ImageBeacon: Broadcasting Color Images over Connectionless Bluetooth LE Packets

Chong Shao
Department of Computer Science
Univerisity of North Carolina at Chapel Hill
Chapel Hill, NC 27599
Email: cshao@cs.unc.edu

Shahriar Nirjon
Department of Computer Science
Univerisity of North Carolina at Chapel Hill
Chapel Hill, NC 27599
Email: nirjon@cs.unc.edu

***Abstract*—This paper describes the first 'image beacon' system that is capable of broadcasting color images over a very long period (years, as opposed to days or weeks) using a set of cheap, low-power, memory-constrained Bluetooth Low Energy (BLE) beacon devices. We design an image processing pipeline that takes into account the background and foreground information of an image and then applies an adaptive encoding method which priorities more important regions of an image during encoding, in order to achieve the best quality image under a very strict size limit. We test our system with different types of RGB images that contain indoor and outdoor objects, buildings, and road signs. We empirically determine the tradeoffs between the system lifetime and the quality of broadcasted images, and determine an optimal set of parameters for our system, under user-specified constraints such as the number of available beacon devices, maximum latency, and life expectancy. We develop a smartphone application that takes an image and user-requirements as inputs, shows previews of different quality output images, writes the encoded image into a set of beacons, and reads the broadcasted image back. Our evaluation shows that a set of 2–3 beacons is capable of broadcasting high-quality images (70% structurally similar to original images) for a year-long continuous broadcasting, and both the lifetime and the image quality improve when more beacons are used.**

I. Introduction

In this modern age of the Internet of Things (IoT), it is now possible to literally glue tiny computers to everyday objects, so that they can sense, react, and tell their own stories. The IoT community has embraced wireless standards such as Bluetooth Low Energy (BLE) and developed programmable 'beacon' devices that periodically broadcast a small amount of preloaded data, while lasting for multiple years on a coin-cell battery. Broadcast messages from beacon devices typically contain information about an object, a location, a web-resource, or just an arbitrary string. This connectionless mode of BLE does not require a receiver to pair/bond or connect to a sender, and hence, there is no overhead of connection setup and no inconvenience of requiring a user to enter pins and passwords. These broadcast messages are received by a BLE capable mobile device to obtain relevant information just-in-time and on-the-spot. Emerging applications of beacon devices include advertising merchandise in retail stores, identifying late passengers at the airports, authorizing people at the hospitals, smarter signage, indoor navigation, and tracking moving platforms like airline cargo containers, computers on wheels, museum artworks, or even humans.

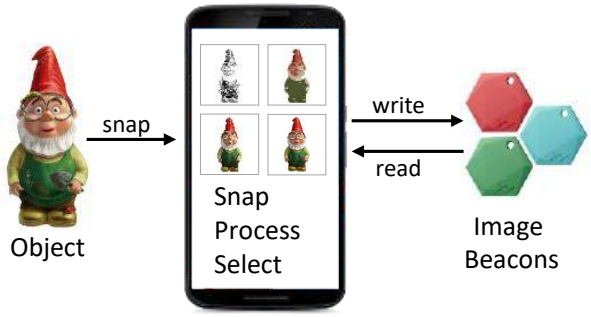


Fig. 1. An image beacon system.

The enabling technology behind these applications is the ability of a beacon to simply broadcast a few bytes of data (called UUID) as BLE 4.0 advertisement packets at a rate of less than 16 bytes/sec. The bound in data rate comes from the lifetime requirement of these devices. Such a tight budget on payload size and the maximum data rate have limited a beacon's capability to only be able to broadcast an identifier or a small amount of text (effectively $\sim$18 bytes). The next generation BLE 5.0 beacon is expected to have an 8X increase in broadcasting capacity ($\sim$256 bytes). Such an increase opens up the possibility to design beacons that can serve larger assets, e.g., an image, carried by connectionless BLE advertisement packets. However, even a simple 72×72 PNG image, such as the Android launcher icon, has a size of over 3KB. To store and broadcast this image, either we require to use a dozen of BLE 5.0 beacons, or we will have to accept a very long image transmission and loading time.

Image compression is a natural way to deal with this problem. Existing image compression algorithms, however, fail to achieve the desired compression ratio for an image to be broadcasted over BLE. Hence, a fundamental challenge toward realizing an image beacon is to devise an algorithm that efficiently represents an image using as few bits as possible, while taking into account the application-driven limits on the number of usable beacons per image, broadcast message size, data rate, latency, and lifetime. In an earlier work [14], we devised an image beacon system that broadcasts binary images

of a few limited categories (e.g., handwritten characters) only. This paper is a continuation to that line of work, but this time, we have taken a harder challenge, i.e., to develop a beacon system that works for color images, e.g., images taken with a mobile phone.

Being able to broadcast images from beacons enables more powerful and feature rich applications than the ones supported by today's beacons. We envision that like the web has evolved from serving hypertexts to streaming multimedia contents, the natural successor of today's beacon devices would be the ones that broadcast images. Applications of image beacons would be in scenarios where there is no Internet connectivity but there is a need for storing and broadcasting information that can be best described by an image. For example, coordinating rescue workers in disaster areas, creating a bread-crumb system for adventurous hikers and mountaineers, remote surveillance (when coupled with a camera), or even a simple system just to let someone know that 'We were here'. Recently, Google started to experiment with an idea called *'Fat Beacons'*, where they are looking into broadcasting html pages over BLE. However, for lack of a suitable image compression technique, the pages do not support images.

In this paper, we chase this seemingly impossible goal of creating an image beacon system that efficiently broadcasts color images, carried by BLE broadcast messages, over an extended period of time. We propose a self-contained system that stores and broadcasts actual image contents as opposed to IDs, links, or URLs of an image. We assume availability of no additional information on the broadcasted image from any other sources – globally (on the web) or locally (on a user's smartphone that receives the broadcast).

The crux of the system is an algorithm that analyzes an image to identify its 'important' semantic regions (as defined by the user or the use case) and then encodes them differently than the rest of the image to reduce the overall image size. The image data are written to and read from the image beacon system using a smartphone application, which runs the proposed compression and rendering algorithms. We use the term 'beacon system' instead of 'a beacon', since a compressed image may still require more than one physical beacons to ensure its acceptable quality. Allowing multiple beacons per image makes the system flexible. It widens our scope for optimizations and helps satisfy users who are willing to dedicate more beacons for better results. Besides, until BLE 5.0 is available, we need to simulate its broadcast capacity with multiple BLE 4.0 devices anyways.

We have developed a prototype of an image beacon system using a set of commercially available Estimote beacons [2], and developed an Android application that takes images of an object of interest along with user-specified requirements and constraints on broadcasting the image as inputs, generates previews of the image to be written, writes the image representation into a set of beacons, and reads the broadcasted image back. Figure 1 shows an example scenario where a user snaps photos of a gnome statue which he is interested in broacasting. The smartphone application performs image processing on the phone to produce multiple versions of broadcast image. The user selects one of these compressed images that satisfies his requirements (e.g. available beacons, image quality, lifetime, and image loading latency). The user is allowed to change his requirements and the app immediately shows options for the best possible compressed images under those constraints. The application writes the image data into the beacon system and the image is broadcasted by the beacons. A reader application reads the broadcasted image and displays it on the phone.

