PROMAR: Practical Reference Object-based Multi-user Augmented Reality

Tengpeng Li*, Nam Son Nguyen*, Xiaoqian Zhang*, Teng Wang†, and Bo Sheng*
*Department of Computer Science, University of Massachusetts Boston
*Email: {Tengpeng.Li001, SonNam.Nguyen001, Xiaoqian.Zhang001, Bo.Sheng}@umb.edu
† Electronic Arts, 207 Redwood Shores Pkwy, Redwood City, CA 94065, Email: tewang@ea.com

***Abstract*—Augmented reality (AR) is an emerging technology that weaves virtual objects into physical environments, and enables users to interact with them through viewing devices. This paper targets on multi-user AR applications, where virtual objects (VOs) placed by a user can be viewed by other users. We develop a practical framework that supports the basic multi-user AR functions of placing and viewing VOs, and our system can be deployed on off-the-shelf smartphones without special hardware. The main technical challenge we address is that when facing the exact same scene, the user who places the VO and the user who views the VO may have different view angles and distances to the scene. This setting is realistic and the traditional solutions yield a poor performance in terms of the accuracy. In this work, we have developed a suite of algorithms that help the viewers accurately identify the same scene and restore the VO under a moderate range of view angle difference. We have prototyped our system, and the experimental results have shown significant performance improvements. Our source codes and demos can be accessed at https://github.com/PROMAR2019.**

I. Introduction

Mobile Augmented Reality (MAR) represents an emerging category of applications that bring users interactive experiences with the physical environment. In such applications, users can place virtual objects in real-world space, and view them through a camera view. Several toolkits have been developed recently to support MAR applications, such as ARCore [1], ARKit [2], Vuforia [3], and Wikitude [4].

In this paper, we target on a framework that supports multi-users interactions for the MAR apps, where an overlay scene including the physical environment and the virtual objects (VO) are shared among multiple users. In a simple setting, the VO placed by one user can be recognized by other users. Multi-user MAR has a bright application scope by benefiting our lives in many ways. For example, a user can leave virtual marks to guide visitors to her or his office; a repairman can locate a broken chair by finding the VO submitted with the repair request; and people can play the scavenger hunt game to find the VOs placed by other users. We aim to develop a practical framework that can be deployed with general hardware and software settings. More detailed system requirements will be introduced in Section III. Most importantly, we expect the target system to tolerate moderate viewpoint difference between the user who places the VO and the user who views it. This realistic setting leads to significant research challenges in system design.

This project was supported by National Science Foundation grant CNS-1527336, and UMass Boston Oracle Doctoral Research Fellowship.

The current AR systems, however, are mostly designed for a single user. They often require additional processes to effectively work in practice, e.g., preloading high quality 'tag' images, or have constraints on the VO placement (ARCore [1] and ARKit [2] only place virtual objects on recognized plane surfaces by default). Some cloud-side services such as 'ARCore cloud anchors' can support multi-user AR applications, but have the same constraint of plane surface, and it requires a 'scanning' phase to map the environment. The VO position's accuracy is also low with viewpoint changes.

In this paper, we present PROMAR, a practical framework supporting multi-user AR applications. Our system does not require special hardware or preprocessing of scenes. The virtual objects can be arbitrarily placed in the physical environment. Basically, when a user places a virtual object, our system recognizes the objects in the scene, and use them as reference objects. The other users who wants to view the virtual objects need to identify the recorded reference objects, and then further restore the virtual object in their camera views. In this way, we convert the problem of recognizing the exact scene into the problem of recognizing the same reference objects. The basic design of our system is based on the image features that have been well studied in computer vision. We have developed novel algorithms for image feature extraction and comparison particularly for multi-user AR applications. We consider not only the image feature properties, but also the phone's viewpoint data such as the orientation readings. To the best of our knowledge, this is the first system work that supports multi-user AR applications in a practical setting. We have prototyped our system, and conducted extensive experiments. The results show that the scene recognition accuracy is high even with the viewpoint changes, and the runtime image processing is fast suitable for real-time applications.

II. Related Work

Mobile augmented reality has been an extensively exploited field, as several frameworks have been released to assist developers with low level function implementations, such as ARKit 2 [2], ARCore [1], Wikitude [4], Vuforia [3] to name a few. The latest releases of ARCore and ARKit have announced to support multiple-user AR, but the prerequisite of detecting

plane restricted the deployment and their performance in a practical setting with viewpoint differences is not satisfactory. The techniques used in our solution are related to the prior work in the following categories.

Object detection. Detecting physical objects in a image would help understand and identify the environment in an AR application. Multiple methods [5], [6], [7], [8], [9], [10] were used to categorize an object from an input image with given category. More recently, deep neural networks [11], [12], [13], [14], [15], [16] have dominated this field and achieved impressive precision. In this paper, the design goal is to recognize an exact target object instead of the category of the object. Our solution uses object detection to identify reference objects in a frame to confine the region where feature points are extracted. The feature matching algorithm in our solution is based on the traditional computer vision methods, such as SIFT[17], SURF[18], ORB[19], and others [20], [21], [22], [23]. But these existing approaches incur large computation overhead and the matching accuracy is low with the change of viewpoints. Our solution significantly improves the overhead and accuracy with new algorithm designs.

