Aristo: An Augmented Reality Platform for Immersion and Interactivity

Zhongyang Zheng, Bo Wang, Yakun Wang, Shuang Yang, Zhongqian Dong
Tianyang Yi, Cyrus Choi, Emily J. Chang, Edward Y. Chang
eyuchang@gmail.com
HTC Research, Beijing, China & San Francisco, USA

ABSTRACT

This paper introduces our augmented reality platform, Aristo, which aims to provide users with physical feedback when interacting with virtual objects. We use Vivepaper, a product we launched on Aristo in 2016, to illustrate the platform's performance requirements and key algorithms. We specifically depict Vivepaper's tracking and gesture recognition algorithms, which involve several trade-offs between speed and accuracy to achieve an immersive experience.

CCS CONCEPTS

• **Human-centered computing → Mixed / augmented reality**;
• **Computing methodologies → Mixed / augmented reality**;

KEYWORDS

Aristo; Vivepaper; augmented reality; virtual reality;

1 INTRODUCTION

Aristo is our *augmented reality* platform, which allows users to interact with virtual multimedia content rendered on physical objects. For instance, a virtual book can be rendered on a piece of card stock to allow a user to select and browse content. Virtual parts rendered on cuboids can be overlaid on physical equipment (e.g., a wind turbine or an engine) to train repair workers both safely and cost effectively. In general, a physical object is turned into or overlaid with a virtual object when viewed through a head-mounted display (HMD) for interactivity.

The Aristo platform consists of three main components: an HMD with a frontal camera, physical objects printed with visual fiducial markers (e.g., Figure 1(a) shows a piece of card stock as a physical object), and an HCI module. A user sees virtual objects on the HMD rendered upon physical objects (e.g., Figure 1(b) shows a virtual book rendered on the physical card stock). A user can interact with virtual objects through natural hand gestures and eye positioning, while physical objects provide feedback via grip function and tactile perception.

We have implemented two applications on the Aristo platform. The first application, *Vivepaper*, is a virtual book for immersive browsing and reading. The second is a virtual workshop for training novice workers. In this paper we focus on the Vivepaper application. The configuration/algorithmic differences between supporting reading and training are discussed in Section 5.

The implementation of Aristo in general, and Vivepaper specifically, faces technical challenges in three areas, which our contributions aim to address:

- *Marker tracking.* Vivepaper must track the fiducial markers on the card stock both accurately and timely to perform pose (position and orientation) estimation. We propose a marker-grid scheme with carefully tuned configurations to achieve both high tracking accuracy and speed.
- *Gesture recognition.* A user interacts with Vivepaper using hand gestures. We take advantage of the geometry of the card stock and its markers to achieve accurate hand segmentation and hence effective gesture recognition.
- *Real-time processing.* Real-time tracking is an essential requirement for any augmented reality system. Our algorithms can run comfortably above 60fps on both desktop and mobile platforms, where a GPU is available. Note that although some markerless tracking and gesture recognition schemes have been proposed (including our own work [9]), their high latency (a less than 60fps frame rate) means they are not yet production ready on current mainstream PCs and mobile devices.

Leveraging a frontal camera, we have deployed Vivepaper with VIVE on PC, Daydream on Android, and Cardboard on iOS to enable an augmented reality experience. Additionally, we have created a content generation platform, which allows third parties to compose and upload books for users to explore. Vivepaper's initial partners include the vertical domains of education, training and tourism.

The remainder of this paper is organized as follows: Section 2 details related work in computer vision and articulates the constraints Vivepaper faces and trade-offs it makes. Section 3 depicts our marker-based tracking algorithm. Section 4 presents an effective gesture recognition algorithm. We show our empirical study, which facilitates the selection of several configuration parameters to improve accuracy in both object tracking and gesture recognition. Section 5 discusses the additional configuration/algorithmic considerations required to support assembly line training, and offers our concluding remarks.

2 RELATED WORKS

Several key components of VR/AR systems are borrowed from the computer vision community. We divide and discuss related computer vision research in two areas: marker tracking and hand segmentation.

MM '17, October 23–27, 2017, Mountain View, CA, USA

ACM ISBN 978-1-4503-4906-2/17/10…$15.00
https://doi.org/10.1145/3123266.3123308

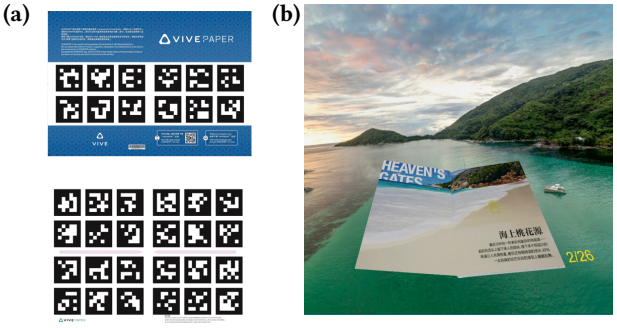


Figure 1: (a) Vivepaper's card stock with markers (top: back/front cover, bottom: two inner pages). (b) Augmented reality content displayed on card stock.

2.1 Marker Tracking

Due to the difficulties in tracking general objects, using markers is a compromise in AR for indirectly tracking objects. Markers can be either passive or active, and they can be tracked by traditional cameras, infrared cameras, or lasers. Vivepaper uses passive markers, and they are tracked by HMD cameras.

One of the earliest and most popular visual fiducial systems is ARToolKit [21], which uses a square tag containing a wide black border with an image pattern inside the square. The four corners of the square are used to estimate the pose of the square, while the inner pattern is matched against a database of valid patterns for identification. However, ARToolKit's template matching method can suffer from high false positives [14]. Many later square-based fiducial systems use binary codes to remedy the problem of inter-tag confusion. ARTag [13] is one example of a system that uses a binary coding scheme to correct bit errors and improve detection robustness. Other similar systems include ARToolkit Plus [40] and Studierstube Tracker [41].

Apriltag [29] optimizes the previous approaches by guaranteeing a minimum Hamming distance between a tag and its four rotations. However, the algorithm for detecting quads (each quad represents the black border around a tag candidate) is computationally intensive for deriving image gradients, fitting segments, building lines, and searching quads. Our experimental result (see Table 2) shows that the frame rate is at 23fps on average running with a modern CPU, and thus not suitable for our real-time detection requirement. The detector has a lower false negative rate, but a relatively higher false positive rate due to a large number of quad-candidates. (Figure 5(a) in Section 3.3 compares various schemes.)

ArUco [15] is another popular open source library used to track visual fiducial markers. The authors focus on configurable marker generation and occlusion resolution. For marker detection, they use a contour-based method to speed up quad detection. Compared to the edge-based approach used in Apriltag [29], this method may lead to location error of quad corners, causing virtual book jitter. Our method is based on ArUco, and we make improvements to reduce the aforementioned error (details in Section 3).

