel size of the square in the mobile device's image view must be estimated – from this, the user's distance from the square can be computed with simple geometry. Figure 5 depicts the user's interaction with the black square.

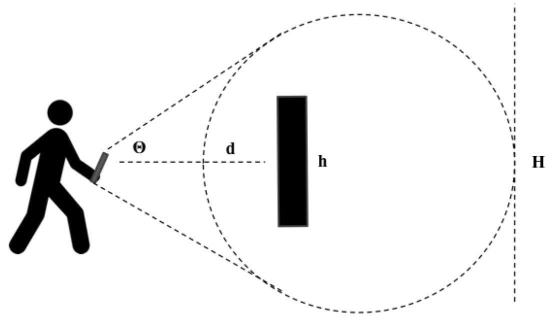

Figure 5: Indoor positioning diagram

We desire d, the user's physical distance from the square. Given that we know the physical height of the square (h), the mobile device camera's pixel resolution (r), and the viewing angle of the camera attached to the device (θ), we need to compute p, the pixel height of the square on the mobile device's display. From this, we can obtain the perceived physical height of the square (H) from the mobile device camera's perspective – namely, the physical height that the square would be if proportionally projected from the camera view onto the physical world. We calculate H as follows:

$$H = \frac{r}{p} * h \tag{1}$$

Then, given H, d is as follows:

$$d = \frac{H}{2 * \tan(\theta)} \tag{2}$$

Let us assume that the user is scanning the square marker from an angle α. We can define the square's 2-dimensional coordinates as $(x = 0, y = 0)$. The coordinates of a user's location are thus:

$$x = \sin\alpha * d \tag{3}$$
$$y = \cos\alpha * d \tag{4}$$

To determine p, the user first takes a picture of the square. Using OpenCV, an open-source, cross-platform, and real-time computer vision library, a Canny edge detector is applied to the image to extract the four edges of the square [9]. The edge detector is comprised of the following four steps:

1. Apply Gaussian filter to the image to remove ambient noise.

2. Identify intensity gradient of image to determine which pixels belong to foreground and which to background.

3. Apply non-maximum suppression to remove pixels that are not part of an edge (*i.e.* any pixels that do not lie along a "thin" line).

4. Carry out a hysteresis process to identify edges among candidate edges. Reject pixels whose intensity gradient values fall below α and accept if above β. Pixels with gradients between α and β are accepted if neighboring pixels have gradients above β.

Once the square's edges have been obtained, a Shi-Tomasi corner detector [10] is implemented (again via OpenCV) to obtain coordinates of the four corner points of the square. The midpoints of both horizontal sides are calculated from these corner points – from there, p is simply the distance between the midpoints.

3.2 Intersection Detection

After downloading all existing object tag metadata onto his mobile client device, user B scans the room searching for a potential intersection between his current line of sight and the line of sight seen by user A when tagging an object. VirtualTag models each user's line of sight as a line segment originating at the user's location and continuing 30 feet in the direction of the tag (which serves as an upper bound on the user's line of sight in an indoor setting). The intersection detection algorithm subsequently detects an intersection by examining whether the intersection point is contained within the two line segments' bounding intervals. Figure 6 depicts two possible intersection scenarios.

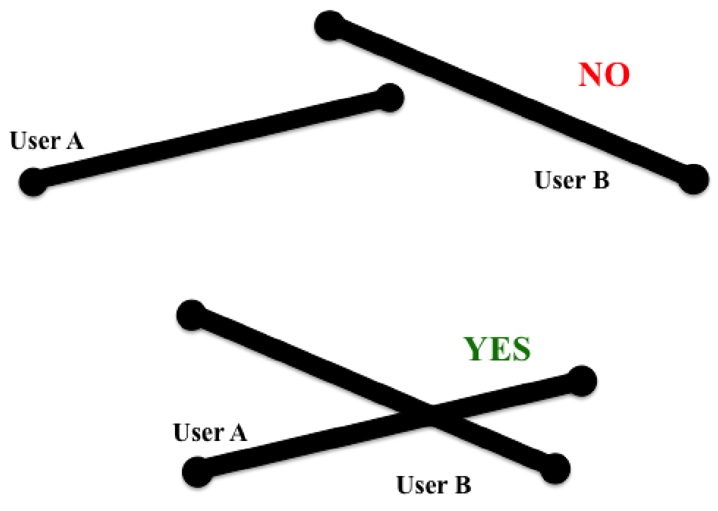

Figure 6: Two different intersection detection scenarios

The pseudocode for the intersection detection algorithm is as follows:

