FLASH: Video-Embeddable AR Anchors for Live Events

Edward Lu* John Miller* Nuno Pereira* Anthony Rowe*

Carnegie Mellon University

Abstract

Public spaces like concert stadiums and sporting arenas are ideal venues for AR content delivery to crowds of mobile phone users. Unfortunately, these environments tend to be some of the most challenging in terms of lighting and dynamic staging for vision-based relocalization. In this paper, we introduce FLASH[1], a system for delivering AR content within challenging lighting environments that uses active tags (i.e., blinking) with detectable features from passive tags (quads) for marking regions of interest and determining pose. This combination allows the tags to be detectable from long distances with significantly less computational overhead per frame, making it possible to embed tags in existing video displays like large jumbotrons. To aid in pose acquisition, we implement a gravity-assisted pose solver that removes the ambiguous solutions that are often encountered when trying to localize using standard passive tags. We show that our technique outperforms similarly sized passive tags in terms of range by 20-30% and is fast enough to run at 30 FPS even within a mobile web browser on a smartphone.

1 Introduction

Mobile Augmented Reality (AR) platforms like ARKit and ARCore have opened the door for new types of interactive content that are easily accessible to the masses. One compelling use-case is the inclusion of AR effects in entertainment venues, like sports arenas, theaters, and concert halls. Figure 1 shows two potential scenarios where users could see additional content overlaid on a live action event. The first scenario is a car race where fans in the bleachers could look through their phones and see annotations about the statistics of passing cars with comparative overlays of racing lines and breakpoints laid out on the track. Many of these effects are already being added to television coverage but would be equally (if not more) valuable for fans physically present at the event. The second example shows additional effects that could be added to a live indoor concert performance where the background lighting and staging is often wildly variable. In this case, we see an image capture from the BTS boy band tour "Love Yourself; Speak Yourself," where the creative directors applied live AR effects to footage shown within a stadium in real-time. One could imagine a personalized version of these effects delivered through mobile AR with the ability for the crowd to directly interact with and influence the content. Alongside these entertainment use-cases, one could also imagine adding AR signage for navigation, friend finding, advertisements, concessions, etc.

One of the biggest challenges in making these types of applications work in practice is the ability to accurately estimate the pose of the user's camera. With current mobile AR applications, this is done through image matching-based optical registration or with purpose-built AR tags. Unfortunately, in entertainment scenarios, the conditions are extremely challenging for most vision-based techniques due to changes in the visual appearance of the event from

Figure 1: FLASH example use-cases. Sporting event (top) and Live entertainment (bottom).

dynamic lighting and set changes. For robust tracking, we advocate using a small amount of dedicated visual infrastructure in the form of optical tags. However, standard tags would need to be extremely large and artificially illuminated. To enable truly robust camera pose estimation in entertainment environments, we need to overcome the following three challenges. First, a marker needs to be robust to lighting changes and to not be affected by dramatic scene changes. Some systems currently attempt to localize based on visual features naturally occurring in the venue. This is difficult in environments like theaters or concerts, where the background changes dramatically with new sets. Second, markers need to be detectable from a long distance and across a wide viewing angle. This is especially critical in entertainment applications to avoid making them too obtrusive such that they would detract from the main event or disrupt the flow of a story. Third, the marker should be able to provide an accurate camera pose estimate without having to significantly move the camera. Most optical markers have been designed with tracking the tag as the primary goal, as opposed to estimating the pose of the camera. When detecting 2D tags, there is often an ambiguity due to detection noise that makes it difficult to determine which of two potential camera poses is the correct solution. We see that in practice, the lower residual solution is often *not* the correct pose of the two. In entertainment environments, this effect is compounded by long distances to the marker as well as the fact that content is

*e-mail:{elu2, jmiller4, npereira, agr}@andrew.cmu.edu

[1]Flickering Light Augmented reality SHapes

likely placed around the entire field of view and not just attached directly to the marker. In robotic systems, this ambiguity problem is typically solved through motion filtering, but that is not practical in a seated scenario.

In this paper, we present FLASH, an active optical AR anchor system specifically designed for entertainment venues. Instead of relying on passive visual markers, we propose using an active optical marker that can be displayed on digital signage within a stadium. Active optical tags, like LightAnchors and InfoLED [1, 41], use blinking light patterns to encode an identification code (ID) of the tag that can be determined by processing a video sequence [1, 6, 33]. Instead of encoding the tag value spatially like passive tags (QR codes, AprilTags, AR Tags, etc), active tags encode their data over time across video frames. This means that the tag is still decodable with a much smaller surface area, hence providing significantly more range.

Current active optical tags were not designed for the purpose of camera registration and lack the features needed to estimate viewing pose. The one example of an active tag that does address camera pose [33] is extremely computationally intensive (using GPU acceleration on a native app), requires a constellation of multiple tags for pose (camera pose can't be computed from a single tag), and suffers in terms of range of motion since it doesn't have trackable features that can be identified on each frame. FLASH takes the geometry-based recognition features of passive tags and applies them to active tags to create a more versatile option for mobile AR entertainment applications. We propose using a quad on a video display where the center component flashes at half the typical camera frame rate. The quad could be completely passive (a black box on a white background) or it could be drawn as part of the video feed. While conceptually quite simple, this allows us to optimize a decoder specifically to more easily locate and decode tags compared to passive alternatives. It also provides the structure needed to determine camera pose from a single target as opposed to requiring a constellation of markers. To combat the ambiguity problem, we present a perspective-n-point solver that utilizes the gravity vector easily available on mobile phones. Given that our goal is to display these tags on existing video monitors, we design a decoder that works with standard display refresh rates.

In order to maintain correct AR overlays on live video data, we design FLASH to be significantly faster per frame compared to current passive tags. To achieve video framerates, FLASH uses a higher edge-to-quad contrast ratio, which is possible due to the active illumination of the tag. Since the contrast ratio is higher, we can tune our quad detector to operate with fewer false positives, which means less time wasted attempting to decode non-tags compared to typical passive tags printed on paper. This boost in efficiency makes it possible for us to operate FLASH at full frame rate (30 FPS) even within a mobile browser on a modern smartphone, while still providing 6DOF localization using the quad corners. This means that with Web frameworks like AR.js [23], it is possible to see AR content without even installing an app on most phones (with the recent addition of `MediaDevices.getUserMedia()` on iOS, it is supported in over 95% of the browser user base) [26]).