In addition to widely used square tags, other tag encoding schemes exist. RectTIVision [5] is based on blob detection introduced by d-touch [12]. RUNE Tags [6] use circular dot patterns to make up markers, but do not provide enough correspondence points to perform pose estimation. Since Vivepaper relies on accurate pose estimation, employing square markers is our logical choice.

While direct object tracking and markerless tracking may be desirable, these are still very difficult tasks to accomplish in real-time. Researchers address accuracy and speed issues [27] by providing 3D models of the tracked objects [37] or by relaxing to planar scenes [22]. The work by Comport et al. [11] uses prior model structures of the targeted objects as search candidates. Digilog books [22] uses SIFT feature matching followed by a RANSAC based homography estimation to track planar targets. The work by Grinchuk et al. [16] proposes a deep learning approach to generate and decode visual markers, which are color images with artistic stylization. The appearance of these methods is more "human-friendly" than regular visual markers but suffers from being extremely highly computational intensive [16, 37]. Additionally, unavoidable object recognition errors can decrease tracking robustness [11, 22]. The Aristo platform is designed to work with any object without prior model information, and must support highly reliable and efficient tracking. Therefore, the marker-based approach is our current choice.

2.2 Hand Segmentation

Hand segmentation is the first step of our gesture recognition algorithm. Hand segmentation methods have been widely proposed in literature. Different methods utilize different types of visual features and, in some cases, a combination of them [30] to improve segmentation accuracy. Some popular features are skin color, textures, shapes, motion, and anatomical models of the hand. The work by Li and Kitani [23] examines a pool of widely used local appearance features and global appearance features. The authors propose that a sparse set of features achieves the best performance, and global appearance models are useful for adapting changes in illumination of hand. Baraldi et al. [4] adopts superpixel, Gabor filters, and histograms to perform hand segmentation and utilize a collection of random forests to deal with different illumination conditions. Zhu et al. [48] further introduce a shape mask with structured forests to better utilize shape information. Betancourt et al. [7] and [8] propose a GPU-accelerated implementation and formalize a multi-model hand-segmenter based on random forest and K^{th}-nearest neighbors. These random forest-based methods achieve state-of-the-art performance, but they require pixel-level hand annotations for training, which requires a large amount of labeled data.

Recently, deep convolutional neural networks (CNNs) have attracted a lot of attention. Zhou et al. [47] propose a EM-like learning framework trained iteratively with weakly supervised hand bounding boxes. Ma et al. [25] train a pixel-to-pixel hand segmentation network using raw images and binary hand masks to produce a hand probability map. Vodopivec et al. [39] extract features with convolutional layers and maps them directly to a segmentation mask with a fully connected layer. Our own proposed CNN-based image alignment scheme [9] achieves superior performance compared to several state-of-the-art algorithms. Although CNN-based methods perform well, a trained model is typically very large, and most mobile devices and wearables are resource-strained to compute fast enough to fulfill the real-time requirement of hand segmentation.

Input images for hand segmentation are not limited to RGB images. With the introduction of commodity depth cameras (e.g. Kinect), researchers can make use of the depth information. Sinha et al. [34] perform a large blob detection in a depth range with median filter and depth normalization as post-processing. Kang et al. [20] propose to use depth information with random forest, and add bilateral filtering and decision adjustment for hand segmentation. Liang et al. [24] train a random regression forest using depth context features of randomly sampled pixels and fuse the information from both color and depth. In the commercial field, the Leap Motion [1] controller uses two monochromatic IR cameras and three infrared LEDs to capture hand and finger motions. By using depth sensors, the problem of changes in illumination can be partially alleviated. However, for the current generation of depth sensors, the depth information still contains substantial noise.

Instead of learning features from the foreground, there are methods focusing on dealing with special backgrounds. In the Visual Panel system, Zhang et al. [45] take an arbitrary quadrangle-shaped panel as a background and use dynamic background subtraction to segment hands out. Garrido-Jurado et al. [15] model the background of markers by a mixture of two Gaussian and compute an occlusion mask by color segmentation. Ha et al. [17] and Malik et al. [26] make use of a black and white background and calculate a histogram for setting the threshold of segmentation. In our work, we also take advantage of a controlled background to yield higher accuracy and faster speed.

Two design constraints compel us to use a rule-based algorithm for Vivepaper instead of deep learning: compatibility with most modern HMDs and support of real-time performance. Since most HMDs are only equipped with RGB cameras, we are limited to supporting traditional RGB images. Due to the real-time performance requirement to support a 60fps frame rate, deep learning based methods cannot be employed (although recent research has revealed software and hardware methods for accelerating in the classification stage, they are at least a year away from being deployed with commercial HMDs). Moving objects (both the HMD and card stock) and variant features (due to illumination conditions, orientations, and occlusion), compounded with the 60fps requirement, narrow our choice algorithm to a simple and effective color-based scheme. We take advantage of the geometry of Vivepaper's card stock and the marker matrix on it to fulfill both the accuracy and speed requirements of Vivepaper.

Some traditional augmented reading systems (e.g., [10, 32, 38]) may require special physical edition for each book, with AR markers embedded on select pages. Instead of AR markers, a markerless approach tracks images in a book (e.g., [18, 28, 33, 36]). This approach requires designing a book with distinct images to avoid mismatch, and a time-consuming training model for each book. Vivepaper uses a generic card stock, which can render any book cover-to-cover, hence offering flexibility and convenience for content selection, delivery, and updates.

3 MARKER TRACKING

Vivepaper demands fast and accurate tracking of the card stock's position and orientation to render selected book content accurately and timely on the HMD. However, both of the common open-source tracker algorithms we evaluated (Apriltag [29] and ArUco [15]) fail to fulfill our following requirements:

- High decoding speed. A hand-held piece of card stock can be freely moved. In order to track this movement accurately and timely, decoding speed must be above 60fps. Our experiment (Section 3.3.3) showed the speed of AprilTag is 23fps.
- Low jitter error. Several factors such as signal noise and low camera resolution can cause object tracking errors. Our experiment (Section 3.3.2) showed that ArUco suffers from high jitter error.

Our proposed pipeline addresses not only the above two problems, but also the issue of occlusion. In the remainder of this section, we first depict the steps of marker tracking and highlight our novelty (Section 3.1). We then introduce the idea of a *marker grid* and present an algorithm to cluster markers (Section 3.2). We report our experiments on simulated data, which help us tune/set important parameters to improve tracking accuracy and speed (Section 3.3). We also report our evaluations in real environments on all trackers we studied to demonstrate that Vivepaper employs the best tracker to achieve a balance between accuracy and frame rate.