Algorithm 1 Intersection detection

```
1: procedure CHECKINTERSECTION
2:     $\theta_{tag} \leftarrow \theta_{tag} -$ tag initial offset
3:     $\theta_{cand} \leftarrow \theta_{cand} -$ cand initial offset
4:     $x_{tag2} \leftarrow -30 * \sin(\theta_{tag}) + x_{tag1}$
5:     $y_{tag2} \leftarrow -30 * \cos(\theta_{tag}) + y_{tag1}$
6:     $x_{cand2} \leftarrow -30 * \sin(\theta_{cand}) + x_{cand1}$
7:     $y_{cand2} \leftarrow -30 * \cos(\theta_{cand}) + y_{cand1}$
8:     intersect min $\leftarrow \max(\min(x_{tag1}, x_{tag2}), \min(x_{cand1}, x_{cand2}))$
9:     intersect max $\leftarrow \min(\max(x_{tag1}, x_{tag2}), \max(x_{cand1}, x_{cand2}))$
10:    if $\max(x_{tag1}, x_{tag2}) < \min(x_{cand1}, x_{cand2})$ then return false
11:    $A_{tag} \leftarrow \frac{y_{tag1} - y_{tag2}}{x_{tag1} - x_{tag2}}$
12:    $A_{cand} \leftarrow \frac{y_{cand1} - y_{cand2}}{x_{cand1} - x_{cand2}}$
13:    if $A_{tag} == A_{cand}$ then return false
14:    $b_{tag} \leftarrow y_{tag1} - A_{tag} * x_{tag1}$
15:    $b_{cand} \leftarrow y_{cand1} - A_{cand} * x_{cand1}$
16:    $x_{new} \leftarrow \frac{b_{cand} - b_{tag}}{A_{tag} - A_{cand}}$
17:    if $x_{new} <$ intersect min OR $x_{new} >$ intersect max then return false
18:    return true
```

3.3 Image Similarity Recognition

If the intersection detection algorithm returns true, VirtualTag has identified a candidate tag to be retrieved to user B. However, because of the infinite intersection problem, VirtualTag applies an image similarity recognition algorithm to the image being viewed by user B to ensure that it was indeed the image tagged by user A. The similarity recognition includes two steps: (1) feature extraction and (2) feature matching.

3.3.1 Feature Extraction

Feature extraction is applied twice during VirtualTag – first, when metadata from user A's object tag is sent to the server and secondly, when user B's mobile device has identified a candidate intersection. Using OpenCV, an oriented BRIEF (ORB) feature extractor is applied to raw image data to extract key components of the image. ORB utilizes the FAST keypoint detector and the BRIEF descriptor [10]. By identifying the image points most strongly associated with corners and intense colors, ORB both reduces the dimensionality of the image to allow for easier storage of the image features in the VirtualTag server. It also provides a quantitative means by which to compare the image to others.

3.3.2 Feature Matching

Feature matching is implemented via OpenCV to quantify the similarity between two different sets of image features. VirtualTag implements a Brute-Force Hamming matcher [11] to identify the closest feature point from feature set A to each feature point in feature set B (along with a Hamming distance of how similar the two feature points are). The output of the feature matcher is then assessed to determine how similar the two images are. If this assessment yields a match percentage that exceeds a given threshold (set as 0.5 during testing), VirtualTag identifies the image as a match, and user A's content is displayed to user B.

The pseudocode for processing the output of the feature matcher is as follows:

Algorithm 2 Feature Matching

```
   procedure CHECKIFMATCH
2:     threshold ← 0.5
       for matchPoint in matchSet do
4:         min ← min(matchSet)
           max ← max(matchSet)
6:     count = 0
       for matchPoint in matchSet do
8:         if matchPoint.HammingDistance < $3 * min$ then count++
       matchPercent ← $\frac{\text{size(matchSet)}-\text{count}}{\text{size(MatchSet)}}$
10:    if matchPercent > threshold then displayContent()
```

4 Implementation and Results

VirtualTag was implemented on both Android and iOS platforms to assess its deployability for consumer-grade applications and to determine the sensitivity of the aforementioned algorithms to different sensors. I will here describe key details of the implementations and additionally describe how well they performed.