We perform an in-depth evaluation of the beacon system. We describe a set of results showing the tradeoffs between system lifetime and image quality, when the image type and the number of beacons are varied. We also deploy an image beacon system indoors, and perform a user study in a real-world scenario in order to have a subjective measure of the quality of the received images, where a group of 20 participants are asked to identify objects from their beaconed images of various resolutions, and locate it among a set of similar looking objects in the real-world.

The main contributions of this paper are as follows:

- To the best of our knowledge, we are the first to propose an image beacon system that uses multiple BLE beacons to broadcast color images over the BLE advertisement messages.
- We have devised an image approximation algorithm that is tailored to the need of an image beacon system. We quantify the tradeoffs between the image quality and the device lifetime, and determine the best set of parameters, under the user-specified constraints on the number of beacons, latency, and expected system lifetime.
- We have developed and evaluated a prototype of an image beacon system that broadcasts color images of various types (e.g., near-distance indoor and outdoor objects, road signs, and buildings). Our evaluation shows that one BLE 5.0 beacon would be capable of broadcasting good-quality images (70% structurally similar to original images) for a year-long continuous broadcasting, and both the lifetime and the image quality improve when more beacons are used.

II. Problem Formulation

A. Generic Problem Setting

The problem is formally stated as: given an image x (where each pixel is represented by b bit) having the dimensions of $\mathrm{N} \times \mathrm{M}$, the number of available beacon devices K, the payload size of each beacon packet C bytes, the maximum allowable broadcast rate of R packets/sec, and the maximum allowable latency for an image T, the objective is to find an approximate representation of the image $\hat{\mathrm{x}}$ so that the lifetime τ of the beacon system is maximized while the approximation ratio $\lambda(\mathrm{x}, \hat{\mathrm{x}}) \in [0, 1]$ of the image is high ($\lambda = 1$ means no distortion). Now, for a single beacon, the broadcast rate:

$$\mathrm{R} = \left(\frac{\mathrm{bNM}}{8\mathrm{C}}\right)\frac{1}{\mathrm{T}} \tag{1}$$

For K beacons, considering $\log K$ overhead bits for addressing the beacons, and K times more payload capacity:

$$\mathrm{R} = \left(\frac{\mathrm{bNM} + \log \mathrm{K}}{8\mathrm{CK}}\right)\frac{1}{\mathrm{T}} \tag{2}$$

Both (1) and (2) are for undistorted images.

The lifetime τ of a BLE device depends on its inter packet interval and in general, $\tau \propto \frac{1}{\mathrm{R}}$. Replacing R and incorporating approximation ratio λ into (2):

$$\frac{1}{\tau} \propto \left(\frac{\lambda \mathrm{bNM} + \log \mathrm{K}}{8\mathrm{CK}}\right)\frac{1}{\mathrm{T}} \tag{3}$$

The above equation relates the lifetime of an image beacon system and the approximation ratio of any image compression algorithm. Ideally, we look for an image approximation algorithm that achieves a sufficiently large λ for a reasonably high lifetime of the system.

B. Broadcast Capacity of Bluetooth LE

According to the BLE 4.0 specification, the maximum payload size C available in beacons is 18 bytes. However, there are 33 reserved characters that cannot be read from the beacon devices. So, practically the payload size is $\lfloor \log(256 - 33)^{17} \rfloor \approx 132$ bits ≈ 16 bytes.

The expected lifetime of BLE beacons depends on the inter-packet interval [5]. For example, a BLE 4.0 beacon would last up to 3.5 years, if a packet is sent at every second (i.e. R = 1). Therefore, for a beacon system to last for 3.5 years, its broadcast bandwidth cannot exceed 16 bytes/sec.

Recently, BLE 5.0 has been announced [1] to offer an 8X increase in broadcast capacity and a 2X increase in transmission speed. It is scheduled to be released in early 2017. In coming days when BLE 5.0 capable devices will be widespread, we expect to have a 128 byte sized payload and about 256 bytes/sec broadcast bandwidth.

C. The Case for Loss-Less Image Broadcast

The size of a typical 72×72 PNG image can be anywhere between $3 - 13$ KB. Therefore, to transmit such an image, a BLE 4.0 beacon would require $191 - 832$ broadcast packets, or alternatively, we would require up to $K = 832$ beacons to simultaneously broadcast different slices of an image. The latency of a complete image transmission cycle would be up to $\mathrm{T} = 13.9$ minutes for a single beacon, or 1 second for a set of 832 beacons.

When BLE 5.0 beacons will replace 4.0, the transmission latency will drop to 52 seconds for one beacon, or 1 second 52 of them. Therefore, without compressing the image content, even the new BLE 5.0 beacons will not be able to support a fast image beacon system with a reasonably small number of beacons.

D. The Case for Compressed Image Broadcast

If standard image compression algorithms could generate compressed images that meet the size and quality requirements of an image beacon system, the problem would have been already solved. But the fact is, even the best of existing image compression methods, such as JPEG/JPEG2000 and PNG, are not capable of optimizing for both quality and size at the same time. Figure 2 illustrates that JPEG/JPEG2000 generates extremely poor quality images given a size requirement of 300 bytes even for a very low-resolution (64×64 pixels) image. On the other hand, to have a compressed image of acceptable quality (having a minimal useful visual information to the viewer), JPEG/JPEG2000 takes about 2K bytes.

Fig. 2. A 64x64 resolution image compressed in high/low quality settings using JPEG/JPEG2000: (a) JPEG high quality, 1963 bytes (b) JPEG2000 high quality, 2026 bytes (c) JPEG lowest possible quality, 738 bytes (d) JPEG2000 lowest possible quality, 391 bytes.

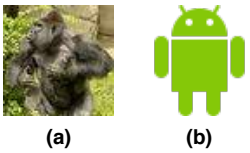
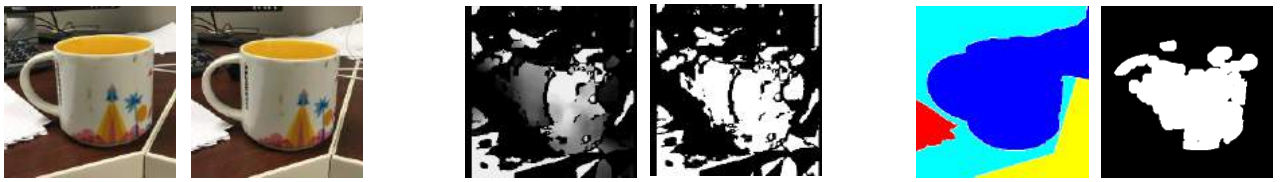


Fig. 3. Two types of 64x64 resolution image compressed in PNG (a) from natural scene, 12112 bytes (b) JPEG2000 high quality, 1012 bytes. PNG is good for handling images with large uniform color regions.

PNG and Vector Graphics image, on the other hand, have the potential to generate a smaller compressed image that *may* fit our constraints. However, these codecs generate smaller images only if the input image is of a specific type – such as an image containing a few regions of uniform colors like a cartoon drawing, or when the shape is not complicated. This is illustrated in Figure 3. In general, PNG and Vector Graphics image encoding do not meet the requirements of an image beacon system that broadcasts color images taken by a smartphone user.