3.1 Tracking Process

The tracking process consists of three steps: quadrilateral (quad) detection, marker decoding, and pose estimation. Each step involves carefully setting parameters to achieve a good balance between speed and accuracy for smooth content presentation.

3.1.1 Quad Detection. Vivepaper's card stock is printed with $x \times y$ fiducial markers (see Figure 2). As discussed in Section 2, because we use square markers, we refer these markers as quadrilaterals or quads. Detecting these quads is the most time-consuming step among the three steps. Detection errors of the corners of the quads will be propagated to the next two steps. We analyze the substeps of the quad detector as follows:

- Pre-processing. To achieve a better detection rate of the quads under varying illumination, for example in a slightly darker environment, we perform a white balance adjustment on each captured image. Then, we convert the image to grayscale and blur it using a Gaussian filter. Lastly, adaptive thresholding is performed to create good contrast between areas of different illumination conditions. We use functions in CUDA, OpenCL, and FastCV libraries to speed up this pre-processing step.
- Quad extraction. Next, contours are extracted using the algorithm proposed by Suzuki et al [35]. Then, we find the convex hulls of the contours and conduct polygonal approximation to obtain 4-vertex quads. (This substep was also employed in ArUco [15].)
- Corner refinement. Since the previous two steps often introduce quad corner location errors, we must make the quads more reliable. We recompute the four corners of each quad to help improve reliability by fitting new lines along the original lines of the quads. This idea draws on the experience of Apriltag2 [42], an improved version of the original Apriltag [29]. First, we sample some points along each line, and then adjust the points to the largest gradient of the original image. The new lines are fitted to these new points using the least-squares algorithm and the new corners are the intersections of the four new lines. The improved performance can be seen from the experimental result in Figure 5(b).

3.1.2 Marker Decoding. The marker decoding step is similar to that in Apriltag and ArUco. First the quads are transformed perspectively. Then we use a spatially-varying threshold to decide the value of the corresponding bit as stated in Apriltag [29], which is robust to illumination. After getting the binary code of the quad, we rearrange the code to its four rotations. Then, we iterate through all the valid codes and use a Hamming distance hash table to look up the final ID. It should be noted that sometimes there are two quads detected with the same ID. In this case we only keep the one with the smaller Hamming distance or larger perimeter. To speed up this process, since the decoding process of a quad is independent of the others, we use OpenMP to parallelize computation.

3.1.3 Pose Estimation. Once a marker has been detected, with known camera intrinsic parameters (focal length and lens type) and the physical size of the marker, we can use the principles of trigonometry to compute the position and orientation of the markers with respect to the camera.

Contrary to most augmented display systems where only the relative position between camera and target object is available, Vivepaper running on VIVE can obtain the world coordinate of the virtual book using the precise camera and HMD pose. Even when tracking occasionally fails due to variations like rapid head movement and card stock moving out of sight, the virtual book can still be rendered by using the last known parameters.

3.2 Marker Grid

Generally speaking, the larger the size of a marker, the easier it can be tracked. However, since Vivepaper supports gesture recognition above these markers, a large marker can easily be partially occluded. Moreover, since the camera on a typical HMD has a smaller field of view compared to that of human eyes, when the card stock is brought near the camera, some markers on the borders can be outside of the field of view. Although using smaller markers can avoid all markers being simultaneously occluded, this approach reduces tracking accuracy because smaller markers are more susceptible to camera noise and environmental variations.

In Vivepaper, we use a matrix grid configuration (see Figure 2) to address the above large versus small marker dilemma. Our card stock consists of two A4 (21cm × 29cm) folds (two pages). On this available space, the question is then what is the required number and size of the markers to achieve a targeted accuracy, given a range of maker-to-camera distances and possible hand occlusion. Section 3.3.1 reports our experiments and findings.

3.2.1 Marker Grid Pose Estimation. Instead of simply averaging the poses of all the detected markers on a grid, the pose of the marker grid is determined as follows. First, we cluster the detected markers into groups based on their positions. Then, to ensure the reliability of the final pose, we select a cluster that contains at least half of the detected markers. Lastly, the positions of markers in the selected cluster are averaged to estimate the virtual book's position. The position of the virtual book in the previous frame is used if no candidate cluster is available. The same procedure is applied to estimate the orientation of the virtual book. With the calculated position and orientation, the HMD renders the content of the virtual book on its display. Experiments in Section 3.3.1 show the effectiveness of this method.

To mitigate motion jumpiness, we perform a Kalman filter smoothing on the pose of the virtual book. Such smoothing may make a real, swift movement of the card stock move slower. However, since swift movements are rare, the overall user experience is enhanced by making pose changes smooth.

3.2.2 Marker Grid Tracking Parameters. Table 1 lists important parameters affecting tracking accuracy. As the value range of a parameter may have physical limitations, and tradeoffs exist between parameters (e.g., a lower resolution image requires larger markers to achieve the same level of accuracy), we conduct an empirical study to examine how these parameters affect tracking accuracy.

Table 1: List of parameters affecting tracking accuracy.

Category	Parameters	Note
Marker	Size of card stock	A4 (21cm × 29cm), standard printing size.
	Size and Number of marker	Marker-grid (see Figure 2) is used to minimize the effect of hands occlusion.
	Distance and orientation	Distance between camera and hand-held card stock is usually $0.3m$ to $0.6m$.
	Marker type	We generated an Apriltag family with 36 unique ids and min. hamming distance 9.
Camera	Camera resolution	For PC, with USB 2.0 camera (60MBps) in 60fps, the max. resolution in uncompressed RGB channel is 612 × 460.
	Noise of camera	
External	Illumination environments	Color of lighting (daylight, or fluorescent in white or yellow color), intensity (smooth, too strong, or too weak), reflection of card stock.
	Occlusion	By hands
	Tracking algorithm	See section 3.3.2 for comparison result with Apriltag [29] and ArUco [15].

3.2.3 Marker Grid Real-time Tracking. Besides accuracy, real-time decoding speed is an important factor in our system, since computer-generated objects must synchronize with physical objects on the display. In Vivepaper, the real-time tracking of the card stock's position and orientation is achieved by our carefully tuned marker decoding pipeline configuration. We additionally reimplemented some steps of the algorithm using CUDA/OpenCL/FastCV to speed up the detection process. The performance comparison is presented in Section 3.3.3.

3.3 Experiments

This section reports impacts of parameter choices on tracking performance via empirical studies. As previewed in the previous sections, the performance factors we examined are:

- The effect of marker size and number on tracking accuracy (section 3.3.1),
- The effect of choice of tracking algorithm on accuracy (section 3.3.2), and
- Validation of processing speed (section 3.3.3).