4.1 Server Implementation

In order to persistently store object tag information, a Python server was implemented along with a lightweight MongoDB datastore. The server was hosted on Heroku for quick deployability and steady uptime. Each object tag corresponded to a unique record in the MongoDB datastore. The schema for the datastore is as follows:

Name	*Type*	*Description*
marker.id	integer	unique ID for each black square
init.direction	float (degrees)	direction facing when scanning square
direction	float (degrees)	direction facing when tagging object
x_{coord}	float (feet)	x-coordinate relative to square
y_{coord}	float (feet)	y-coordinate relative to square
image.features	string	extracted features converted to string
message	string	user-generated message

Table 1: Database schema for object tags

Whenever user A submits a tag to VirtualTag, a new entry gets populated in the VirtualTag database according to the schema above. Init.direction and direction are determined by the compass sensor on the user's mobile device. Marker.id simply captures the unique ID associated with each black square – this way, the server can return only those object tags associated with a given black square when user B begins the retrieval process. x_{coord} and y_{coord} are calculated according to the indoor positioning system, and image.features is a string encoding of extracted features. Finally, message is the text annotation provided by user A when submitting a tag.

4.2 Application Details

4.2.1 Android

VirtualTag was built and tested on an HTC One S device running Android version 4.0 (Ice Cream Sandwich). The HTC One S contains a digital compass and a camera with a viewing angle (θ) of 35.4°.

4.2.2 iOS

The application was also built and tested on an iPhone 4S running iOS version 7.0. The iPhone 4S contains a digital compass (accessible via its magnetometer) along with a camera with viewing angle of 43.2°.

4.2.3 Compass Noise Reduction

Raw measurements from the compass sensor yielded noisy values, especially on the Android platform. Figure 7 displays raw values of the compass measurements on an HTC One S device.

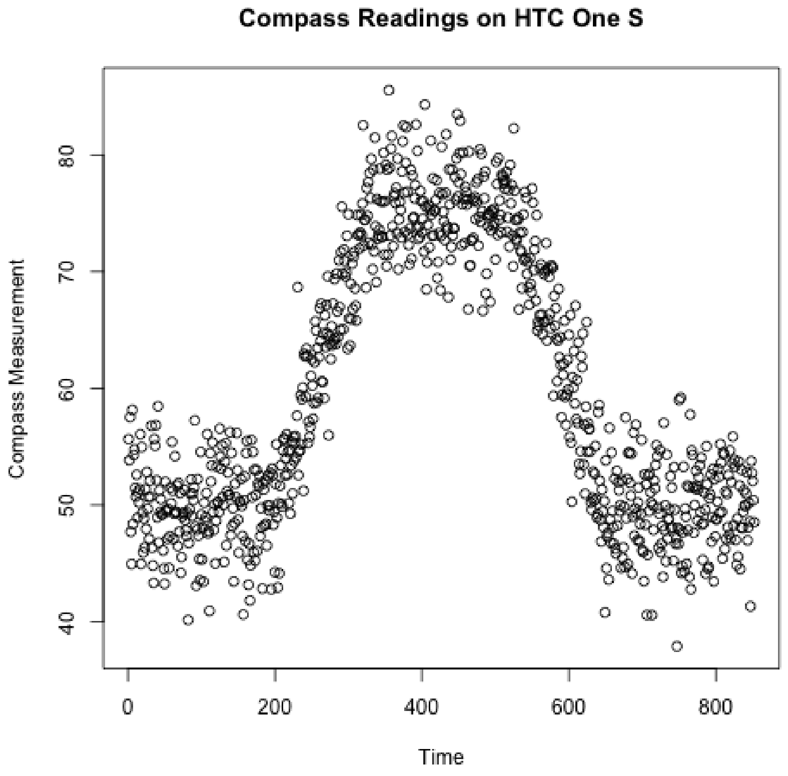

Figure 7: Raw compass measurements

In the above figure, the device was placed at rest at 50°, after which it was rotated gradually towards 70° before being rotated back to 50° (these degree measurements were externally verified by a calibrated compass). Even at rest, the measured compass readings appear to be influenced by Gaussian noise.

To obtain measurements that could more properly be utilized in VirtualTag algorithms, the raw compass readings were filtered using a moving average filter. Due to the high frequency with which sensor readings were sampled, a 50 point moving average filter was applied. Figure 8 displays the results of the moving average filter; the compass readings could now more reliably be used to determine the orientation in which the mobile device was facing.