III. Image Beacon System Overview

In a typical usage scenario of the proposed image beacon system, a user at first takes pictures of an object with his smartphone's camera. A smartphone application analyzes the image (which may contain objects, portraits, scenes, shapes, signs, and/or text), identifies semantic regions on it, and processes each region differently to produce a compressed version that satisfies the beacon system's requirements such as the number of available beacon devices, maximum allowable loading time, and lifetime. The user is also shown an interactive preview of the image so that he can verify it, as well as relax/constrain the system requirements. Finally, when he is satisfied with the preview, the image is written into the image beacon system. The beacon system would then broadcast the image periodically over BLE, and any other smartphone user would



(a) Multiple View Capture (b) Depth Estimation (c) Depth-Refined Segmentation

Fig. 4. Multiple views of a scene are used to estimate the depth map. Combined with standard image segmentation, this can identify the pixels of an image that may be of more interest than the rest, e.g. a foreground object.

be able to receive that broadcast and see the image on their phones.

A. System Design Choices

The design choices we made in developing an image processing algorithm for the proposed image beacon system are as follows:

- First, our custom image compression technique is designed to work for images taken with a smartphone. We assume that the phone has an on-board IMU in it. The final compressed image will be a color image with a lower resolution, such as 64×64 pixels.
- Second, we make a reasonable assumption that the image to be compressed is linked to a real-word "thing" like a near-distance object, a road sign, or a building – which has one or more regions of interest that a person who took the picture wants to preserve with a higher priority than the rest. By exploiting this, we design an image encoding technique that prioritizes foreground information preservation during image compression, so that the most important information in the image is delivered under a given size constraint.
- Third, under a very tight budget for the final image size, any image compression algorithm would distort the original image – which is reflected in different ways such as lacking boundary details, increased noise, changed colors, or removal of texture. We introduce the concept of *adaptive encoding* that applies different encodings to different regions of an image based on the image content (e.g., a road sign vs. a t-shirt), image regions (foreground vs. background), and what a user would prefer to preserve (e.g., texture or true color). The proposed compression algorithm should employ an adaptive approach that applies the most suitable encoding technique for different image types and region types, so that an optimal compression strategy is chosen for a given image based on its content.
- Fourth, since both capturing an image and writing the compressed version into the beacon devices involve the smartphone user in the loop, we provide an interactive user interface in order to guide the user in taking pictures, and to preview and select the desired image under a given set of system constraints.

B. Image Processing Pipeline Overview

For a given set of user-defined beacon system requirements, the overall image processing and compression pipeline (Figure 5) consists of four basic stages: multiple view capture, depth estimation, depth-refined segmentation, and image compression. These steps are briefly described in this section, and elaborated in detail in the subsequent sections.

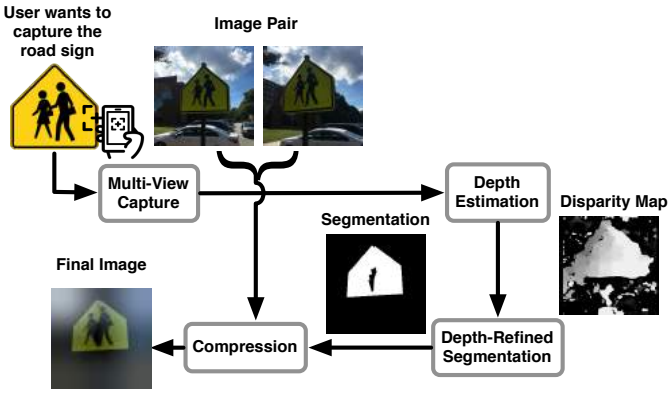


Fig. 5. Image processing stages.

- *Multiple View Capture:* The proposed system requires a user to capture two or more views of an object – which helps at a later stage when the depth map is generated. Estimating pixel depths using a pair of images takes about 2 seconds on a mobile device. Because processing too many images would be time consuming, a careful selection of views (e.g. images having adequate overlaps) makes a difference. Figure 4(a) shows two views of a mug that have enough overlap to create a depth map. To guide the user and to select the best pair of images for depth estimation, we leverage IMUs of the smartphone. The algorithm is described in Section IV.
- *Depth Estimation:* Depth of each pixel is estimated by finding and matching corresponding 'feature points' (e.g., corners and edges) in two or more images. The matched points are then used to generate the camera relative geometry, so that the depth of every pixel can be estimated. Figure 4(b) shows two depth maps of the same image. The left one is the computed depth map, and the right one is thresholded to separate the background from the foreground pixels. However, due lack of enough views, low resolution, and inaccuracies in estimation, depth map alone

is not sufficient to segment semantic regions in an image. Depth estimation from multiple views is discussed as part of Section V-A

- *Depth-Refined Segmentation:* Like depth map, color/texture-based image segmentation algorithms often fail to identify semantically different/similar regions in an image. For example, the left image in Figure 4(c) is the result of applying marker-controlled watershed segmentation [12] on the original image. When we overlay this with the depth map, we obtain a better segmentation, which performs a much better job in isolating the mug from the rest. This step is inspired by one of our earlier work [10] that used RGB and depth images from Kinect sensors. In this work, we use only images to estimate the depth (previous step) and then apply this step to get the final segmentation. The details of this step are in Section V-B.
- *Image Compression:* The image compression stage takes both an image (for texture and content information) and its segmentation map (for semantic region information), and produces the best quality image under the user-specific constraints of the beacon system. Until the resultant image size does not satisfy the system requirements, the algorithm gracefully degrades the quality of different semantic regions, starting from the least important one (e.g., the background). This step is described in detail in Section VI.

IV. Multiple View Capture

A. Need for Multiple Views

The first step of our proposed image processing pipeline is to guide the user in capturing two or more views of an object of interest. Further down the pipeline, these images are used to estimate the depth information of each pixel, so that an image can be segmented into background and foreground regions, prior to applying appropriate region-specific encodings.

An alternative to using multiple views is to apply standard image segmentation algorithms [12], [11], [7], [19] on a single image. These algorithms groups adjacent pixels of an image based on information derived from pixel intensity in various ways. However, in order to obtain a sufficiently accurate segmentation for the proposed system, we require computationally expensive algorithms, such as convolutional/recurrent neural networks [19], which are not suitable for running on smartphones and does not produce results in real-time.

B. Challenges with Multiple Views

Even though smartphones cannot run state-of-art image segmentation algorithms in real time, many other computer vision techniques, including depth estimation from two views, can be implemented on them. In order to get a sense of their real-time performance, we used OpenCV library for Android to compute the depth map for a pair of 400×400 pixel images. It took about two seconds for the algorithm to finish on a Nexus 5 phone. This gives us the lower limit for depth map computation, which may only happen if the user is well-trained and knowledgeable to know which views or camera poses would produce the most effective depth map.

A good pair of images is critical in generating a good depth map. However, the finding of a good pair of images depends on many factors. The most important of which is a suitable difference in view angles. It also depends on the distance between the object/scene and the smartphone. Furthermore, there are other factors such as lighting, texture and shapes of the image.