We used Apriltag's open source code [29] and the ArUco module in OpenCV version 3.2.0 [2] to conduct empirical studies.

3.3.1 Evaluation via Simulation. In this experiment, given the constraint of a fixed size of card stock (A4 page 21cm × 29.7cm), we studied the trade-off between the number of markers and accuracy of the algorithm under the combinations of different distances,

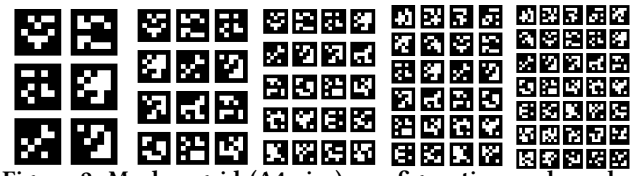


Figure 2: Marker grid (A4 size), configuration and marker length, from left to right, 3x2 8.05cm, 4x3 5.67cm, 5x4 4.27cm, 6x4 4.06cm, 7x5 3.43cm.

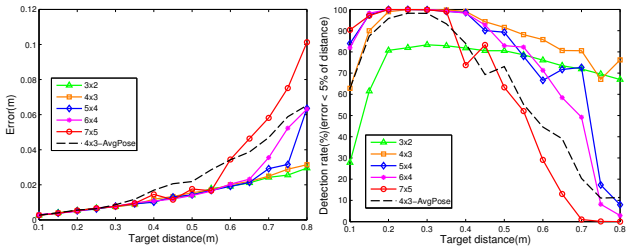
occluded positions, lighting conditions, and orientations. Figure 2 shows the five marker-grid configurations we studied.

The experiment was set up as follows: A simulator written in Unity3D [3] simulated what was captured by the frontal camera with varying card-stock and hand positions, light conditions, and orientations using five different configurations of the marker grid. The simulated camera had the same intrinsic parameters as our frontal camera (612×460 resolution and focal length 289.6 pixel).

Distance Study

To study the effect of different marker-grid configurations with different distances, the simulator placed the card stock directly facing the camera (off-axis 0 deg), and then moved away from the camera optics axis from 0.1m to 0.8m with a step size of 0.05m (15 positions). For each position, we placed a virtual hand (the width of palm 0.14m) up-right without rotation at 0.05m on top of the card stock. We then moved the hand from the top of the card stock to its bottom at 6 equidistant positions. At each of these positions, we also moved the hand from left to right at 6 equidistant positions, creating 36 distinct positions. For each of the 36 hand positions, a spotlight, placed on top of the card stock $z = 0.15$m away, was shined on at a random position on the card stock whose aperture angle (the width of light) was randomly set between 50 to 100 deg. The spotlight procedure was repeated 37 times for each hand position. The total number of simulated settings yielded $99,900$ images ($5 \times 15 \times 36 \times 37$).

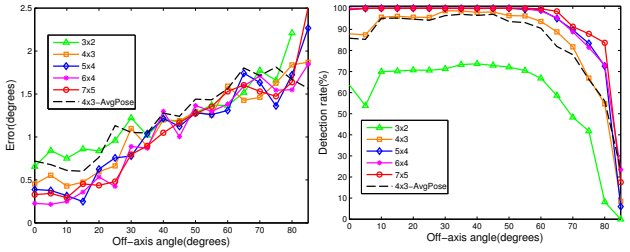
The images were loaded directly to our developed tracker. As the ground-truth of the marker position was known, we computed the average error for different marker-grid configurations for each distance.



(a) (b)

Figure 3: Distance accuracy. (a) Position error vs. distance (b) Detection rate vs. distance.

Figure 3 reports the localization accuracy and detection rate (y-axis) of the five marker-grid configurations with respect to distance (x-axis). Figure 3(a) reports error distance, the gap between the ground truth position and the detected position (if the detection is deemed successful), on the y-axis. As expected, the error increases as the card stock moves farther away from the camera. The 3×2 and 4×3 grids achieve the best accuracy. While Figure 3(a) shows the error of the successfully detected cases, Figure 3(b) shows the successful detection rate, which we define as the ratio of the error divided by the distance (to the camera), to be within 5%. A detection failure could result from three factors:



(a) (b)

Figure 4: Orientation accuracy. (a) Orientation error vs. off-axis angle (b) Detection rate vs. off-axis angle.

(1) *Hand occlusion.* The 3×2 grid suffers from the worst detection rate due to that large markers can easily be occluded by hand.
(2) *Limited field of view.* When the card stock is brought near the camera (e.g., less than $0.2m$), larger markers are more susceptible to be out of sight.
(3) *Distance away from camera.* As grid-to-camera distance increases, detection rate decreases. Markers of smaller size (5×4, 6×4 and 7×5) are more susceptible to longer distance.

Orientation Study

To further study the effect of the orientation, we repeated the same procedure with a fixed target distance (0.3m), and varied off-axis angle, 0 to 85 deg, with a step size of 5 deg.

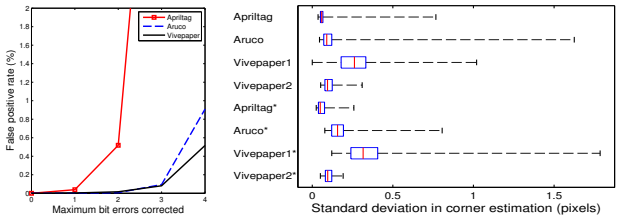
Figure 8 reports the localization accuracy and detection rate (y-axis) of the five marker-grid configurations with respect to orientation (x-axis). The error rate grows as the marker-grid rotates away. In terms of detection rate, the 2×3 grid suffers the worst performance, while the others are more or less the same.

In both experiments, we also compared our maker-grid approach with the average performance of tracking individual markers (the black dash line, denoted as AvgPose in both figures). Both figures show that Vivepaper's marker-grid approach enjoys the best tracking accuracy with a 4×3 grid and 5.67cm marker configuration, thus being our choice of setting.

3.3.2 Evaluation in Real Environment. Once we determined good parameter settings to configure the Vivepaper marker grid, we used real-world images to compare our tracker's accuracy and speed with two widely used trackers, Apriltag and ArUco. Our evaluation was based on two factors:

(1) *False positive detection rate.* Although using the marker grid configuration can reduce false negatives (thanks to redundancy), false positives can still be problematic.
(2) *Virtual-book jitter.* Inaccuracy of the detected corner locations of marker quads may lead to jitter in the virtual book in the virtual display, degrading user experience.