We discuss the design and then evaluate FLASH along a number of dimensions. We show that it is able to increase range by 25-35% over AprilTags of a similar size, as well as almost doubling the total viewable area. Most importantly, we show that FLASH is lighting invariant and significantly more immune to hand shake since it doesn't suffer as significantly from motion blur. We also provide an in-depth analysis of how camera and tag geometry contribute to pose ambiguity and show that our gravity assisted solver is able to select the correct solution even in the presence of noise.

In summary, the contributions of this paper are:

1. Design and evaluation of a hybrid active optical marker for long-range, wide-angle, highly dynamic environments
2. A camera-tag system that is compatible with standard video screens
3. A gravity-assisted pose solver for mobile AR applications
4. An open-source implementation that runs within a mobile phone Web browser.

2 Related Work

We classify previous work for content registration in AR into two main approaches: (1) marker-based and (2) model-based. We also discuss some of the recent advances in computer vision pipelines in Web browsers which are complementary and enabling technologies in subsection 2.4 as well as some of the custom hardware solutions used for headsets.

2.1 Marker-based

Visual markers (also called fiducial markers) have, for a long time, been used as a practical solution for content registration in AR [14]. Over the last few decades, researchers have proposed many different markers, such as AprilTags [22], AR Tags [9], and ARToolKit [38] (to name just a few). These tags can be very effective and easy to use, but they have limited range and can often be quite obtrusive. Other systems, such as Vuforia [40] can learn features from arbitrary images and thus do not have that problem. However, these image-based systems tend to be more limited in the number of tags they can support and are often less robust compared to tags specifically designed for positioning.

Common to all previous visual markers is the fact that they are highly dependent on lighting conditions and can, depending on their size, only work over short ranges. Researchers have also explored the idea of using active tags [1, 3, 6, 15, 24, 33, 41], which can be more robust to lighting conditions. Of these approaches, visible light communication (VLC)-based techniques [15, 24, 39] only offer relatively coarse localization, work at short ranges, require very low camera exposure settings to avoid saturation (which is not ideal for AR), and require high blinking rates (1KHz+ rates) that are not compatible with commodity displays. Other active visual tags [1, 6, 33] are not practical in common mobile devices as they either require a special vision sensor [6], high camera frequencies (not yet compatible with current ARKit/ARCore) [1], support relatively short ranges [1,41], or are computationally very expensive and suffer from hand shake motion [33]. Visual MIMO [3] is another related active tag technique. It uses a combination of spatial and temporal coding to enable high data rate communication, but it is also designed for high frame rate cameras (hundreds of FPS or greater) over short distances. All spatial coding techniques trade off data density for range. FLASH attempts the extreme opposite, where data is decoded from potentially a single pixel across multiple frames.

2.2 Model-based

Model-based methods are attractive as they don't require markers in the environment. For example, authors have shown methods for camera pose tracking for outdoor AR [25], and these can be used for pose estimation given a known 3D model of the environment and initial camera position. Instead, simultaneous location and mapping (SLAM) can provide pose estimation with respect to a known 3D model using visual or depth [12, 20, 35] sensors. Many recent headsets [16, 19, 36] and mobile AR platforms like ARKit [2] and ARCore [11] use SLAM to determine the device's pose without the use of initial camera positions. SLAM requires acquiring a model of the space before a location can be determined (increasing acquisition latency) and doesn't work well in low-feature environments or when the scene changes. Active optical markers complement these approaches to reduce the time and robustness of model acquisition.

Determining a camera pose from known correspondences between 3D reference points and their 2D image projections is a well

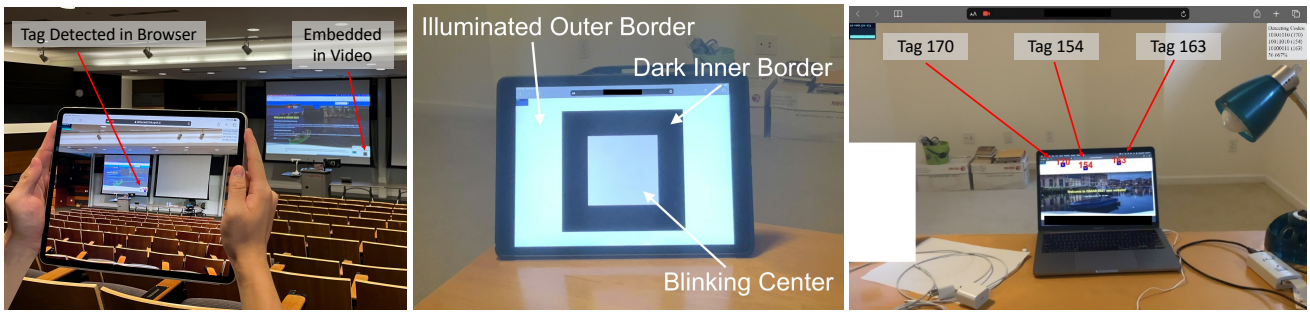

Figure 2: FLASH active tag embedded in slides during a talk and detected in a browser on an iPad (left). Zoomed in view of the tag in its "on" state shows its simple anatomy (middle). Three small tags with ID 170, 154, and 163 displayed on a laptop (right).

studied problem in computer vision, known as the perspective-n-point (PnP) problem [10]. Classical methods to solve the PnP problem formulate a non-linear least-squares problem and then solve it using iterative optimization [18]. Numerous solutions to the PnP problem have been proposed in the last two decades, many with specialized solutions to the problem for a particular number of points, with few practical implementations [34]. One notable formulation [17], adopted by the ORB-SLAM [20] solver, is based on linearization and, while fast, may not be applicable to a small number of points. In this paper, we leverage using the gravity vector from the phone to help improve camera pose localization.

2.3 Specialized Solutions

Many commercial headsets employ beacons or trackers for localization. For example, the HTC Vive [13] uses a sweeping IR laser to detect horizontal and vertical angle and can be very accurate. These, however, require powered beacons installed in the environment and are not designed for long ranges. Interestingly, the Oculus Rift [21] uses blinking IR LEDs on the headset, which are detected and decoded by a fixed IR camera. This system has some similarities to active tags presented previously and to our approach. Still, it requires very tight synchronization between the LEDs and the camera, not achievable in mobile phones without adding specialized hardware.

UWB localization [8] has shown increasing potential in supporting AR applications, but also requires external hardware in mobile phones[2] and are not robust in high multi-path environments or over large distances.

2.4 Browser Support

Recent advances in Browser technology support the ease of the Web to load AR content without installing an app, which we deem important to facilitate the adoption of these technologies and is an important enabler of this work. With the increasing adoption of WebXR [37], we will have access to standard APIs that allow support for very different display capabilities and experiences, and, while current WebXR [37] support is still immature [27], it is possible to experiment with it in a wide variety of platforms [29].