If a real-time depth estimation system could display the current depth map as the user takes images, it would be easier for him to generate a good pair of image for depth estimation. However, depth estimation does not run in true real-time on most smartphones. In absence of a real-time feedback, a user has to take the trial-and-error approach, i.e., he has to take two images, look at the result after two seconds, and then repeat the entire process until the result looks good. This may lead to a very long time in just taking the right photos, and result in a non-smooth user experience.

C. IMU Assisted View Capture

To address this problem, we designed a method to make use of the inertial measurement unit (IMU) of the phone to shorten the image capture time and to improve the user experience. We adopt a machine-learning based approach.

In the offline training phase, we use a smartphone to capture video/image of multiple indoor and outdoor objects. We also keep record of the IMU values for each captured image. The IMU data consists of $\delta \mathrm{x_r}$, $\delta \mathrm{y_r}$ and $\delta \mathrm{z_r}$, which represent the components of the difference vector between the rotation vectors between a pair of images. The IMU data also contains $\mathrm{x_a}$, $\mathrm{y_a}$ and $\mathrm{z_a}$ components of acceleration when taking an image. After this, we run depth estimation for all pairs of images and estimate its accuracy by comparing the result against a manually generated ground-truth segmentation.

The segmentation accuracy is measured in terms of *intersection over union* (IoU), where *intersection* is defined as the area of intersections between foreground regions of two segmentations, and *union* is defined as the set of pixels either marked as foreground in the testing segmentation or in the ground truth segmentation. For a segmentation that is identical to the ground truth, IoU equals to 1.

Using IoU values as the variable Y, and the IMU data for the corresponding pair of images as the variable X, where $\mathrm{X} = [\delta \mathrm{x_r}, \delta \mathrm{y_r}, \delta \mathrm{z_r}, \mathrm{x_a}, \mathrm{y_a}, \mathrm{z_a}]$, we create a data set for many pairs of images, and then train a regression tree model to learn the relationship between the change in IMU values between a pair of images and an expected quality of depth segmentation.

During the online phase, when the user is taking images for depth estimation, the trained regression tree model keeps track of current IMU readings and gives hints about if current view is a good choice, given the already taken photo(s).

V. Depth Estimation and Segmentation

The second and third stages of the proposed image processing pipeline are described together in this section.

A. Depth Estimation

In this step, the depth of each pixel is estimated from a pair of images. We use a standard algorithm [6] that at first estimates the 'disparity' between the corresponding points on two images, and then estimated depth from disparity. For example, if a point P_1 on the first image and a point P_2 on the second image correspond to the same point P on the actual physical world object, then $(\mathrm{P}_1 - \mathrm{P}_2)$ is called the disparity between them. Depth of a pixel is, in general, inversely proportional to its disparity. This is based on the principle that points in the scene that are closer to the camera will have larger disparity, and points that are very far away will be effectively at the same or very close location on both images. Hence, finding the depth maps is essentially equivalent to finding the disparity map.

The disparity map is generated by the *stereo matching* algorithm described in [6]. The goal of the algorithm is to find matching pixel blocks in a pair of images. This is also called the *correspondence problem* as the it looks for the pixel coordinates on the image pair that correspond to the same world point. The disparity map is computed based on the matching result. The disparity map is a gray-scale map, where the intensity directly corresponds to depth.

B. Depth-Refined Segmentation

The depth estimation algorithm groups pixels purely based on depth. It may not group pixel regions even if the regions share a common appearance pattern. As a result, even an accurate depth map tends to contain holes in foreground regions and isolated, incorrectly marked, small, bright regions in the background. Therefore, it is necessary to introduce other types of information derived from the image to obtain a cleaner and better segmentation. The refinement process is described as follows:

- *Thresholding:* At first, a binary segmentation of the depth map is obtained by applying a threshold on depth values.
- *Combining with Watershed Segmentation:* We combine the depth-based segmentation map with another image segmentation method which groups pixels into several connected large regions and is computationally inexpensive to run on a mobile device. Watershed segmentation algorithm fits these requirements. However, the traditional watershed segmentation tends to generate an over-segmented result. Hence, we adopt the marker-controlled watershed segmentation, which uses mathematical morphology operations to pre-process an input image to avoid over-segmentation [6].
  At first, the input image is converted into grayscale. Then we run the marker-controlled watershed segmentation on the grayscale image. A successful segmentation contains more than one segmented regions to separate foreground from background in the image. To determine which region belongs to the foreground, we apply a voting approach: the region that includes the highest number of common pixels with the foreground region in the depth-based segmentation map is labeled as foreground. Here we denote the number of common pixels as N, where N is defined as:

$$\mathrm{N} = \max_{\mathrm{i}} \ \mathrm{C}(\mathrm{W_i}, \mathrm{D}) \tag{4}$$

  Here, i is the index of a region, and $\mathrm{W_i}$ is the corresponding region. D represents the foreground region in the depth-based segmentation. $\mathrm{C}(,)$ computes the number of common pixels between two regions. If there is another region $\mathrm{W_j}$ for which, $\mathrm{C}(\mathrm{W_j}, \mathrm{D}) \geq 0.8 \times N$, then it is also considered as foreground. This procedure iterates until no more regions can be added to the foreground. Having two result segmentation maps, we produce a final map by labeling pixels that are considered foreground in both maps as foreground.
- *Final Refinement:* Finally, we perform a pixel-level refinement process. We remove all connected foreground pixel regions with size less than 1000 pixels because regions of this size tend to be a background region. Then we apply two common mathematical morphology operations erosion and dilation, to clean out any remaining bright pixel islands in the background and to expand the foreground regions, respectively. Lastly, we enforce the accuracy of the segmentation boundary by combining the result with the labeled watershed foreground region.

VI. Image Compression

The last stage of the proposed image processing pipeline is the image compression step. Using the segmentation information from the previous stage, this step encodes different segments of an image using different encoding techniques. The overall goal is to make sure that the resultant image fits the storage requirement of a beacon system, while making sure that the foreground regions are the least affected by during the compression process.

We propose three encoding options for image compression – *discrete cosine transform* (DCT) and coefficient reduction, *wavelet transform* with coefficient reduction, and *foreground texture triangularization*. All three are applied on the input image and finally the one that produces the best quality image is chosen as the output. The effect of applying different encodings is illustrated in Figure 7.

A. Discrete Cosine Transform Encoding

Discrete Cosine Transform (DCT) is a widely adopted image encoding technique. We integrate DCT encoding into our compression system as a baseline encoding option. The benefit of using DCT is that – by reducing low frequency coefficients of an image (in the DCT transformed space), the resultant compressed image's *appearance* details is removed first, while it *global shape* is preserved. This is useful in usage scenarios when a user wants to preserve the shape of an object in the image more than its detailed appearance.