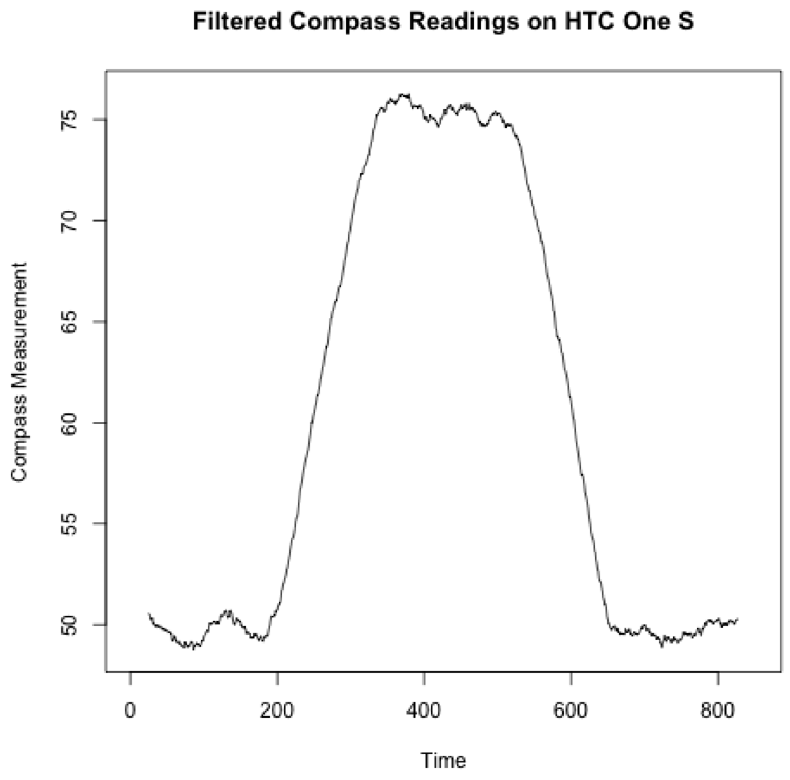

Figure 8: Filtered compass measurements

4.3 Application Evaluation

The two VirtualTag implementations were assessed in multiple environmental conditions to determine the robustness of the system design and to identify significant differences between platforms. Two experiments were performed to pinpoint precise strengths and weaknesses of the design.

4.3.1 Marker Distance

The first experiment assessed the impact of the black square marker's distance from users A and B on the likelihood of tag retrieval. This experiment was conducted in a well-lit setting with minimal visual noise, except for the black square on the room's wall and a nearby chair. User A would stand x feet away from the square, obtain his indoor location, rotate in place until facing the chair, and subsequently tag the chair with a message. User B would then stand $x + 1$ feet away from the square, obtain his indoor location, and rotate in place attempting to retrieve user A's message only by facing the chair. If the tag message would display on user B's mobile device when facing the chair, the retrieval was deemed a success. Figure 9 includes the results of this experiment.