WebXR's features for advanced computer pipelines is yet under proposal [30]. For this work, we can use the existing Web API's `MediaDevices.getUserMedia()` to access camera frames, which, as of May 2021, is supported in 95% of the browser user base, including iOS which added support very recently (December 2020, with the release of iOS 14.3) [26]. Together with WebAssembly [4, 31], which efficiently runs codebases for computer vision previously developed for native applications in C/C++ [5], we can build very powerful computer vision pipelines that run on all major browsers. This work leveraged previous implementations of the AprilTag decoder [32] for our implementation and evaluation on the Web.

[2]UWB hardware is available in the latest iPhone; however, it is not accessible to third-parties, and its localization performance is still to be determined.

3 System Design

In this section, we discuss the design decisions and operation of FLASH tags along with its detector. Our goal was to create a robust and computationally efficient active optical anchor decoder for camera pose estimation that works with commodity video displays. We targeted a decoder that would operate at reasonable AR framerates even within a mobile Web browser to facilitate widespread adoption. Unlike many traditional VLC-style systems, we assume that the camera decoder knows the potential set of tag IDs in the scene, allowing us to design a decoder that needs to perform detection instead of data transfer.

3.1 Tag Design

FLASH tags are simply quads (squares) displayed on a screen with three main components: (1) an illuminated outer white border, (2) an inner black border for the quad, and (3) a central region that can change from dark black to bright white depending on the state of the tag. Figure 2 shows an example of a tag in the "on" state displayed on an iPad. Note that the innermost blinking shape at the center does not necessarily have to fill the entire quad, but should fill up most of the inner part of the tag. The inner region of the tag blinks a coded identification signal using an On-Off Keying (OOK) scheme that repeats continuously over time. Since the tag ID will be decoded using a camera, many constraints need to be considered when designing the tag's symbol rate and duty cycle, as we discuss next.

3.1.1 Symbol Rate

While we could build a native application to decode tags at higher frame rates by enabling mobile Web support, this limited us to 30 frames per second (FPS) for our decoder at reasonable resolutions. This 30 FPS constraint requires that the blinking rate of our tag is limited to (FPS/2) Hz. Since we know that the decoder will run at twice the frequency of the tag, it will sample a single on/off state twice.

Figure 3 shows how a camera at a sampling rate of 30Hz could potentially align with a screen flashing at 15Hz. Let n be the code length in bits. After a full ID is transmitted, our decoder will have captured $2n$ values for a single tag. However, we know that we may need to sample the odd bits or the even bits to correctly decode the tag ID due to sampling time skew. One of the codes will be in the acceptable data set, while the other alternative will likely be random. To avoid nearby conflicts in codes, we recommend putting codes with relatively orthogonal IDs in areas where they might overlap with other codes. Though the number of bits is variable, in practice, we typically use 8-bit rolling codes. Using rolling codes reduces the available number of codes since values like `0b001` are identical to

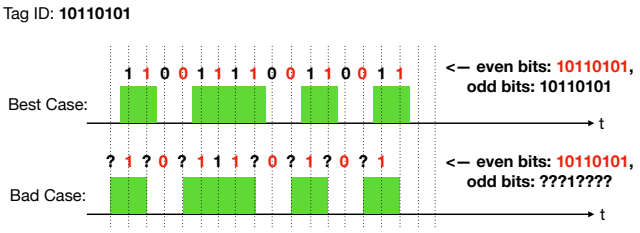

Figure 3: Timing diagram of a decoder searching for tag ID `0b10110101` in two possible cases. Dotted vertical lines are when a camera samples a frame. A green box means a tag is in an "on" state. The top is the best/typical case, where the camera is out of phase with the tag. The bottom shows a bad case where the camera's frame capture is perfectly in sync with a tag's blinking and some values are unknown to the decoder. Since our system matches using even *or* odd video frames, the unknown values do not affect the decoder's ability to find a tag ID.

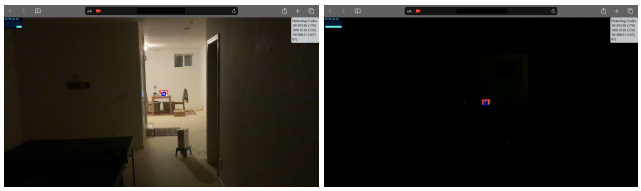

Figure 4: FLASH tag with ID 170 being detected at 12.192m (40ft) under medium (left) and dark (right) lighting conditions within an iPad Web browser window.

`0b010` and `0b100`. For a bit sequence of length $n = 2^m$, the total number of unique codes/tag IDs would be:

$$\sum_{k=1}^{m} \frac{2^{2^k} - 2^{2^{k-1}}}{2^k} \tag{1}$$

Using Equation 1 for an 8-bit bit sequence ($m = 3$), we get 36 unique codes. Given this symbol rate, the average latency for detecting an 8-bit tag is $1/(15Hz) \times 8bits \times 1000ms/s = 533ms$.

3.2 Tag Detection

Our tag detection system includes four major steps: (1) blurring and grayscale conversion, (2) quad detection, (3) quad tracking across frames, and (4) an ID decoding scheme. As mentioned above, the decoder is searching for a specific list of IDs that are expected to be within the field of view, and it should be assumed that we are trying to minimize detection latency given that the decoder can begin capturing frames halfway through a rolling data sequence. We will describe the four tag detection steps in the remainder of this section.

3.2.1 Blurring and Grayscale Conversion

Since active tags can encode their IDs across time, we are less reliant on sharp details. High-resolution images can produce a substantial amount of false-positive quads, leading to longer processing, more tracking, and a slower detector. We applied a Gaussian blur to reduce the false positive rate, similar to most optical tag systems, which reduces overall computation time in later processing stages. We also converted the image to grayscale for processing. We ruled out using color information for increasing code density since color consistency was wildly variable across different cameras, exposures, and white balance settings. In many cases, bright objects saturate mobile phone cameras making extraction of meaningful color information extremely inconsistent. As camera image quality improves, this is something that would be worth investigating in future work.

3.2.2 Quad Detection

Once the image has been pre-processed, we now identify all the potential quad location candidates. We use AprilTag's quad detection algorithm [22] with adjusted parameters to bias the detector for higher contrast edges. We also allow smaller quads to be detected, since we no longer require a minimum pixel density within the quad to hold spatially coded data. We achieve this by increasing the value used for the minimum difference threshold between white and black quad edges in the AprilTag quad detector. In addition to adding an image blur before the frame processing, this dramatically reduces computation time for quad detection as discussed in Section 4.2.