Distance estimation. To present an vivid augmented reality, an accurate distance estimation is crucial for placing the virtual object on the right position. Inspired by human's depth perception system which takes advantage of binocular disparity, stereo cameras [24] use two or more lens to capture the scene to obtain the depth. [25], [26], [27], [28] put forward different algorithms to solve occlusion problem in depth measuring. Another option is the sensor-equipped cameras, such as time-of-flight (TOF) cameras [29] which can measure the depth by counting the time for the light traveling between the objects and the camera. In this paper, we develop new techniques to estimate the distance without special hardware.

Scene recognition and localization. Scene recognition is necessary to reconstruct the virtual scenes in multi-user AR project. There have been a lot of work in the literature using different models and features [30], [31], [32], [33], [34]. However, their goal is to recognize the category of the scene while in our system setting, the multi-user AR applications require the recognition of the exact scene. Advanced Simultaneous Localization and Mapping (SLAM) techniques, such as [35], [36], [37], can help solve this problem. But they are not feasible in our application setting. Finally, indoor localization algorithms, e.g., [38], [39], [40], can be an alternative to recognize specific indoor environments. However, our problem requires accurate placement of virtual objects that must involve image features. We would consider incorporating localization schemes to improve the performance in the future work.

III. System Model and Motivation

In this section, we first present our problem formulation and some challenges the existing solutions cannot address.

A. System Setting

We consider a general scenario for multi-user AR applications, where virtual objects (VOs) are placed in scenes by a user and can be viewed by other users. Thus, we define two user roles in the application as follows:

- **VO owner:** the user who places VOs in the scenes.
- **VO viewer:** the user who views VOs based on the information passed by the VO owner.

After placing a VO, the VO owner prepares a package file to dispatch to designated VO viewers so that they can view the VO in the proximity (illustrated in Fig.1).

We consider a practical setting for multi-user AR applications with the following requirements and features.

Hardware requirements. First, our target applications are based on common off-the-shelf mobile devices. We do not assume special hardware that can measure the distance from a camera view such as in-depth cameras and laser sensors. In addition, we assume that there is no assistance from other devices in the environment. The VO owner and viewers independently launch the application on commercial smartphones.

VO placement. We consider a VO can be placed arbitrarily. The VO does not have to be placed on a plane (e.g., in [1], [2]) or attached to a particular object whose image has to be uploaded and processed beforehand (e.g., in [4], [3]).

Viewpoint difference. In a realistic setting, our system is expected to tolerate the viewpoint difference between the VO owner and viewers. As illustrated in Fig.1, the VO owner and viewers may view the same scene from different angles and distances. We expect the VO viewers to accurately restore the VO if the viewpoint difference is within a certain range.

Runtime overhead. Our target system for the VO viewers is a real-time application with a camera view user interface. It refreshes the camera frame frequently and requires instant response to achieve smooth user experiences. Thus, the system has a rigorous requirement of keeping the runtime computation overhead to a minimum level.

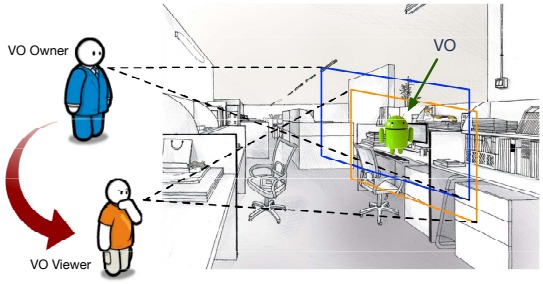

Fig. 1. An application example

B. Challenges

The key function in such applications is to enable the VO viewers to recognize the extract scene where the VO owner places the VO. Our basic solution is essentially based on the existing image analysis and comparison techniques in the literature. However, we find that the following two challenges significantly hinder the deployment of the regular approaches.

1) Recognition accuracy: The first issue is the accuracy of recognizing the extract scene with the viewpoint difference. We have conducted experiments to examine the performance of the traditional approaches. We use the images from

ALOI [41] where each image represents a quite simplified scene with a single object. For each object, the dataset includes the images taken from different horizontal angles ranging from 0 to 360 degrees with an interval of 5 degrees.

Fig. 2 illustrates the accuracy results of the traditional ORB [19] algorithm. We randomly pick 5 objects in our experiments. For each object, we select one image at angle d as the VO owner's image, and compare it with the images taken at a different angle $d + \delta$ as the VO viewer's image. In our setting, δ varies from -30 to $+30$. Then, we use ORB algorithm to extract feature points (FPs) from both images, and compare these two sets of FPs. In this test, we extract 100 FPs from each image, and apply Brute-Force matching algorithm with common techniques such as cross-checking (refer to OpenCV [42] library). The average matching ratios, i.e., the ratio of the matched FPs and the total FPs, are reported in Fig. 2. In addition, we have also tested the accuracy when comparing to *false images* which are not relevant to the same scene. In our tests, we search the object title in Google image, and the first 100 returned images are downloaded as the false images. The average values of the matching ratios with the false images are illustrated in the last bar in Fig. 2.