We downloaded $207,849$ images from LabelMe [31] to compare the false positive rate between our evaluated trackers. These images are various indoor/outdoor photos containing no markers. If a marker is detected in an image, we consider that to be a false



(a) (b)

Figure 5: (a) False positive rates tested on LabelMe [31]. (b) Comparison of four trackers under normal condition and suboptimal illumination conditions (*).

positive. Figure 5(a) shows that Apriltag suffers from the highest false positive rate, while our tracker enjoys the lowest.

Virtual-book jitter is an important performance issue for Vivepaper. If the virtual book displayed in the virtual space keeps vibrating even with the user holding the card stock stable, the jitter may cause the user dizziness or discomfort. The jitter problem is due to the inaccuracy of the detected corner locations of marker quads caused by imperfections in digital camera capture sensors (such as Gaussian noise and relative low resolution).

The experiment was set up by placing the same camera used by Vivepaper in an indoor environment, at three fixed locations (0.2m, 0.4m, and 0.6m) with respect to a card stock, and with three different angles (0, 20, and −20 degrees). Hence, 9 videos were captured. In addition to normal lighting conditions, this process was repeated in suboptimal lighting conditions. We use the box plot [43] to analyze the quad corner jitter of four methods: 1) Apriltag, 2) ArUco, 3) Vivepaper1 (before corner refinement), and 4) Vivepaper2 (after corner refinement).

Due to Gaussian noise and low resolution in camera sensors, the corners detected at the quad detection stage may be slightly different between frames. In each video, we assume all the 24 markers (96 corners) are visible. For each corner's coordinate we computed the standard derivation across the 500 frames, denoted as corner estimation error (jitter level). Ideally, this error should be zero. We take the corner estimation error of all the corners in the 9 videos (no more than 96×9 values), and show each tracker's corner estimation error under normal and suboptimal lighting conditions in a box plot (Figure 5(b)).

In Figure 5(b), our observations are: 1) Regarding the deviation range of lower first and upper third quartiles, ArUco's third quartiles is higher than those of the other methods. Apriltag is the most accurate method with the lowest deviation, attributed to its complicated method of quad detection. 2) Regarding the minimum and maximum deviation values (lower/upper ends of the whiskers), Apriltag suffers from high false positives. Some more extreme outliers may appear, leading to a higher deviation value than our method, which can be seen in Figure 5(b) where the upper ends of the whiskers of Apriltag is higher than those of our method. 3) Regarding the median (middle band in the box), Apriltag and our method are lower than those of the other two. Vivepaper without corner refinement suffers from the highest deviations, and its jitter level under suboptimal lighting conditions (denoted with *) worsens.

Although the median jitter level of our method is slightly higher than Apriltag's, it is small enough to provide good user experience.

Table 2: Average processing time (μs) on PC and Mobile. Libraries used to speed up: * CUDA, # OpenMP, + FastCV.

Test case	Sub steps	AprilTag		ArUco		Vivepaper (CPU only)		Vivepaper (boosted)	
		PC	Mobile	PC	Mobile	PC	Mobile	PC	Mobile
6 markers (half page card stock)	Quad det	29, 221	102, 597	2, 944#	34, 661	8, 810	17, 396	4, 333*	6, 950+
	Decoding	191	942	514	15, 779	230	1, 349	108#	383
	Pose est.	840	3, 300	727	2, 795	1, 947	3, 224	845#	197+
	Total	33.1fps	9.4fps	238.9fps	18.8fps	91.0fps	45.5fps	189.2fps	132.8fps
12 markers (one page card stock)	Quad det	41, 627	127, 751	3, 092#	41, 134	8, 945	21, 230	4, 398*	8, 553+
	Decoding	414	1, 422	599	16, 225	516	1, 727	204#	726
	Pose est.	1, 735	7, 017	1, 201	6, 194	4, 604	6, 465	1, 566#	274+
	Total	22.8fps	7.3fps	204.4fps	15.7fps	71.1fps	34.0fps	162.1fps	104.7fps
24 markers (two pages card stock)	Quad det	44, 451	124, 741	3, 759#	42, 621	9, 722	21, 419	5, 080*	7, 859+
	Decoding	584	1, 618	866	22, 762	884	3, 018	347#	1, 318
	Pose est.	2, 798	13, 201	1, 923	12, 635	8, 192	13, 136	2, 723#	1, 422+
	Total	20.9fps	7.2fps	152.7fps	12.8fps	53.2fps	26.6fps	122.7fps	94.3fps

Consider the deviation range (lower to upper ends of whiskers). Vivepaper's method enjoys the smallest range and hence the most stable. Furthermore, as we will show next that when speed is considered, our method runs much faster than Apriltag to support the required 60fps.

3.3.3 Processing Time. For a real-time AR product, the processing time is a critical factor for good user experience. The latency of Vivepaper should be no longer than 16ms (or 60fps) or it can easily cause discomfort for the user. This section compares Vivepaper, Apriltag, and ArUco on their processing time in quad detection, marker decoding, and pose estimation.

For better speed, we implemented several substeps with CUDA on PC and FastCV on Android, as well as some parallelizable steps on OpenMP. The same techniques are also applied to speed up the performance of detecting hand gestures. Of the latter two systems, Apriltag and ArUco have only open-sourced the CPU versions.

On the PC platform, our experiments were conducted on a computer with an Intel(R) Core(TM) i7-3770 working at 3.40GHz, running Windows 10. The processor has four cores, each running two threads. The DRAM size is 16GB. We took three video sequences with different numbers of markers, 6, 12, and 24, respectively. In general, handling more markers demands a longer processing time, especially in the last two steps of the pipeline: marker decoding and pose estimation. Each video sequence contains 5, 000 frames with a resolution of 612×460 pixels taken by VIVE's frontal camera in a regular indoor environment. Table 2 summarizes the results.

As shown in Table 2, Apriltag suffers from the worst performance in all test cases. ArUco and Vivepaper use a contour-based method to detect quads, which is much faster than the edge-based method used in Apriltag. ArUco performs even better than Vivepaper, reaching 204.4fps and 162.1fps. The reason our system is slower than ArUco is that we added some extra steps to improve accuracy as described in Section 3.1. Nevertheless, Vivepaper is still fast enough for a real-time AR application. The boosted version of Vivepaper achieves more than 2× speedup compared to the original version.

On mobile, we used a state-of-the-art phone, the Google Pixel, with a Qualcomm821 (MSM8996pro) processor and 4GB DRAM running Android 7.1. This time, we used the phone's camera to take three video sequences using the same three different configurations of markers. Each video sequence has 5, 000 frames at a resolution of 800×600. Table 2 presents the results. It turns out that our FastCV version enjoys a significant performance boost (3× to 4×) over the CPU version.

In summary, Vivepaper reaches 162fps on average on PC and 104fps on mobile, both exceeding the 60fps speed requirement of a real-time AR system.