3.2.3 Quad Tracking

Next, we need to track quad locations across frames to ensure correct data bit matching with motion. Once a quad is detected in an image I_t, it is matched with its nearest neighboring quad with the closest center found in the previous video I_{t-1} frame. We enforce constraints on quad displacement and quad shape difference between frames to aid in selecting the most likely sequence of quads across frames. For instance, a quad in I_t is rejected if the distance of its center to the center of its nearest neighbor in I_{t-1} is above a given threshold, or if the average distance of each corner of the quad from the center is far greater than that of its nearest neighbor. This process is applied to all quads found in I_t.

The average intensity value of the region in the center of the quad is saved across $2n$ frames that will later be used to match against the current code book. The quad is rejected if the difference between the highest and lowest intensity values of a quad across the $2n$ frames is greater than a given threshold (similar to an RSSI value). These $2n$ image intensity values are saved and form the ID of a possible tag detection.

A valid tag may not be detected at a far enough distance for $2n$ consecutive frames and will constantly be rejected due to the constraints listed above. In these cases, we interpolate intensity values for n frames for tags/quads we have already validly decoded to account for this noise. In other words, after determining that a quad is a valid tag, it has some leeway before the decoder rejects it, allowing for a few frames where the quad wasn't fully decoded. The quad associated with a tag can be missing at most $ntries$ frames before giving up on the tag, where $ntries$ is configurable based on the environment, and it is typically set to the bit length of the code divided by 2 or 3.

3.2.4 Decoding

Once a quad has accumulated $2n$ center intensity values, the decoder checks to see if the value occurs in our current codebook. The $2n$ brightness values are normalized and thresholded to create a binary code of size $2n$ bits. The threshold value is calculated as the average of all $2n$ intensity values. As previously mentioned, we compare the ID formed by the even n bits and the odd n bits to each ID in our codebook. If a quad has a valid ID, it is returned as a detection. See Figure 2 for an example of FLASH correctly detecting three very small tags simultaneously. A detected tag returns a value and coordinates the four corners of the quad within the camera image. We also assume that the pose of the tag is known and can be passed to our gravity-assisted decoder to determine the camera's pose.

It is worth noting that the detection latency of FLASH is potentially longer than the time it takes to find a passive tag in a single frame. However, we do get a pose update for the tag on each quad that is detected. We currently return the quad within the sequence of frames with the lowest residual error as the selected pose, but this could be improved by averaging the pose across all frames. It would also be possible to return a pose update for each quad detected, allowing a much higher frame rate stream of poses once the tag is identified.

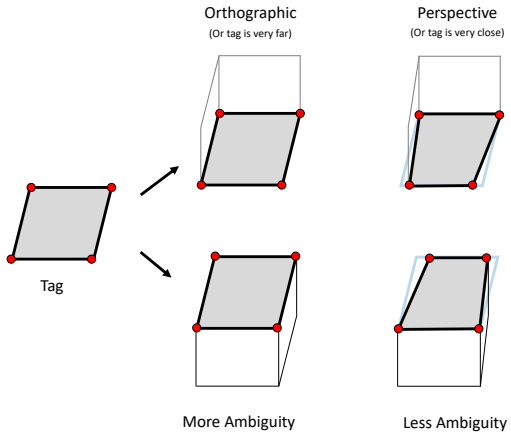

Figure 5: Tag pose ambiguity illustration. Is the viewer seeing the tag from above looking down at an angle, or from below looking up at an angle? Perspective projection alleviates the issue, but only when the tag is close or very large.

3.3 Gravity Assisted Localization

In this section, we describe how to determine the camera's pose from the detection of FLASH quad corners and how gravity sensing with an accelerometer can significantly improve the robustness of the pose estimation. Camera pose is necessary for displaying AR content that is registered in the correct location relative to the real world. For example, to display informative markers on a sporting field, the system needs to know the 6DOF pose of the camera relative to the field.

We start with the assumption that the pose of the FLASH tag in world coordinates is known. This can be determined by measuring the pose of the tag using surveying techniques. If only one tag is used and no other venue infrastructure is in place, it is viable to consider the location of the tag to be the origin of the coordinate system, which forgoes the need for surveying.

The FLASH detection system outlined in prior sections returns the locations of the quad corners on the camera screen in pixel coordinates. From these pixel coordinates and with knowledge of the geometry of the tag, many methods in the literature describe how the pose of the tag relative to the camera can be determined. Several of these works are described in Section 2.

Once the pose of the tag in camera coordinates has been determined, it is easy to infer the location of the camera in world coordinates. Given the pose of the tag in camera coordinates and the pose of the tag in world coordinates, the pose of the camera in world coordinates can be obtained by simply composing the inverse of the former with the latter.

However, due to an ambiguity issue described in the following subsections, it is common for localization methods in the literature to return multiple solutions for the pose of the tag relative to the camera. FLASH incorporates accelerometer measurements to disambiguate the solutions and choose the correct one. Accelerometer measurements are common on mobile devices designed with AR in mind and are even easily obtainable from within a Web browser environment. Thus, it is natural to use these sensor measurements to aid in the camera localization.

3.3.1 Planar Ambiguity

Planar fiducial markers have a fundamental ambiguity issue, illustrated in Figure 5. In short, when viewing a planar tag from an oblique angle, there are two solutions for the pose of the tag relative to the camera that result in very similar projections onto the camera screen. The two possible poses amount to a reflection about a plane normal to the line between the camera center and the tag center. Typically, one of the two solutions will have lower reprojection error than the other, which is a consequence of perspective foreshortening as shown in the right side of Figure 5. This is utilized by most modern PnP algorithms, such as [7], which distinguishes the two solutions by choosing the one with lowest reprojection error.

A downside to this approach is that the global minimum reprojection error does not always correspond to the correct pose solution; noise in the corner detection can cause a situation where the incorrect solution actually has a lower reprojection error than the correct one. This is especially common when the tag is far from the camera, in which case the foreshortening effect is diminished and the reprojections of the two candidate poses are almost identical. Instead, FLASH uses accelerometer measurements to distinguish the solutions.

3.3.2 Evaluation of Ambiguity Occurrence

We demonstrate the prevalence of this ambiguity issue empirically through numerical simulation. We consider the case where a quad is placed at the origin, with its four corners lying on the $x-z$ plane. We then simulate camera poses in a surrounding region (always facing the quad) and determine where the projected corners fall on the simulated camera screen, assuming perfect corner detection with sub-pixel precision. We then compute the two candidate pose estimations using the IPPE algorithm [7], which also returns the associated reprojection error for each of the candidate solutions, which we convert to pixels. For these experiments, we used a 15cm quad, a 720p camera with focal length 1000 pixels, and a camera distance ranging from 0.5m to 6m. In Figure 6, we show a log plot of the average reprojection error of the two solutions for a variety of horizontal camera positions at three different vertical viewing angles.