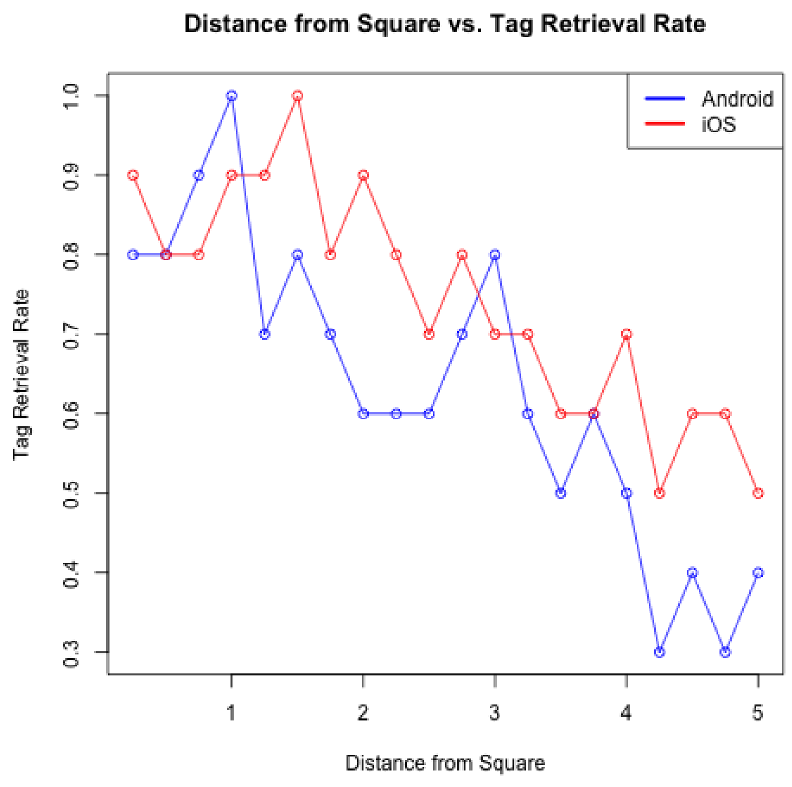

Figure 9: Effect of distance from square (ft) on tag retrieval rate

For each distance, 10 trials of the above experiment were performed. The retrieval rate was calculated as the ratio between the number of successfully retrieved tags and the total number of trials conducted.

Two key trends can be noted from these experimental results. The tag retrieval rate drops precipitously when users stand beyond 3 feet away from the black square. Initially, this result seems surprising. Given that VirtualTag has accurately localized the user, it would seem as if the system's effectiveness is independent of the user's location. However, increasing the user's distance from the square also increases his distance from the object being tagged. As a result, the image similarity recognition algorithm is more prone to incorporating features from the image background, given that more of the background will be visible. For nearby tags, however, VirtualTag performs quite well.

Additionally, the iOS application performed better than the Android one, particularly when the effect of distance from square became significant (*i.e.* after 1.5 feet). This can likely be attributed to the superior performance of the iPhone 4S hardware – the device's camera has significantly higher resolution, thereby producing a higher dimensional image feature vector. Additionally, the magnetometer used to generate compass measurements for the iPhone was much less noisy than than that of the HTC One S, thereby producing higher fidelity directional data.

4.3.2 Lighting Conditions

The second experiment assessed the importance of ambient lighting conditions on the likelihood of tag retrieval. VirtualTag's heavy reliance on computer vision after identifying a candidate tag makes it susceptible to changes in lighting. In order determine this, the following process was applied to three different lighting scenarios (again, in a room with minimal visual noise): (1) user A stands 2 feet away from the square and tags the aforementioned nearby chair and (2) user B stands 3 feet from the square and attempts to retrieve user A's tag. The three lighting conditions included natural light at 11:00 A.M., natural light at 4:00 P.M., and artificial indoor light at 9:00 P.M. Figure 10 depicts the results of this experiment.

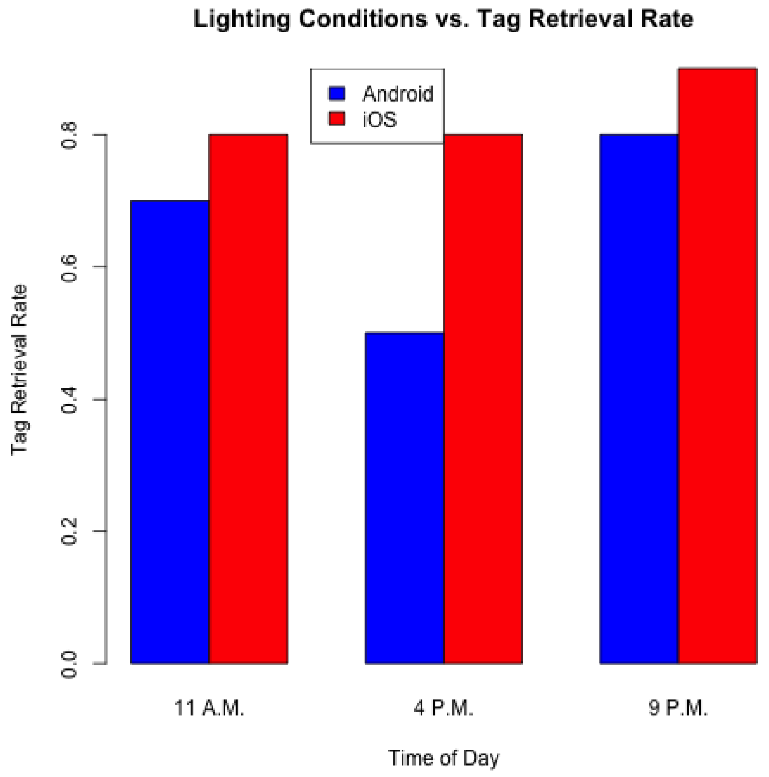

Figure 10: Effect of room lighting conditions on tag retrieval rate

For each lighting condition (and for each platform), 10 trials of the above experiment were performed. The reported tag retrieval rate was calculated in the same manner as for the marker distance experiment.