At the beginning of encoding, we generate a foreground image by using the segmentation map to set the background pixels' intensity of the input image to zero. The image is then down sampled to 64×64 pixels. To further reduce the size, we

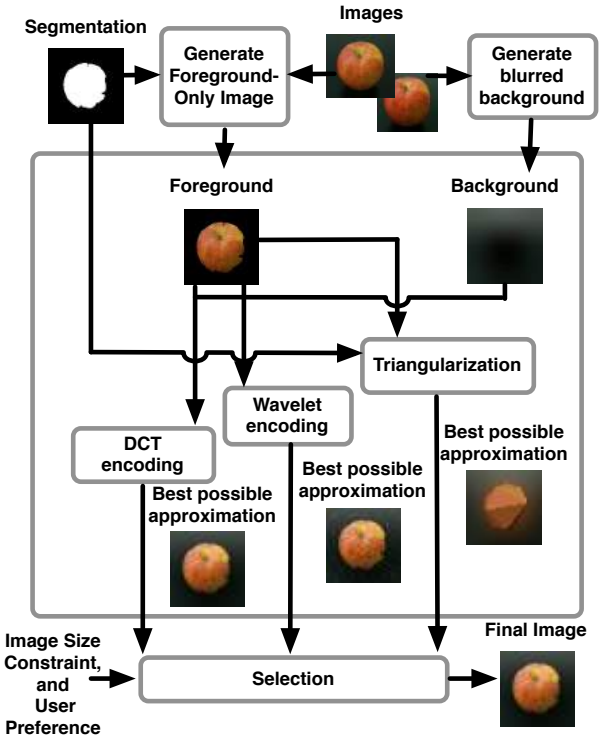


Fig. 6. Image compression details.

Fig. 7. 64x64 resolution building image compressed in high/low quality settings using our customized DCT/Wavelet/Triangle encoding: (a) Original image, (b) DCT 342 bytes, (c) Wavelet high 360 bytes, (d) DCT 1114 bytes, (e) Wavelet 1098 bytes, and (f) triangularization 366 bytes. For a similar compressed image size, DCT preserves less details than Wavelet method. But for low quality settings (about 350 bytes), Wavelet-encoded images have strange color block defects. Triangularization failed to preserves the information in the original image.

make use of the fact that the quality of an image depends more on its brightness information than its color information. During the encoding, at first, an image is transformed into the YUV space, where Y represents the brightness information and U/V represents the image color information. We further down sample U and V channels into 32×32 pixels, while keeping the resolution of Y channel intact. Then DCT is applied on all three channels, followed by a data-reduction step that sets high frequency components to zero. Finally, the resultant frequency coefficients are compressed using gzip. The data is sent to the beacon along with the blurred background image, which is also encoded using DCT. The size of a DCT compressed image, $\mathrm{S_{DCT}}$ can be expressed as:

$$\mathrm{S_{DCT}} = \mathrm{B(g(d)) + B(g(b))} \tag{5}$$

Where, B() denotes the bit length, g() denotes the gzip encoded data size, d is the DCT transformed (reduced coefficient version) foreground information, and b is the coefficient of the DCT transformed (blurry) background image data.

At the receiving end, a broadcast image is recovered by superimposing the foreground image and the background image. Note that, the background image needs to have the foreground pixel intensities set to zero, before it is down sampled. This is done to make sure that the foreground pixel intensities are not added up twice.

A limitation of this above approach is that, for an image with a uniform dark background and a foreground having more details, DCT may yield a ringing artifact close to the sharp edges, especially in a low quality setting. This problem can be addressed by switching to using wavelet.

B. Wavelet Encoding

Wavelet is the second image encoding method that we integrate in the adaptive image encoding process. When compared to DCT, wavelet tends to better handle images whose backgrounds have an uniform intensity. Similar to DCT, the best information reduction parameters are sent to the wavelet encoding module in order to generate the highest quality image under a given storage limit. We adopt global thresholding of the wavelet coefficients and Huffman encoding, based on the method described in [13].

Similar to DCT, prior to encoding the foreground image, its background pixel intensities are set to zero to obtain the wavelet data. The down sampled background image (with zero intensity foreground) is also sent along with the wavelet data. At the receiving end, wavelet coefficients are inverse-transformed to generate the foreground image and then superimposed on the background image to render the final image.

The size of a wavelet compressed image $\mathrm{S_W}$ is as follows:

$$\mathrm{S_W} = \mathrm{B(H(w)) + B(g(b))} \tag{6}$$

where, H() denotes the Huffman encoded data, and w denotes the reduced wavelet coefficients on the wavelet transformed foreground image. All other symbols carry the same meaning as discussed in the previous section.

The weakness of wavelet encoding is that, for limited storage requirements (< 500 bytes), an wavelet encoded image may have unrealistic texture patches after decoding.

C. Triangularization-Based encoding

Both DCT and wavelet encodings blend the geometric information and texture information of the foreground image. However, for some cases, an accurate texture information and a fine-grained boundary representation are not necessary. For example, a "football" image's foreground texture and shape could be decoupled. For a viewer to understand that the image is about a "football", a repeated patch of a football's surface texture along with an approximate "football" shape information would suffice. Both DCT and wavelet would encode too much redundant information for such an image. To address this, we designed a triangularization-based image

encoding method, which consists of three stages: triangularization, triangle reduction, and color/texture filling. These are illustrated in Figure 8, and are described as follows:

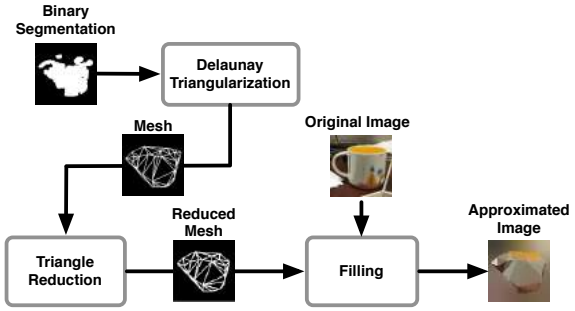


Fig. 8. The process of Triangularization-based encoding.

- *Triangularization:* Given an input image with the foreground/background segmentation, the first step is to generate a binary *boundary map* from the segmentation map, in which, only the pixels on the segmentation boundary have an intensity of 1. A Delaunay triangularization [4] is performed on the boundary map. The parameters of are chosen to produce a high number of triangles to capture boundary details.
- *Triangle Reduction:* Having a set of triangles, the next step is to reduce the number of vertices, iteratively, one vertex at a time. For this, we compute the sum of distances for every vertex from its nearest 3 neighbors, and then remove the one with the minimum sum of distances. The intuition behind this process is that – a region of vertices group together densely because of the non-smooth boundary in the boundary image. Since the goal of removing vertices is to reduce the details and preserve the general shape information, we should pick a vertex from dense regions.
- *Color/Texture Filling:* We provide two options for filling a triangle – with texture or with a single color. For texture, we choose to fill all triangles with a limited set of textures which are derived from regions surrounded by each triangle. To reduce the number of textures, we take an average of textures from different triangles. Fig. 9 shows the process. For each triangle, we transform it to a fixed-size triangle by an affine transform, and then compute one or two average textures for all triangles. For the case of two textures, we apply k-means algorithm.