4 HAND GESTURES

Hand gestures are the most natural way for users to interact with books. However, reliable hand tracking and gesture recognition is still a very challenging problem. Vivepaper simplifies the problem by focusing hand segmentation and gesture recognition in the physical boundary of the card stock. In its current version, Vivepaper supports two gestures: *page flipping* and *point-and-click*. By utilizing the priorly known marker-grid background, we can achieve reliable hand segmentation and fingertip detection, even with a camera of 600×400 resolution.

4.1 Hand Segmentation

When using skin color as the visual cue for a hand, we must consider various factors such as illumination, background, and camera characteristics [19]. For an input image I_i, we use the HSV color space to conduct skin color detection. As concluded in [19], the transformation of RGB to HSV makes color identification more invariant to high intensities of white light and ambient light, as well as surface orientations relative to the light source. The HLS representation is as effective as HSV for color identification. Vivepaper uses the HSV color space in the resource constrained mobile platform, since the FastCV library provides a quick RGB to HSV conversion API.

To better separate skin from non-skin, we use a white balancing algorithm to pre-process image I_i. The algorithm removes color casts caused by the light and yields I_i'. Starting from n color seeds, we use a boundary decision method for skin color detection. We obtain the bitwise-or M_i of several binary masks m_n with respect to n different color seeds. After we have obtained the binary mask M_i, we use post-processing rules to remove some of the false positives, yielding M_i'. Our post-processing takes advantage of the known marker-grid background. Fingertip detection and localization are subsequently performed based on the post-processed mask M_i'.

4.1.1 Calibration. Differences in illumination and camera characteristics are also issues that must be dealt with when using skin color based methods. If a user wears a pair of gloves, the predefined color seeds may fail to identify the correct hand regions. Vivepaper thus performs calibration for different users under various environments beforehand. Vivepaper requests users to put their hands on a specified region of the card stock for one second and obtains n color seeds for hand segmentation. Once the color seeds and the location of a hand is identified, the changes in color due to environmental factors or hand movement can be subsequently captured to calibrate color seeds dynamically.

4.1.2 Controlled Background. One of the issues for skin color based methods is that background colors could be similar to the skin color, causing false positives. Therefore, Vivepaper takes advantage of a controlled background. Since the operating region of the user's hands in Vivepaper is within the boundary of the card stock, the background for the hands can be limited to the black and white markers. We take advantage of the boundary and known color of card stock to obtain better accuracy. First, we use the boundary of the card stock to remove regions with similar color to the user's skin in the background. Second, we obtain the temporal colors of black and white for different illuminations and separate them from the skin color mask M_i'. We also eliminate the regions of detected markers with the quads of markers. In addition, processing time is reduced because we handle only the pixels within the boundary of the card stock.

4.1.3 Empirical Validations. To validate the practicality of our strategies, we set up a dataset, which was collected when users used Vivepaper. This dataset contains 600 images from one male and one female user, collected under white and yellow light. Additionally, for each light color we collected images under three illumination intensities: dark, normal, and bright. For each image, we manually labeled a hand segmentation ground-truth. Since our main concern is the hand position inside the card stock, we labeled the card stock's boundary in each image. We calculated both the accuracy and false positive rate inside the boundary.

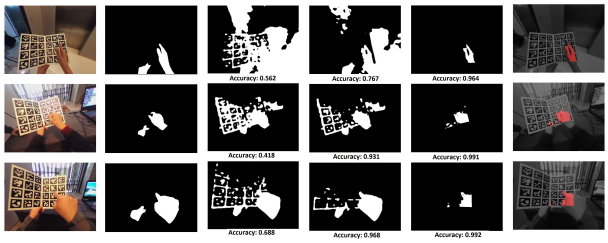
Table 3: Hand segmentation results.

Accuracy/False Positive Rate		Calibration Only	+ White Balance	+ Controlled Background
White Light	Dark	0.98/0.02	0.99/0.01	0.99/0.01
	Normal	0.99/0.01	0.99/0.00	0.99/0.00
	Bright	0.99/0.01	0.99/0.01	0.99/0.01
Yellow Light	Dark	0.92/0.07	0.95/0.03	0.95/0.01
	Normal	0.64/0.40	0.88/0.13	0.94/0.06
	Bright	0.92/0.09	0.91/0.08	0.93/0.06

Table 3 presents the accuracy and false positive rate of hand segmentation in three processing stages: calibration only, with white balancing, and with controlled background. Each stage is conducted based on the previous one. We can see from the table that under white light, calibration alone suffices. Under yellow light, additional processing stages can improve segmentation accuracy. The reason for this improvement is that when yellow light shines on the card stock, white colored regions on the card stock can appear to be skin color, as shown in Figure 6(c). Performing white balancing can partially remedy this ambiguity problem, as shown in Figure 6(d). However, there are still white colored regions on the card stock that can be read incorrectly. A background of a similar color to the skin can cause segmentation confusion, as shown in Figure 6(d). When we restrict hand segmentation to be within the boundary of the card stock, the controlled background allows us to further improve segmentation accuracy under yellow light, as shown in Figure 6(e).

4.2 Fingertip Detection

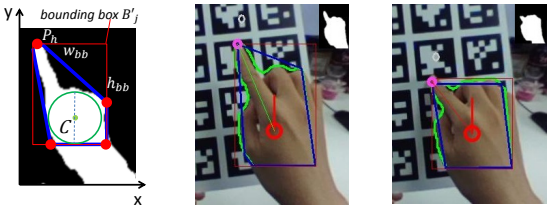
Accurately locating fingertips from images is critical for recognizing the two gestures (i.e., page flipping and point-and-click) that Vivepaper supports. Gestures are recognized by tracking the movement of the user's fingertip. After hand segmentation, the binary mask M_i' is obtained. We regard the top two largest contours $K_j (j = 0, 1)$ of M_i' as hands. Because M_i' may contain a region of the arm, the palm center of each hand is estimated using the method in [44] to distinguish between the hand and arm regions. As illustrated in Figure 7(a), the center C of the largest circle inside K_j is regarded



(a) (b) (c) (d) (e) (f)

Figure 6: Comparison of segmentation results. (a) Original; (b) Ground truth mask; (c) Segmentation with calibration; (d) with white balancing; (e) with controlled background; (f) Composition of the final segmentation with the original. Accuracy is calculated inside card-stock boundary.

...garded as the length of ...nd refined contours $K_j^{'}$ ... region of the arm.



(a) (b) (c)

Figure 7: Fingertip Detection. (a) Fingertip detection with hand's bounding box $B^{'}$ (red), convex hull (blue), convex points S_{convex} (red points); (b) Detected fingertip P_h (pink point); (c) Estimated fingertip (pink) in less prominent situation.