Unsurprisingly, the 11:00 A.M. and 9:00 P.M. lighting settings do not yield very different tag retrieval rates. Artificial light and bright sunlight illuminate indoor objects similarly, insofar as OpenCV-based image similarity recognition algorithms are concerned. However, for the relatively low-light environment at 4:00 P.M., the HTC One S device performed poorly. This is likely due to the lower resolution camera of the HTC One S compared to the iPhone 4S, which remains robust to a wider range of environmental light conditions. The feature detection and matching steps did not perform adequately in the 4:00 P.M. setting for the Android device. This suggests that increasing the image contrast in low-light settings might be necessary prior to applying feature-based analysis.

5 Limitations and Future Work

5.1 Geometric Improvements

The existing geometric model of tags – namely, 1-dimensional line segments originating at the user's indoor location – works well but makes the simplifying assumption that the user does not change the altitude of the mobile device after getting his coordinates. In reality, a user will not hold the device at precisely the same height when tagging objects. Further, a user may be in an enclosed space containing multiple floors. Here, the user clearly may change altitudes. This issue would be addressed by incorporating a third dimension into the coordinate system used to localize the user. This coordinate could be obtained either by using an altimeter (that is embedded natively in certain mobile devices) or by trivial extension of the indoor positioning system to three dimensions.

Alternatively, the representation of the tag itself can be extended to two dimensions. Rather than merely modeling a 30 foot line segment, VirtualTag can model each tag geometrically as a 30 foot long cone, with the cone's vertex angle being equivalent to the viewing angle of the mobile device's camera. Figure 11 depicts how intersections would be detected with this tag model.

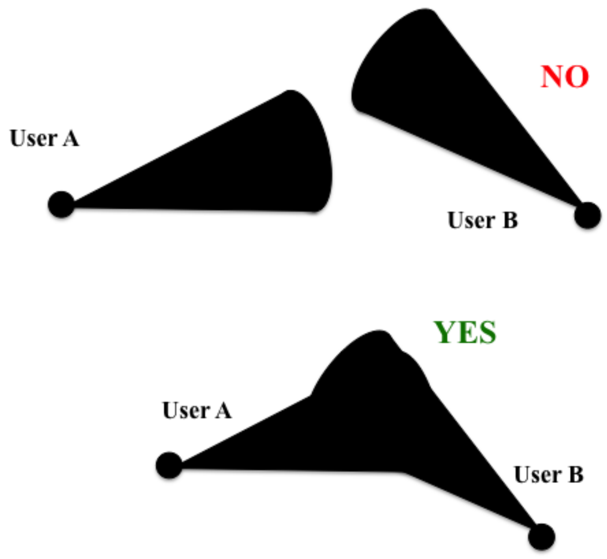

Figure 11: Two intersection detection scenarios with conical tag models

5.2 Computer Vision Improvements

Currently, much of the image similarity recognition utilizes existing OpenCV methods with minor modifications for performing feature matching. However, as evidenced by the drop-off of the tag retrieval rate past a short distance, there exists much opportunity to visually filter the image of the tagged object prior to performing feature extraction. Reddy, *et. al* describe the development and application of a background filtering algorithm on a single static image by consulting a training set of known indoor background environments [12]. Applying this filtration step prior to extracting the tagged object's features may eliminate background noise that yields false positives during the image retrieval step. The key, however, will be to reduce computational overhead during this filtration step to be able to perform it in real-time during user B's scanning sequence.

Another existing issue is the large size of the feature vectors themselves. Although the feature extraction step is performed locally on the client devices, the full feature vectors are transmitted to the VirtualTag server. Thus, there exists a large bottleneck to send and retrieve tag metadata for both users A and B. In order for the VirtualTag system to scale to more users, the feature vectors must be compressed prior to being uploaded to the server.

5.3 User Experience Improvements

VirtualTag expects the user to remain stationary after obtaining his coordinates via the indoor positioning system. While this is a feasible constraint given the emphasis of this project on determining the details of the tagging system, this poses quite a restriction on the user's ability to freely tag objects inside of a room. One area of future work is to incorporate the accelerometer sensor to track user motion within the room. This process, known as dead-reckoning, entails double integration of the user's acceleration data, given knowledge of the user's initial coordinates and assuming 0 initial velocity. It can be used to accurately predict a user's location within a confined space [13].