The size of the compressed image using triangularization encoding with color filling $\mathrm{S_{Tc}}$, and with texture filling S_{Tt} are as follows:

$$\mathrm{S_{Tc}} = \mathrm{B(g([v, c, f]) + B(g(b))} \tag{7}$$

$$\mathrm{S_{Tt}} = \mathrm{B(g([v, c, i]) + B(g(t)) + B(g(b))} \tag{8}$$

Here, v denotes the location of the vertices, c denotes the connectivity list, f denotes RGB color values, i denotes the texture index, and t denotes the reduced DCT coefficients on the texture patch transformed using DCT. All other symbols carry the same definition as in previous sections.

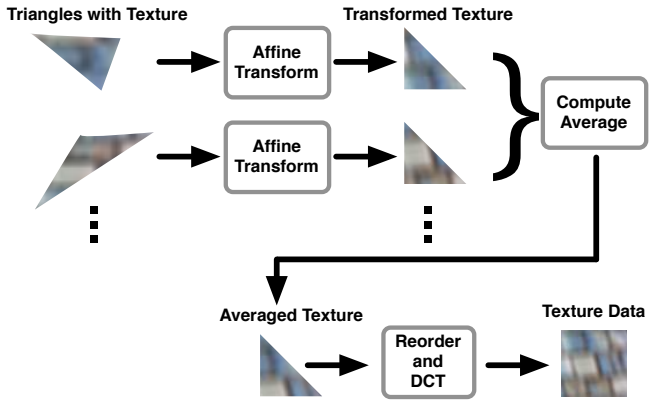


Fig. 9. Triangle texture averaging process.

VII. Empirical Evaluation

In this section, we describe a series of empirical evaluations. First, we evaluate the performance of IMU-guided multiple view capture and depth-refine segmentation. Then our image compression approach is compared with JPEG encoding. After that, we describe a set of results that quantifies the trade offs between the beacon system lifetime and image quality, when the image type and number of beacons are varied. We also perform a full system evaluation involving real users, which is described in Section VIII.

A. Experimental Setup

In all of our experiments, we have used Estimote model REV.F2.3 Radio Beacon [2] having a 32-bit ARM Cortex M0 CPU, 256 KB flash memory, 4 dBm output power, 40 channels (3 for advertising), and 2.4-2.4835 GHz operating frequency. We vary the BLE broadcast interval for a beacon between 100 ms to 2,000 ms. However, an encoded image (broadcasted from multiple beacons) reaches a user's device in less than 1 second. The transmission power is set to -12 dBm, which limits the range of each beacon to about 30 meters. The image writing and reading application runs in a Nexus 5 smartphone having a 2.26GHz quad-core Qualcomm Snapdragon 800 processor, 2 GB RAM, BLE v4, and runs Andriod 6. We mimic BLE 5.0 broadcast packets by a set of rolling BLE 4.0 packets. The rolling mechanism is implemented by configuring the Estimote Location beacons to broadcast customized advertising packets. The customized data is received from an Android compatible LightBlue Bean device [3] via the beacon's GPIO, configured as an UART interface.

We use four types of images in our experiments: images containing road signs, common indoor and outdoor objects, and buildings. Examples of these images are shown in Figure 10. Indoor object images are taken from a 50cm - 150cm

distance. Outdoor objects and signs are taken from 2m to 5m distance. Building images are taken from far. All images are cropped into square shaped images, and are down-sampled to 288×288 pixels prior to compression for a fast disparity map and watershed result computation. Each image is down sampled to 64×64 pixels before writing into the beacon system.

Fig. 10. Test images used in the empirical evaluation.

The two main metrics that are used in the experiments are structural similarity (SSIM) scores, and device lifetime in months. We measure these two under different conditions and show their tradeoffs. The structural similarity scores are used to measure the quality of the produced images when compared to the original ones. The device lifetime is estimated from its relation to a beacon's transmission frequency. Before each experiment, we program the beacons to set a transmission frequency and use the corresponding estimated device lifetime (as reported by the Estimote beacon API) in our experiments.

B. Performance of IMU-Guided Multiple View Capture

We evaluate the accuracy and time to capture multiple views with and without the guidance of IMU. Recall that, the regression tree model takes IMU data and predicts if the current smartphone positioning is good for taking an image for depth estimation, given an already taken image. In the evaluation, for each test, a set of images and their corresponding IMU readings are recorded. If the predicted image produces the best segmentation result when compared to other images in the set, we record this test result as a 'hit', otherwise, a 'miss'. The accuracy of prediction is determined by the ratio of 'hits' to total tests. A total of four tests are performed separately for indoor and outdoor objects. For indoors, there is 1 miss over 4 tests, whereas for outdoors, the model correctly identifies the best image pair for all tests. The model for outdoor objects is more robust since the smartphone's rotation angle shows smaller variations and usually the phone is held vertical. Indoors, users often take pictures of an object from its above, from its below, or from the same hight, which results in a large variation in angles.

Figure 11 shows the expected time for finding a good pair of images with and without the assistance IMU. We observe that the average time to obtain the depth map from a pair of images takes about 2 seconds, which includes the time to focus, raw image processing, and disparity map computation. Since the IMU-guided system predicts a good pair in real time, it significantly decreases the time from 8 seconds to 2-2.5 seconds.

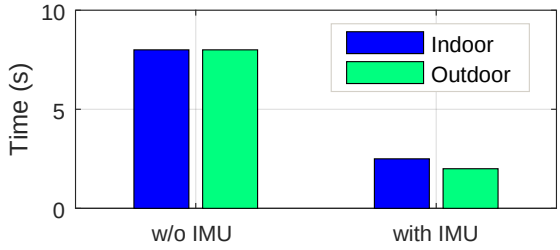


Fig. 11. Performance of IMU-guided view capture.

C. Performance of Depth-Refined Segmentation

We compare our depth-refined segmentation method against a segmentation approach that is purely based on the depth map. We use intersection over union (IoU) as the evaluation metric. We tested our segmentation method on 20 images of four types: road signs, buildings, indoor and ourdoor objects. For each set, we manually generate the ground truth segmentation. Figure 12 shows the result. We observe that our segmentation technique, which combines depth information and watershed segmentation information, outperforms depth-only approach by up to 26%. Our method performs better in cases where the object and the background has larger different in depths, e.g., signs and indoor objects.

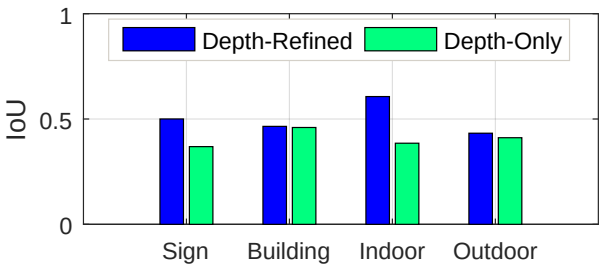


Fig. 12. Performance of depth-refined segmentation.

D. Comparison with JPEG

In this experiment, we compare our proposed image encoding method with JPEG. For an in-depth illustration, we use the

picture of an apple as our test image. We compare four different options of our adaptive image encoding methods in the comparison, i.e. DCT-based, wavelet-based, triangularization with color filling, and texture filling. We measure the quality of a compressed image using Structure Similarity (SSIM) [16]. A SSIM score ranges from 0 to 1, and two identical images have the best SSIM score of 1. We compute the SSIM value between every compressed image and the original image. We plot the SSIM versus compressed image size in bytes in Figure 13.