Typically, the two solutions are disambiguated by choosing the one with the smallest reprojection error. However, we see from the simulations that as the camera moves more than a meter or two away from the tag, the average reprojection error becomes very small, even for the incorrect solution. Consequently, only a small amount of corner detection error is required to cause the algorithm to return the incorrect solution. In practice, this means that at longer viewing distances the two solutions quickly become indistinguishable. Furthermore, in Figure 7, we show the severity of the error between the two solution poses by showing the difference in the inferred camera location, which can be several meters at modest distance from the tag.

3.3.3 Resolving the Ambiguity Using Gravity

By using gravity as a reference, it is possible to disambiguate the two solutions. Since the camera device is assumed to have an accelerometer that is accessible from within the Web browser, we can determine the direction of the gravity vector with respect to the camera. Similarly, since we know the pose of the tag with respect to the world (one of our baseline assumptions), we can also determine the direction of the gravity vector with respect to the camera.

Given the two candidate solutions for the pose of the tag with respect to the camera, we resolve the ambiguity issue as follows. First, we transform the gravity vector from the tag frame to the camera frame using each of the two candidate solutions. Then, we compare the two results (gravity vector in camera frame) to the measured result from the accelerometer. We can then select the solution that best aligns with the measured gravity vector using a dot product. We use the same approach to resolve the ambiguity that forms due to the four rotational symmetries of the FLASH tag.

One limitation of this approach occurs when the camera is perfectly level with the ground, with the camera's focal axis parallel to the gravity vector. In this case, both of the tag-to-camera solutions will have the same gravity vector in the camera frame. A similar problem occurs if the tag is placed parallel to the ground, in which

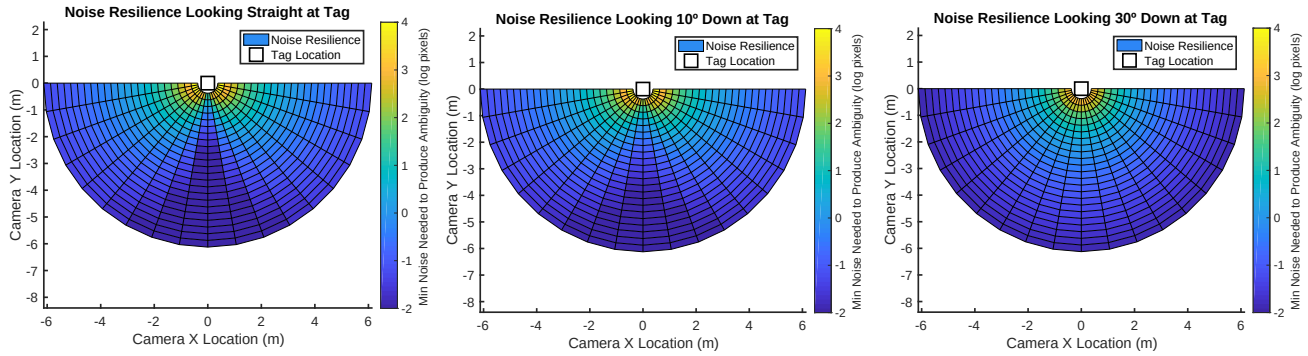

Figure 6: Sensitivity to pose ambiguity error based on angle and distance. The lower the reprojection error, the easier it is to confound the two candidate solutions.

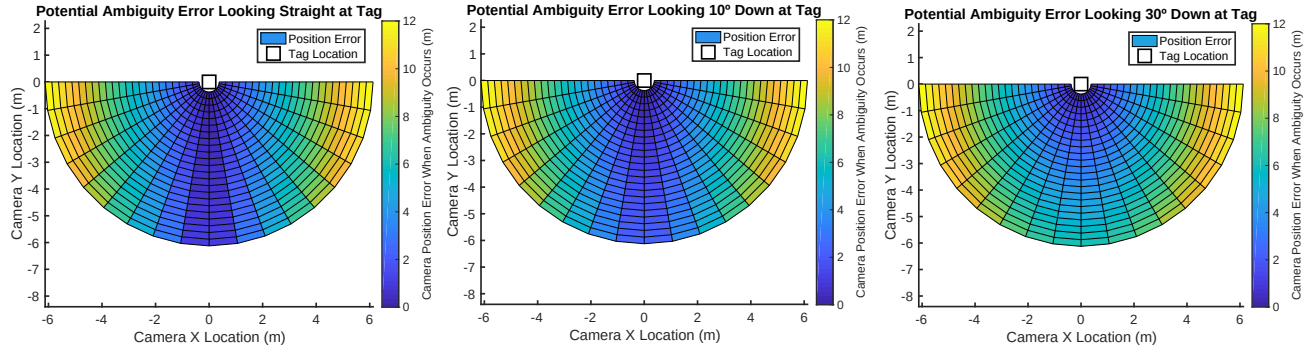

Figure 7: Severity of the ambiguity error at each position when it occurs.

case the four rotational symmetries of the quad produce an ambiguity that cannot be resolved using gravity. However, these limitations are not significant in practice, since video displays are not usually mounted on perfectly horizontal surfaces.

3.4 Implementation

FLASH's implementation has two parts: the active tag implementation and our Web-based decoder framework, which we describe in the following subsections.

3.4.1 Active Tag

For simplicity, we designed Web applications for both the tags and the decoder. Tags could also be generated and stored in video, but we found it useful to be able to quickly and synthetically generate arbitrary shapes and codes that can be viewed within a browser. Figure 2 shows examples of what these tags look like when displayed in an animated Web page. Again, the tag can be implemented in any manner, as long as it adheres to the rules detailed in Section 3.1.

3.4.2 Web Framework

The speed of FLASH and advancements in Web technologies such as WebAssembly (WASM) [4, 31] allow FLASH to be fast enough to be run in a mobile browser on a modern smartphone or tablet. We capture images using live video from the Web API `MediaDevices.getUserMedia()` and leverage WebGL for real-time blurring effects and grayscale conversion. These video frames are buffered and queued up for processing to ensure we sample an image at nearly 30 FPS.