Our pointing fingertip detector is rule-based. We consider the standard upward-pointing gesture first. Intuitively, we select the highest convex point P_h of K_j' as the pointing fingertip if it is prominent enough. That is,

$$\forall P \neq P_h, P \in S_{convex}, P_h.y - P.y > h_{bb}/5, \quad (1)$$

where S_{convex} denotes the set of convex points of K_j' and h_{bb} denotes height of the bounding box B_j', as Figure 7(a) illustrates.

Because Vivepaper uses only one camera, self-occlusion can affect detection stability. While the user is performing the pointing gesture, the pointing finger may be occluded by itself or by the palm, due to the single camera view point (see Figure 7(c)). In this situation, the P_h selected before becomes less prominent and the constraint in the preceding paragraph is not satisfied. We handle this issue by estimating the position of the fingertip based on the historical information of P_h. Using the upward-pointing gesture as an example, in the i^{th} frame, the *estimated fingertip* P^i_{est} is located on the top bounding edge according to Eq. 2

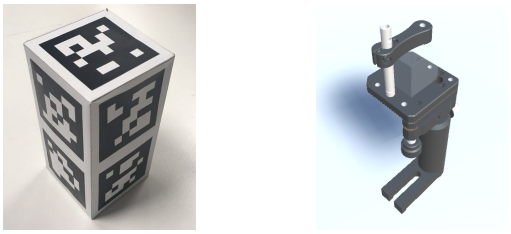
$$P^i_{est} = P^i_{bb} + r^i * \begin{bmatrix} w^i_{bb} \\ 0 \end{bmatrix}, r^i = \begin{cases} d(P^t_h, P^t_{bb})/w^t_{bb}, & \text{if } i - t \leq T \\ 0.5, & \text{otherwise,} \end{cases} \quad (2)$$

where P^t_h, P^t_{bb}, and w^t_{bb} denote the pointing fingertip, the top-left corner, and the bounding box's width respectively in the t^{th} frame, which is the latest frame that satisfies the constraint in Eq. 1. And r^i is equal to the ratio of the distance between P^t_h, P^t_{bb} to w^t_{bb}, if the t^{th} frame is in the window of T frames preceding it, where T is a pre-defined parameter to limit the effective range of the historical information.

Our usability study confirms that virtually all users can adapt automatically to using one finger to perform page flipping and point-and-click gestures. Most users can self-adjust to make their gestures clearly visible to the camera to achieve effective interactions.

5 CONCLUDING REMARKS

We have deployed Aristo as an augmented reality platform to support the vertical domains of education, training, and tourism. This paper uses one of the platform's launched applications, Vivepaper, as an example to illustrate our design considerations and parameter settings in marker tracking and gesture recognition.



(a) 3D object with markers **(b) Virtual object in HMD**

Figure 8: Aristo in a training scenario.

In addition to the Vivepaper deployment, Aristo has also been deployed for training novice workers. Instead of tracking a piece of 2D card stock, this training application tracks several 3D elongated cuboids each printed with 10 different fiducial markers (Figure 8(a)). Each cuboid is rendered into a piece of real-world equipment or product part (an example is shown in Figure 8(b)). Since the surface of each cuboid's face is much smaller than that of the card stock, Aristo cannot use the marker-grid scheme. To maintain high tracking accuracy, Aristo uses recent positions of a cuboid and the position of hand to estimate the new pose of the cuboid. Furthermore, to support the notion of "holding", Aristo must track finger positions on a cuboid and render their positions on the corresponding virtual object to provide visual feedback. We present detailed information on the extended version of this paper [46].

We are currently embarking on three enhancements. First, we have been experimenting with a deep learning pipeline, CLKN [9], to substantially enhance our tracking and segmentation subroutines. One main goal is to trim down the final model size to support at least 60fps on mobile phones. Second, we plan to experiment with RGB-D cameras to enhance tracking accuracy. Finally, we are developing prototypes with partners to integrate with a wide variety of physical objects (e.g., piano keyboards, product parts, and medical instruments) that can be recognized, tracked, and interacted with by users.