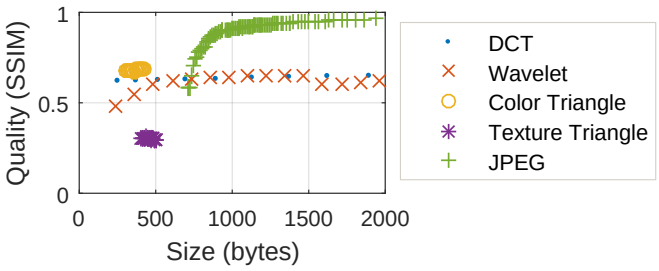


Fig. 13. Image quality versus image size for different encoding methods.

The result suggests that JPEG is able to generate a compressed image with a higher quality (SSIM close to 1). However, such a high quality image has the size larger than 2 KB. When JPEG is set to compress an image into lower than 1KB, the image quality drops sharply. On the other hand, our image compression method is based on foreground/background separation. The background in a compressed image is always blurred. This makes our method impossible to get a high SSIM score larger than 0.8. But our method is able to allocate bits more efficiently under a tight space constraint. This makes our method (when using wavelet/DCT encoding) outperform JPEG when the compressed image size is about 800 bytes. The plot also shows that our method can compress an image into as low as 240 bytes (left most point on the wavelet method curve), which is impossible with JPEG, even in its lowest quality setting.

Triangularization with color filling option performs the best for this test case, and the compressed image size stays less than 500 bytes, while triangularization with texture filling encoding fails to generate a good compressed image.

Besides JPEG, we also studied several other image compression techniques [9][17][18]. But none of these yield suitably small sized images.

E. Effect of Image Type

In this experiment, we test how our image encoding method's performance changes as we vary the type of input images. We consider four categories of images, i.e., road signs, buildings, indoor objects, and outdoor objects. For each image, our algorithm selects the best compressed image from different versions of the adaptive encoded images based on SSIM score. Then we compute the average image quality for a given lifetime for each category of images. We simulate a BLE 5.0 beacon in this experiment to store and read the images. We limit our system to deliver the image data within 0.5 second from one beacon.

The result shown in 14 suggests that, on average, indoor object images achieves the best quality over all types. The building images are best approximated if the beacon system broadcasts packets at a higher frequency, sacrificing the system life time. The road sign images have relatively lower quality that other images, but they tend to last for up to 28 months on a single battery.

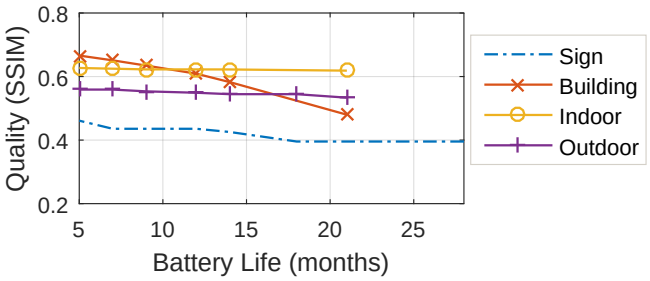


Fig. 14. Image quality versus beacon battery life for different image types.

F. Effect of Number of Beacon Devices

We explore the impact of the number of beacons on image quality. Since each beacon in a system of beacons can be set to broadcast different parts of an image, the more beacons we have, the better quality images we can generate by utilizing the additional space. This is based on the assumption that the image loading time and the device lifetime requirements are fixed.

In this experiment, we vary the number of beacons, and record the SSIM of the best quality compressed image generated by our system. The best image is chosen from the adaptive-encoding results with the highest SSIM score.

We plot the SSIM scores for various expected system lifetime in Figure 15. The experiment results suggest that as the number of beacons is increased from 1 to 3, for the beacon system to have an expected lifetime of e.g., 32 months, the quality of produced images also increases from 0.45 to 0.69.

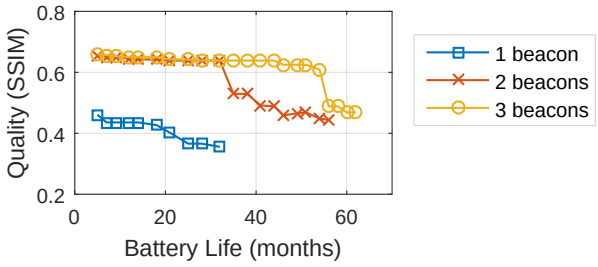


Fig. 15. Image quality versus device lifetime for various number of beacons.

VIII. Real Deployment

We deploy an image beacon system in an indoor environment to evaluate the ability of image beacons in delivering visual information of various real-world objects. An

approximated image stored in the beacon system contains an object's shape as well as its texture information. The goal of this deployment experiment was to understand how an image beacon system delivers these two kinds of information of an image.

To have a subjective measure of the performance of an image beacon system, we conduct a user study involving 20 participants. Each participant is given a smartphone that receives image broadcasts from four different beacons placed inside a room. The goal of the user is to identify the objects in all four images, and then locate them in the room.

The four broadcasted images are of an apple, a chair, a text book and a computer mouse. For each image, we compress it under three size constraints: 256 bytes, 512 bytes, and 768 bytes, and obtain the best quality image produced by our compression algorithm under the constraint. The images compressed in three levels along with the original image are shown in Figure 16. We choose these sizes to mimic 1, 2, and 3 BLE 5.0 beacons, respectively; but our actual implementation used a rolling mechanism with BLE 4.0 (connected with an Arduino), as described earlier. Each user is progressively shown a better quality image until he is able to identify the object in the room.

We make sure that there are at least 3 objects in the room that are similar to the one that the user is looking at on his phone. To test if an object's texture details are preserved, for the apple test case, we put another apple having a green/red mixed color and an orange next to it. For the book test case, we put two other similar sized books next to it that have different covers. To test if an object's shape details are preserved, for the chair test case, we add another two chairs of the same color. For the mouse test case, we add an iPhone and a mac mouse. Figure 17 shows photos of the objects along with the objects that we added to introduce confusions.

The experiment results are shown in Figure 18. For each size limit, we plot the number of correct guesses by our participants for various categories of images. We observe that, as expected, when the image size limit is larger, participants tend to perform better. Even with the lowest size, at about 50% images were always guessed correctly by the participants. Therefore, with 3 or more beacons, the image quality of our system is high enough to let people distinguish very similar objects.