FLASH's quad detector is built off of AprilTag's quad detector with an adjusted contrast threshold for quad edges, detailed in Section 3.2.2. This quad detector, along with our quad tracking and tag decoding code, was compiled to WASM for faster performance over JavaScript on the Web. Quad detection, tracking, and decoding are done in a Web worker in parallel with live video frame capture and pre-processing to make sure the detector runs at a smooth 30 FPS. In Section 4.2, we will evaluate the performance of our pipeline with a breakdown of each step.

4 Evaluation

In this section, we evaluate the performance and robustness of FLASH in comparison to AprilTag 3 [22], a popular passive tag-based system widely used in the robotics community for tracking applications. We evaluate the tags in terms of range, viewing angle, and robustness to parameters like hand shake and lighting. We chose AprilTag as a point of comparison because it is widely used and representative of passive tags as a whole. We have not found a competing active tag implementation that is computationally efficient enough to be detected in a Web application, so we compare only to passive tags in this evaluation.

4.1 Experimental Setup

Unless otherwise specified, for all of our comparisons, we evaluated FLASH using a 0.15m by 0.15m tag using an 8-bit rolling code displayed on an iPad. For our passive tag comparisons, we used the same size 0.15m by 0.15m 36h11 AprilTag of ID 0 printed on a piece of paper. All data was collected using a FLASH and an AprilTag web framework running on an iPad Pro with a resolution of 720p mounted on a tripod or held by hand.

We also performed a test in a large stadium using a 0.70m by 0.70m FLASH tag running in a web page on an LCD monitor. This experimental setup is summarized in Figure 10. Even in bright daylight, the tag was able to be detected from across the stadium at a distance of over 100m.

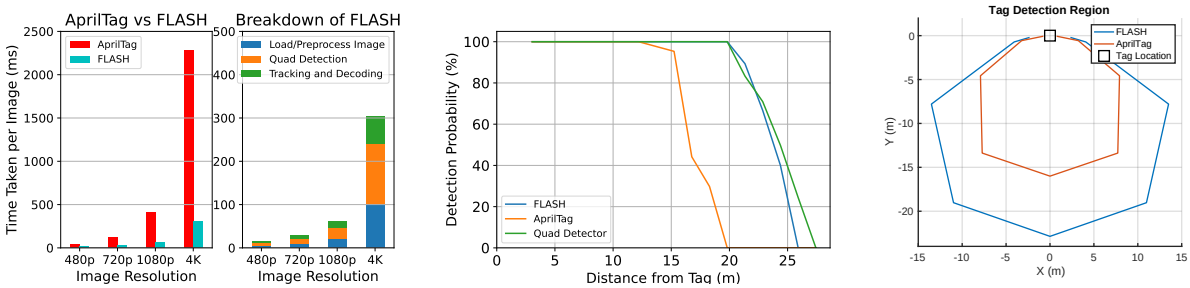

Figure 8: Left: performance comparison of AprilTag versus FLASH. FLASH is around 10 times faster than AprilTag. Right: zoom-in view of performance breakdown for FLASH. In our implementation for FLASH, quad detection and decoding are done in a separate background thread in parallel with loading and pre-processing the image.

4.2 Processing Pipeline Benchmark

Figure 8 shows the performance of a web port of the AprilTag processing pipeline and FLASH at four different resolutions using a 2020 iPad Pro. FLASH is around 10 times faster than AprilTag. Owing to the fact that our active tags provide better contrast, our quad detector outputs significantly fewer false positives, which allows for faster tracking. Decoding in FLASH is simply thresholding a list of intensity values, converting those values into a series of bits, and comparing those bits with a number. To achieve a 30 FPS camera capture rate, the total time taken per frame of FLASH needs to have at most 33.333ms latency, which can be achieved at 480p and 720p. The right side of Figure 8 shows a zoom-in on the performance of FLASH alone, where we see a relatively even distribution of the computation time between the several processing steps.

4.3 Detection Probability Over Distance

To estimate the expected range of our tags, we collected data in the form of five-second video clips at varying distances. We then compute detection probability as the percentage of video frames in which the system being tested detected the tag with the proper ID. Figure 9 shows the detection probability at various distances of three different systems: FLASH, AprilTag, and the quad detector used by FLASH. We included the quad detector because we wanted to see if FLASH was reaching the limits of its own quad detector. The figure shows that AprilTag's detection probability falls down from 100% at around 15m, while FLASH and its quad detector fall at around 20m. AprilTag can no longer detect tags at 20m, whereas FLASH and its quad detector can no longer detect at around 27m. This gives an approximately 25-35% increase in detection range of FLASH over AprilTag. Note that since FLASH does not need a correctly decoded quad in every frame, it is often possible to resolve tags correctly even if the detection probability is dropping off.

4.4 Tag Angle

Similar to how FLASH tags are better in terms of distance, they also allow detection at more acute angles where the spatial content within a passive tag is lost due to perspective and resolution. To test the detection rate of FLASH when encountering an off-axis tag, we measured the detection probability of both systems with tags at various angles at a fixed distance of 12.192m (40ft). We chose this distance since both FLASH tags and AprilTags have 100% detection accuracy at this distance (see figure 8). We found that AprilTag's detection rate falls to 0% when attempting to detect an off-axis tag with an angle of around 35°, while FLASH's rate falls to 0% with an off-axis tag angle of around 60°. Since AprilTag IDs are encoded spatially, if a tag has a high enough off-axis angle, important details of the tag ID will be hidden from the camera at a distance, making the tag impossible to decode. On the other hand, FLASH does not have this problem, as there are no details on the FLASH tag itself that are related to the tag ID. All FLASH needs to decode a FLASH tag is the tag's location across multiple frames. Figure 8 shows the estimated difference in coverage when viewing FLASH tags compared to AprilTags for 0.15m tag sizes.

4.5 Lighting

To test the robustness of FLASH to various lighting conditions, we collected detection probabilities in three different types of lighting environments at a distance of 12.192m. These environments had different ambient lighting conditions: a "bright" environment corresponds to an average image intensity of 121, a "medium" environment corresponds to an average image intensity of 43, and a "dark" environment corresponds to an average image intensity of 1. See Figure 9 for an example of some of the lighting conditions we tested in. Figure 9 shows that the detection probability of FLASH remains relatively the same regardless of lighting, whereas that of AprilTag dramatically decreases as the environment darkens. Not surprisingly, due to the illumination of an active tag on a display, FLASH tags are more resilient to darker lighting conditions than AprilTags, which are printed on paper. One alternative would be to backlight an AprilTag (which is often done in robotics) making the AprilTag performance under different lighting behave similarly to FLASH. However, the range and hand shake issues would still apply, limiting AprilTag's range.