REFERENCES

[1] 2017. Leap Motion. https://www.leapmotion.com. (2017).
[2] 2017. Open Source Computer Vision Library. https://github.com/opencv. (2017).
[3] 2017. Unity3D. https://unity3d.com. (2017).
[4] Lorenzo Baraldi, Francesco Paci, Giuseppe Serra, Luca Benini, and Rita Cucchiara. 2014. Gesture recognition in ego-centric videos using dense trajectories and hand segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 688–693.
[5] Ross Bencina, Martin Kaltenbrunner, and Sergi Jorda. 2005. Improved topological fiducial tracking in the reactivision system. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*. IEEE, 99–99.
[6] Filippo Bergamasco, Andrea Albarelli, Luca Cosmo, Emanuele Rodola, and Andrea Torsello. 2016. An accurate and robust artificial marker based on cyclic codes. *IEEE transactions on pattern analysis and machine intelligence* 38, 12 (2016), 2359–2373.
[7] Alejandro Betancourt, Lucio Marcenaro, Emilia Barakova, Matthias Rauterberg, and Carlo Regazzoni. 2016. GPU accelerated left/right hand-segmentation in first person vision. In *European Conference on Computer Vision*. Springer, 504–517.
[8] Alejandro Betancourt, Pietro Morerio, Emilia Barakova, Lucio Marcenaro, Matthias Rauterberg, and Carlo Regazzoni. 2017. Left/right hand segmentation in egocentric videos. *Computer Vision and Image Understanding* 154 (2017), 73–81.
[9] Che-Han Chang, Chun-Nan Chou, and Edward Y. Chang. 2017. CLKN: Cascaded Lucas-Kanade Networks for Image Alignment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[10] Kyusung Cho, Juho Lee, JS Lee, and HS Yang. 2007. A realistic e-learning system based on mixed reality. In *13th International Conference on Virtual Systems and Multimedia*. 57–64.
[11] Andrew I Comport, Eric Marchand, Muriel Pressigout, and Francois Chaumette. 2006. Real-time markerless tracking for augmented reality: the virtual visual servoing framework. *IEEE Transactions on visualization and computer graphics* 12, 4 (2006), 615–628.
[12] Enrico Costanza and John Robinson. 2003. A Region Adjacency Tree Approach to the Detection and Design of Fiducials. (2003).
[13] Mark Fiala. 2005. ARTag, a fiducial marker system using digital techniques. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Vol. 2. IEEE, 590–596.
[14] Mark Fiala. 2005. Comparing artag and artoolkit plus fiducial marker systems. In *Haptic Audio Visual Environments and their Applications, 2005. IEEE International Workshop on*. IEEE, 6–pp.
[15] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition* 47, 6 (2014), 2280 – 2292. https://doi.org/10.1016/j.patcog.2014.01.005
[16] Oleg Grinchuk, Vadim Lebedev, and Victor Lempitsky. 2016. Learnable Visual Markers. In *Advances In Neural Information Processing Systems*. 4143–4151.
[17] Taejin Ha, Yeongmi Kim, Jeha Ryu, and Woontack Woo. 2006. Enhancing immersiveness in AR-based product design. In *Advances in Artificial Reality and Tele-Existence*. Springer, 207–216.
[18] Taejin Ha, Youngho Lee, and Woontack Woo. 2011. Digilog book for temple bell tolling experience based on interactive augmented reality. *Virtual Reality* 15, 4 (2011), 295–309.
[19] Praveen Kakumanu, Sokratis Makrogiannis, and Nikolaos Bourbakis. 2007. A survey of skin-color modeling and detection methods. *Pattern recognition* 40, 3 (2007), 1106–1122.
[20] Byeongkeun Kang, Kar-Han Tan, Hung-Shuo Tai, Daniel Tretter, and Truong Q Nguyen. 2016. Hand Segmentation for Hand-Object Interaction from Depth map. *arXiv preprint arXiv:1603.02345* (2016).
[21] Hirokazu Kato and Mark Billinghurst. 1999. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999.(IWAR'99) Proceedings. 2nd IEEE and ACM International Workshop on*. IEEE, 85–94.
[22] Kiyoung Kim, Vincent Lepetit, and Woontack Woo. 2010. Scalable real-time planar targets tracking for digilog books. *The Visual Computer* 26, 6 (2010), 1145–1154.
[23] Cheng Li and Kris M Kitani. 2013. Pixel-level hand detection in ego-centric videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3570–3577.
[24] Hui Liang, Jin Wang, Qian Sun, Yong-Jin Liu, Junsong Yuan, Jun Luo, and Ying He. 2016. Barehanded music: real-time hand interaction for virtual piano. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 87–94.
[25] Minghuang Ma, Haoqi Fan, and Kris M Kitani. 2016. Going deeper into first-person activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1894–1903.
[26] Shahzad Malik, Chris McDonald, and Gerhard Roth. 2002. Hand tracking for interactive pattern-based augmented reality. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 117.
[27] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. 2016. Pose estimation for augmented reality: a hands-on survey. *IEEE transactions on visualization and computer graphics* 22, 12 (2016), 2633–2651.
[28] George Margetis, Xenophon Zabulis, Panagiotis Koutlemanis, Margherita Antona, and Constantine Stephanidis. 2013. Augmented interaction with physical books in an Ambient Intelligence learning environment. *Multimedia tools and applications* 67, 2 (2013), 473–495.
[29] Edwin Olson. 2011. AprilTag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 3400–3407.
[30] Siddharth S Rautaray and Anupam Agrawal. 2015. Vision based hand gesture recognition for human computer interaction: a survey. *Artificial Intelligence Review* 43, 1 (2015), 1–54.
[31] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. 2008. LabelMe: a database and web-based tool for image annotation. *International journal of computer vision* 77, 1-3 (2008), 157–173.
[32] Tomoki Issac Saso, Kenji Iguchi, and Masa Inakage. 2003. Little red: storytelling in mixed reality. In *ACM SIGGRAPH 2003 Sketches & Applications*. ACM, 1–1.
[33] Camille Scherrer, Julien Pilet, Pascal Fua, and Vincent Lepetit. 2008. The haunted book. In *Proceedings of the 7th IEEE/ACM international Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 163–164.
[34] Ayan Sinha, Chiho Choi, and Karthik Ramani. 2016. Deephand: Robust hand pose estimation by completing a matrix imputed with deep features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4150–4158.
[35] Satoshi Suzuki et al. 1985. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing* 30, 1 (1985), 32–46.
[36] Nobuko Taketa, Kenichi Hayashi, Hirokazu Kato, and Shogo Noshida. 2007. Virtual pop-up book based on augmented reality. *Human Interface and the Management of Information. Interacting in Information Environments* (2007), 475–484.
[37] Henning Tjaden, Ulrich Schwanecke, and Elmar Schömer. 2016. Real-Time monocular segmentation and pose tracking of multiple objects. In *European Conference on Computer Vision*. Springer, 423–438.
[38] Poonsri Vate-U-Lan. 2012. An augmented reality 3d pop-up book: the development of a multimedia project for English language teaching. In *Multimedia and Expo (ICME), 2012 IEEE International Conference on*. IEEE, 890–895.
[39] Tadej Vodopivec, Vincent Lepetit, and Peter Peer. 2016. Fine Hand Segmentation using Convolutional Neural Networks. *arXiv preprint arXiv:1608.07454* (2016).
[40] Daniel Wagner and Dieter Schmalstieg. 2007. ARToolKitPlus for Pose Tracking on Mobile Devices. In *Computer Vision Winter Workshop Cvww*.
[41] Daniel Wagner and Dieter Schmalstieg. 2009. Making augmented reality practical on mobile phones, part 1. *IEEE Computer Graphics and Applications* 29, 3 (2009).
[42] John Wang and Edwin Olson. 2016. AprilTag 2: Efficient and robust fiducial detection. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 4193–4198.
[43] David F Williamson, Robert A Parker, and Juliette S Kendrick. 1989. The box plot: a simple visual method to interpret data. *Annals of internal medicine* 110, 11 (1989), 916–921.
[44] Zhengwei Yao, Zhigeng Pan, and Shuchang Xu. 2013. Wrist recognition and the center of the palm estimation based on depth camera. In *Virtual Reality and Visualization (ICVRV), 2013 International Conference on*. IEEE, 100–105.
[45] Zhengyou Zhang, Ying Wu, Ying Shan, and Steven Shafer. 2001. Visual panel: virtual mouse, keyboard and 3D controller with an ordinary piece of paper. In *Proceedings of the 2001 workshop on Perceptive user interfaces*. ACM, 1–8.
[46] Zhongyang Zheng, Bo Wang, Yakun Wang, Shuang Yang, Zhongqian Dong, Tianyang Yi, Cyrus Choi, Emily Chang, and Edward Y. Chang. 2017. Aristo: An Augmented Reality Platform for Interactivity and Immersion (extended version). In *HTC Technical Report*.
[47] Yang Zhou, Bingbing Ni, Richang Hong, Xiaokang Yang, and Qi Tian. 2016. Cascaded interactional targeting network for egocentric video analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1904–1913.
[48] Xiaolong Zhu, Xuhui Jia, and Kwan-Yee K Wong. 2014. Pixel-level hand detection with shape-aware structured forests. In *Asian Conference on Computer Vision*. Springer, 64–78.