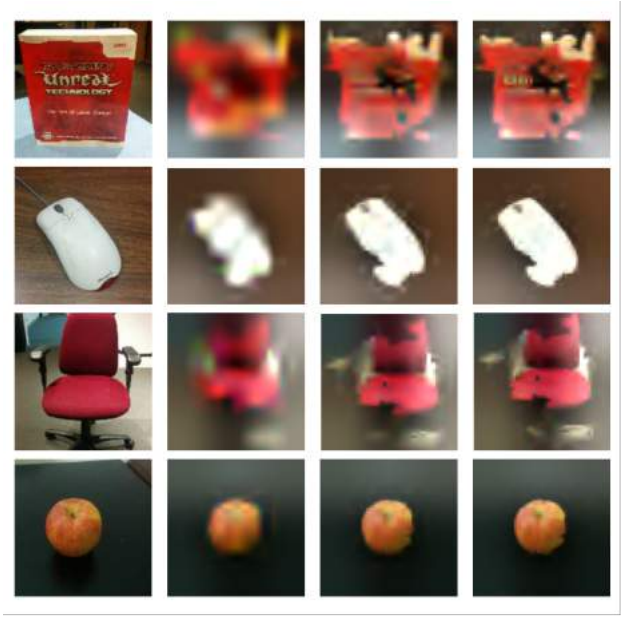


Fig. 16. Test images compressed in three quality levels.

Fig. 17. Photos of the object used in the experiment.

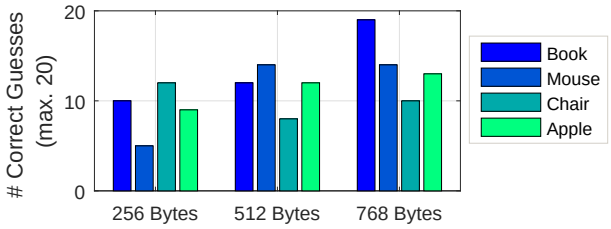


Fig. 18. Responses from user study.

IX. Discussion

Our proposed image beacon system only considers stationary objects. This is an inherent problem of any depth estimation technique. In such case, we have to resort to texture or color based segmentation.

Our IMU-based prediction algorithm uses a regression tree model. Its prediction accuracy is lower indoors than outdoors. A robust model, may train a separate regression tree for different cases, such as one model for taking images on objects below the phone, one model for objects on the same height to the phone.

A property of marker-controlled watershed segmentation algorithms is that they always generate a clean segmentation result. Our combined segmentation method does not fully exploit this feature. We could further enhance the power of combining depth estimation and watershed results by deploying a smarter foreground region growing method.

Animated images (e.g., GIFs) are not supported by the system. However, with additional optimization for time-domain redundancy, we believe it is possible to develop image beacons that supports animation.

X. Related Work

Previously, a binary images beacon system citeshaoyears was developed to enable BLE beacons to store and broadcast binary images. The limitation of the system is that it only

supports a limited category of binary images such as hand-written characters, shapes, and symbols.

A rich set of image segmentation methods exists in the literature. In the past decade, more modern techniques involving machine learning has been invented. State of art neural-network based method [19] can achieve a very accurate result (highest score 90.4 on IoU evaluation on airplane type testing data). The method is based on neural-network and conditional random field. However, this method is computationally expensive to run on a cellphone. Otsu's method [11] is based on finding a separation on image pixel intensity histograms, which does not take care of local image structure. Gabor filter segmentation is based on finding edges in an image using Gabor filtering. Marker-controlled watershed methods [12] use mathematical morphology to pre-process the data, followed by a watershed segmentation. This avoids over segmentation, which is a weakness of traditional watershed method.

The notion of foreground / background information can be obtained by disparity estimation with semi-global matching[6]. The method enforces smoothness in the neighbor matching process to reduce matching errors.

Shapiro [15] developed an image encoding technique named embedded zerotree wavelet (EZW) encoding, which is computationally expensive and slow. Said and Pearlman [13] developed a better wavelet-based image encoding method based on set partitioning in hierarchical trees. This method gives similar image compression performance in terms of quality and size, and the same time it achieves a faster computation time. We have used this method in our system.

Lu et al. [8] introduced a piece-wise linear image encoding method using surface triangularization. Their triangularization algorithm fits the image surface in a top-down manner. The idea is to apply a constrained resource planning to allocate the least amount of triangles while achieve a small image approximation error. Their experiment results shows that the triangularization method compresses images with a compact code length with a guaranteed error bound. But their method is limited to approximating gray-scale images and theit target is to achieve near lossless compression, while our ImageBeacon system's triangularization method supports color images, and is a lossy compression method in general.

XI. Conclusion

In this paper, we described a system involving beacon devices and smartphones with BLE receiving functionality. Our system allows a user to generate an approximation of an input color image, write the approximated image into the very limited beacon device storage, and receive compressed images from beacon devices' broadcast packets. The main contribution of this work is the overall system construction, the adaptive encoding image compression method, the evaluation of various parameters of the system, and quantifying the trade-off between image quality and beacon battery lifetime for our image compression method. Our work widens the usage of the energy efficient, long-lasting beacon devices by allowing easy storage and access of custom image data in scenarios where there is no Internet connection.

Out system will perform at its best with the beacons that adopts the upcoming Bluetooth 5.0 standard. A future work is to evaluate the different aspects of the system performance as the Bluetooth 5.0 is released and gets popular. Moreover, a 'fat-beacon' standard is under development at Google, that allows an even higher broadcast transmission capacity for BLE beacons. The goal of that standard is to equip beacon devices with the ability to broadcast basic web contents to smartphones in absence of the Internet connectivity. It will be meaningful to study the application of our image beacon system combined with a fat beacon.

References

[1] Bluetooth 5.0 Press Release. http://tinyurl.com/hvrosf5.

[2] Estimote Beacons. https://estimote.com/.

[3] Lightblue bean. punchthrough.com/bean.

[4] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[5] A. Dementyev, S. Hodges, S. Taylor, and J. Smith. Power consumption analysis of bluetooth low energy, zigbee and ant sensor nodes in a cyclic sleep scenario. In *Wireless Symposium (IWS), 2013 IEEE International*, pages 1–4. IEEE, 2013.

[6] H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 807–814. IEEE, 2005.

[7] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[8] T. Lu, Z. Le, and D. Yun. Piecewise linear image coding using surface triangulation and geometric compression. In *Data Compression Conference, 2000. Proceedings. DCC 2000*, pages 410–419. IEEE, 2000.

[9] S. A. Mohamed and M. M. Fahmy. Binary image compression using efficient partitioning into rectangular regions. *Communications, IEEE Transactions on*, 43(5):1888–1893, 1995.

[10] S. Nirjon and J. A. Stankovic. Kinsight: Localizing and tracking household objects using depth-camera sensors. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, pages 67–74. IEEE, 2012.

[11] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[12] K. Parvati, P. Rao, and M. Mariya Das. Image segmentation using gray-scale morphology and marker-controlled watershed transformation. *Discrete Dynamics in Nature and Society*, 2008, 2009.

[13] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on circuits and systems for video technology*, 6(3):243–250, 1996.

[14] C. Shao, S. Nirjon, and J.-M. Frahm. Years-long binary image broadcast using bluetooth low energy beacons. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS 2016)*, 2016.

[15] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on signal processing*, 41(12):3445–3462, 1993.

[16] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[17] S. Zahir, K. Dhou, and B. Prince George. A new chain coding based method for binary image compression and reconstruction. *PCS, Lisbon, Portugal*, pages 1321–1324, 2007.

[18] S. Zahir and M. Naqvi. A new rectangular partitioning based lossless binary image compression scheme. In *Electrical and Computer Engineering, 2005. Canadian Conference on*, pages 281–285. IEEE, 2005.

[19] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.