4.6 Camera Shake

To test the effects of real-life hand shake when holding a device, we collected IMU data from the phone while it was collecting data when being held by hand. Like the previous experiments, we ran these tests at a distance of 12.192m. A shake intensity categorized as "low" is done by holding the device in the hand as still as possible. A "medium" shake intensity would encompass trace shaking up to a maximum absolute acceleration of 0.28ms^{-2} according to the phone's accelerometer. Lastly, a shake intensity categorized as "high" corresponds to a maximum absolute acceleration of 0.54ms^{-2}. Again, due to the fact that FLASH does not need to decode intricate spatial details embedded in the tag itself, it is more invariant to motion blur than a system like AprilTag, as shown in Figure 9. Observe that FLASH has a higher detection accuracy than AprilTag under medium and large motion blur.

4.7 Quad Size

Finally, we wanted to provide a tag size guideline based on the required range for an application. We displayed several quads of sizes ranging from 0.05m by 0.05m to 0.70m by 0.70m to determine the relationship between quad size and detection distance. Figure 9 shows that there is an expected mostly linear relationship between the two. Figure 10 shows an experimental validation using a 0.70m by 0.70m tag that we were able to detect at slightly over 100m in a stadium on campus.

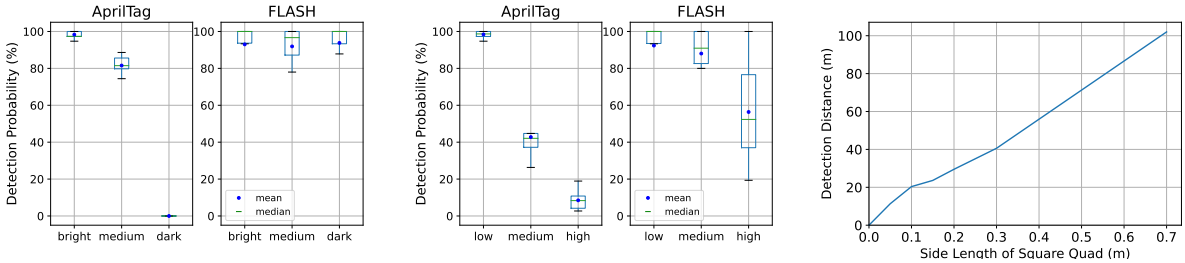

Figure 9: Detection probability of AprilTag versus FLASH in various lighting conditions (left), with various camera/hand shake intensities (middle). Detection distances of various quad sizes (right).

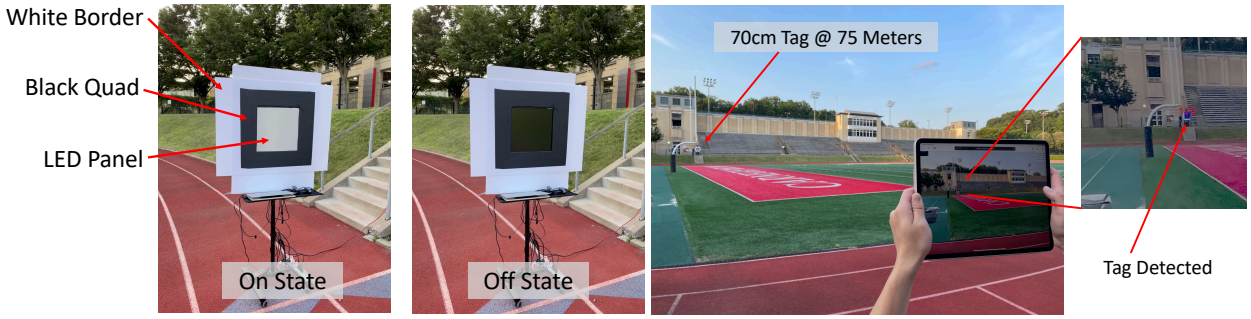

Figure 10: 70cm by 70cm outdoor tag in stadium detected at 75 meters (max detection at ≈102 meters).

5 Limitations

While extremely promising for entertainment applications, this approach does have several limitations and drawbacks. First, the tags flash at a visibly perceivable rate. Having collaborated closely with drama production experts during this project, they assure us that consistently flashing targets tend to fade into the background in complex productions. In fact, they had initially used a large, illuminated image target for crowd AR effects, but found it to be extremely distracting and hard to integrate into the storyline. Minimizing flashing was part of our motivation to make sure that our codes were rolling instead of using a more traditional pause followed by a preamble and then data design. The tags need to have more contrast than the background to be reliably detected, but they do not need to be visually bright. One practical solution for stage environments (often used in commercial displays already) is to adjust the brightness of the tag so that when the lights are dimmed down the screen responds accordingly. As phone cameras operate at faster rates and/or if they have the option to remove infrared filters, one could imagine the tags becoming flicker-free and nearly invisible.

Second, our tags need enough resolution on large displays to create a reasonably well-defined quad shape for tracking. In general, we find that the quad is extremely forgiving in terms of detection (often, high contrast round objects are initially detected as candidate tags), but that if the corners are rounded, the system will suffer dramatically in terms of pose estimation performance. For scenarios where there is no video display available, we have used LED lighting panels like the FLOALT series from IKEA [28] to create a standalone light. The 24 in (0.6m) square FLOALT light costs $129USD and with a quad constructed from black electrical tape is detectable from about 100m away. The light has a simple control interface with an external Zigbee board that can be removed to expose a header with 3.3v, ground, PWM1, and PWM2 headers. One can trivially connect an MCU to this header to generate a simple, dimmable, and low-cost standalone active optical anchor that can adapt to ambient lighting conditions (with PWM control).

Third, our Web implementation is still relatively limited in terms of frame rate. With a native application, we could likely create a decoder that was able to operate on blinking rates that were not humanly visible. This would require both a native application and a custom display, since most large monitors do not provide high enough refresh rates to avoid flicker (or at least stroboscopic flickering with head motion). Even if a native app could process current passive markers faster and at a higher resolution, it would then also be able to detect FLASH tags from comparatively farther. Finally, unfortunately, most release versions of Web browsers do not yet support computer vision processing in a WebXR session, so tracking is limited to just IMU-based rotation instead of full 6DOF tracking. We expect this to change over the coming years.

Finally, FLASH's gravity-assisted solver requires that tags are mounted on vertical surfaces. If a tag is placed on the floor or ceiling, it will suffer from multiple ambiguous solutions. It would be possible to add a corner marker or other such feature to explicitly define rotation (like other passive tags), but this would reduce range since any small features get lost at lower pixel densities.