Additionally, much of VirtualTag testing occurred in simulated physical environments with reduced background noise. In real deployments, VirtualTag should be robust to various objects that might be scattered around the environment. This issue would partially be mitigated by improvements to the computer vision aspect of VirtualTag, although more experimentation could be performed with the existing design in these natural environments.

6 Conclusion

Constructing a robust indoor augmented reality platform continues to be a difficult but fruitful research challenge given the countless consumer applications that can be built on top of it. I have proposed VirtualTag, a new system that integrates prior work using geometric and computer vision-based approaches. VirtualTag leverages the benefits of both approaches – the computational advantage afforded by a purely geometric approach and the greater degree of precision provided by a vision-based strategy – while compensating for their disadvantages. I programmed VirtualTag applications on both Android and iOS devices and demonstrated their efficacy in a variety of environmental lighting conditions and user positions within a room. Although there remains much work to be done to deploy VirtualTag at scale, I hope that this work serves as a proof-of-concept that the ideal solution to the indoor AR problem integrates both geometric and computer vision-based strategies.

7 Acknowledgements

This project relied heavily on the contributions of numerous people. I would like to first thank both of my advisors. I thank Dr. Romit Roy Choudhury for encouraging me to undertake this project in Winter 2012, helping me formulate the problem statement precisely, and brainstorming different system designs for VirtualTag with me during his last semester here. I also thank Dr. John Board who stepped in as my advisor this past year and provided consistent and valuable direction and guidance during the implementation phase of the project. I would also like to thank Animesh Srivastava, Andy Ni, Alex Mariakakis, and other members of the Systems Networking Research Group (SyNRG) for their advice. Finally, I would like to thank Associate Dean Martha Absher and the Grand Challenge Scholars Program for funding my research efforts.

8 References

[1] Piekarski, W., & Thomas, B. (2002). ARQuake: the outdoor augmented reality gaming system. Communications of the ACM, 45(1), 36-38.

[2] Thomas, B., Close, B., Donoghue, J., Squires, J., Bondi, P. D., & Piekarski, W. (2002). First Person Indoor/Outdoor Augmented Reality Application: ARQuake. Personal and Ubiquitous Computing, 6(1), 75-86.

[3] Azuma, R. (1999). A motion-stabilized outdoor augmented reality system. Virtual Reality, 1999. Proceedings., IEEE, 32, 252-259.

[4] Google Goggles. (n.d.). Google. Retrieved April 16, 2014, from https://support.google.com/websearch/topic/25275?hl=en&ref_topic=1733205.

[5] GPS Accuracy. (n.d.). GPS.gov. Retrieved April 16, 2014, from http://www.gps.gov/systems/gps/performance/accuracy/.

[6] Jang, S. H. (2012). A QR Code-based Indoor Navigation System Using Augmented Reality. GIScience, 7, 5-17.

[7] Manweiler, J., Jain, P., & Roy Choudhury, R. (2012). Satellites in our pockets: An object positioning system using smartphones. In Proceedings of the 10th international conference on Mobile systems, applications, and services. ACM.

[8] Tenmoku, R., Kanbara, M., & Yokoya, N. (2003). A Wearable Augmented Reality System Using Positioning Infrastructures and a Pedometer. Seventh IEEE International Symposium on Wearable Computers.

[9] Canny, J., A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679698, 1986.

[10] ORB (Oriented FAST and Rotated BRIEF). (n.d.). ORB (Oriented FAST and Rotated BRIEF) OpenCV 3.0.0-dev documentation. Retrieved April 16, 2014, from
http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_orb/py_orb.html.

[11] Common Interfaces of Descriptor Matchers. (n.d.). Common Interfaces of Descriptor Matchers OpenCV 2.4.8.0 documentation. Retrieved April 16, 2014, from http://docs.opencv.org/modules/features2d/doc/common_interfaces_of
_descriptor_matchers.html.

[12] Reddy, V., Sanderson, C., & Lovell, B. C. (2011). A Low-Complexity Algorithm for Static Background Estimation from Cluttered Image Sequences in Surveillance Contexts. EURASIP Journal on Image and Video Processing, 2011, 1-14.

[13] Steinhoff, U. (2010). Dead reckoning from the pocket - An experimental study. In 2010 IEEE International Conference on Pervasive Computing and Communications. IEEE.