6 Conclusions

We present FLASH, an active light anchor specifically designed to be embedded in video displays to provide mobile AR anchoring in entertainment environments. Using active blinking codes combined with static image structure framing offers a practical and straightforward solution to camera pose estimation that outperforms static tags in range and viewing angles. The addition of structured visual features (a quad) allows the system to be computationally lightweight enough to execute at 30 FPS on 720p images within a browser on a modern mobile phone. While not specific to just our tag, we also provide an analysis and reference implementation of a gravity assisted camera pose estimation solver that avoids common ambiguity problems that plague current open-source systems.

We believe we can further optimize the coding scheme using something similar to Knuth codes to enable lower-latency detection as future work. We also think there could be alternative static marker geometries (other than quads) that might increase range or pose estimation accuracy.

ACKNOWLEDGMENTS

This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] K. Ahuja, S. Pareddy, R. Xiao, M. Goel, and C. Harrison. Lightanchors: Appropriating point lights for spatially-anchored augmented reality interfaces. In *UIST*. ACM, 2019.

[2] Apple. Arkit. https://developer.apple.com/arkit/, 2019.

[3] A. Ashok, M. Gruteser, N. Mandayam, J. Silva, M. Varga, and K. Dana. Challenge: Mobile optical networks through visual mimo. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, p. 105–112. Association for Computing Machinery, New York, NY, USA, 2010. doi: 10.1145/1859995.1860008

[4] D. Bryant. Webassembly outside the browser: A new foundation for pervasive computing. *Keynote at ICWE'20, June 9–12, 2020, Helsinki, Finland*, 2020.

[5] D. Callahan. Experimenting with WebAssembly and Computer Vision. https://hacks.mozilla.org/2017/09/bootcamps-webassembly-and-computer-vision/. Online. Accessed: May 2021.

[6] A. Censi, J. Strubel, C. Brandli, T. Delbruck, and D. Scaramuzza. Low-latency localization by active led markers tracking using a dynamic vision sensor. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 891–898, 2013. doi: 10.1109/IROS.2013.6696456

[7] T. Collins and A. Bartoli. Infinitesimal plane-based pose estimation. *International Journal of Computer Vision*, 109(3):252–286, 2014. doi: 10.1007/s11263-014-0725-5

[8] DecaWave. Decawave. https://www.decawave.com, 2019.

[9] M. Fiala. Artag: Fiducial marker system using digital techniques. In *CVPR*, 2005.

[10] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[11] Google. Arcore. https://developers.google.com/ar/, 2019.

[12] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. *RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments*, pp. 477–491. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi: 10.1007/978-3-642-28572-1_33

[13] HTC. Htc vive. https://www.vive.com/us/, 2019.

[14] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pp. 85–94, 1999. doi: 10.1109/IWAR.1999.803809

[15] Y.-S. Kuo, P. Pannuto, K.-J. Hsiao, and P. Dutta. Luxapose: Indoor positioning with mobile phones and visible light. In *MOBICOM*. ACM, 2014.

[16] M. Leap. Magic leap. https://www.magicleap.com/, 2019.

[17] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o(n) solution to the pnp problem. *Int. J. Comput. Vision*, 81(2):155–166, Feb. 2009. doi: 10.1007/s11263-008-0152-6

[18] C.-P. Lu, G. D. Hager, and E. Mjolsness. Fast and globally convergent pose estimation from video images. *IEEE transactions on pattern analysis and machine intelligence*, 22(6):610–622, 2000.

[19] Microsoft. Hololens. https://www.microsoft.com/en-us/hololens/, 2019.

[20] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015. doi: 10.1109/tro.2015.2463671

[21] Oculus. Rift website, Oct 2020. Online. Accessed: 2020-10-20.

[22] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *ICRA*. IEEE, 2011.

[23] A. org. AR.js - Augmented Reality on the Web. https://github.com/AR-js-org/AR.js. Online. Accessed: May 2021.

[24] N. Rajagopal, P. Lazik, and A. Rowe. Visual light landmarks for mobile devices. In *IPSN*. IEEE, 2014.

[25] G. Reitmayr and T. W. Drummond. Going out: robust model-based tracking for outdoor augmented reality. In *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality*, pp. 109–118, 2006. doi: 10.1109/ISMAR.2006.297801

[26] Can I Use. Stream API Support. https://caniuse.com/stream. Online. Accessed: May 2021.

[27] Can I Use. WebXR Support. https://caniuse.com/webxr. Online. Accessed: May 2021.

[28] Ikea. Floalt led light panel. https://www.ikea.com/us/en/p/floalt-led-light-panel-dimmable-white-spectrum-40303076/. Online. Accessed: May 2021.

[29] Immersive Web WG. Webxr polyfill. https://github.com/immersive-web/webxr-polyfill. Online. Accessed: May 2021.

[30] Immersive Web WG. WebXR Raw Camera Access. https://github.com/immersive-web/raw-camera-access/blob/main/explainer.md. Online. Accessed: May 2021.

[31] WASM WG. Webassembly overview. https://webassembly.org/. Online. Accessed: May 2021.

[32] April Robotics Laboratory, U. of Michigan. AprilTag 3 decoder implementation (C code). https://github.com/AprilRobotics/apriltag. Online. Accessed: May 2021.

[33] R. A. Sharma, A. Dongare, J. Miller, N. Wilkerson, D. Cohen, V. Sekar, P. Dutta, and A. Rowe. All that glitters: Low-power spoof-resilient optical markers for augmented reality. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 289–300, 2020. doi: 10.1109/IPSN48710.2020.00-27

[34] G. Terzakis and M. Lourakis. A consistently fast and globally optimal solution to the perspective-n-point problem. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds., *Computer Vision – ECCV 2020*, pp. 478–494. Springer International Publishing, Cham, 2020.

[35] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[36] M. Vision. Meta2. https://www.metavision.com/, 2019.

[37] W3C. webxr device api. https://immersive-web.github.io/webxr/. Online. Accessed: May 2021.

[38] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *ISMAR*. IEEE Computer Society, 2008.

[39] G. Woo, A. Lippman, and R. Raskar. Vrcodes: Unobtrusive and active visual codes for interaction by exploiting rolling shutter. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 59–64, 2012. doi: 10.1109/ISMAR.2012.6402539

[40] C. Xiao and Z. Lifeng. Implementation of mobile augmented reality based on vuforia and rawajali. In *ICSESS*. IEEE, 2014.

[41] J. J. Yang and J. A. Landay. Infoled: Augmenting led indicator lights for device positioning and communication. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, p. 175–187. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3332165.3347954