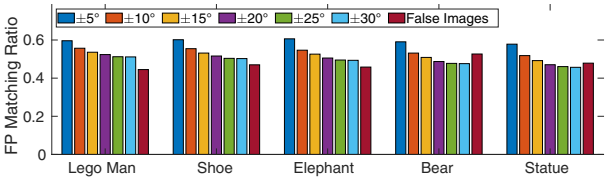

Fig. 2. Matching ratios with change of view angles

We observe that with a moderate angle change, it is not easy to distinguish the false images from the 'true' images by checking the matching ratio. If we set a threshold of the matching ratio for recognizing the scene, it will either miss many 'true' cases or yield a high false positive rate.

2) Computation overhead: The computation overhead is critical for real-time applications. As we will present with more details in the next section, our solution is based on the traditional feature comparison. The frame where the VO owner places the VO is compared to the frames captured by the VO viewers. We find that the computation overhead of feature comparison heavily depends on the number of feature points extracted from each frame.

We conduct experiments on a OnePlus 7 Android phone with OpenCV library. We choose ORB [19] feature extraction algorithm in our experiments, and the algorithm allows us to configure a parameter to indicate the number of FPs in use (the default value is 500). Fig. 3 illustrates the experimental results of the computation overhead measured on the phone. We select two images and extract different numbers of FPs from both of them for comparison. The results in the figure are the average values of 100 independent runs for each setting.

We observe that the computation overhead exponentially increases over the number of FPs. When the VO viewer keeps capturing frames and process them at runtime, the number of FPs we shall use for comparison is limited. In our default system setting, we only consider 100 FPs for each frame. This constraint also requires the system to appropriately select the set of FPs to achieve desirable recognition accuracy.

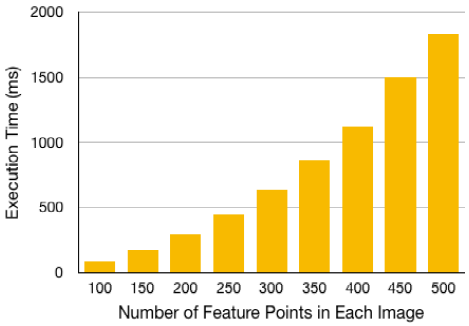

Fig. 3. Execution time of comparing two images' feature points measured on an Android phone.

IV. Design of PROMAR

In this section, we present the design of PROMAR. The major objective of the system is to allow the VO viewers to (1) recognize the exact same scene where the VO owner places the VO, and (2) restore the VO in the right position. We first introduce the concepts of reference objects and matching ratio, and then describe the processes for the VO owner and viewers.

Reference objects (RO): Our solution is based on the objects that can be recognized in the scene. We call them reference objects (ROs). Instead of recognizing the entire scene, the VO viewers intend to recognize the ROs. There are two major benefits that motivate us to use ROs. First, we extract image features from the regions of the ROs, instead of the entire frame, for comparison. In this way, the features are more representative and meaningful avoiding background noises in the image that are unlikely to be persistent. Second, when ROs are available to the VO viewers, it is easier to restore the location of the VO using its relative locations to the ROs.

Matching ratio: The core technical solution in PROMAR to identify ROs. It includes two submodules, (1) feature extraction by the VO owner, and (2) feature matching by the VO viewer. Briefly, the feature extraction takes the rectangle area containing the recognized RO, and outputs a set of feature points (OF) in the area. In the feature matching process, the VO viewer captures a frame, and also extracts FPs from the recognized rectangle area (VF). Then the VO viewer searches each FP $i \in OF$ passed by the VO owner in VF. We use $i \rightarrow j$ to indicate that an input FP $i \in OF$ can find a similar FP j in the frame captured by the VO viewer[1] . Then, the VO viewer calculates the matching ratio as follows,

$$MR = \frac{|M|}{|OF|}, \quad M = \{i \rightarrow j \mid \forall i \in OF, \exists j \in VF\}, \qquad (1)$$

where M is a subset of OF hosting all the matching FPs. The feature matching process returns true if the matching ratio is greater than a threshold parameter τ.

[1]Note that this matching relation is directional.

A. VO Owner: Place a VO

In PROMAR, a user places a VO through camera view with an existing AR framework, e.g., ARCore [1]. We have modified the framework so that the VO can be placed in an arbitrary location rather than on a plane surface in the default configuration. With such a setting, ARCore requires the VO to be specified with three dimensional coordinates relative to the camera. These coordinates are in the unit of physical distance (e.g., meter), and will be transformed to real world coordinate space by the AR framework.

In PROMAR, we place the VO in the center of the camera view by default. Thus, two of the three coordinates are set to 0 indicating that from the camera's view, the VO is on the same horizontal and vertical level. The last coordinate represents the distance between the camera and VO. We use one meter as the default value, and provide a seekbar for the user to adjust the value. VO_{dist} denotes this value. Once the user confirms the VO's location, we will record the value of VO_{dist}.

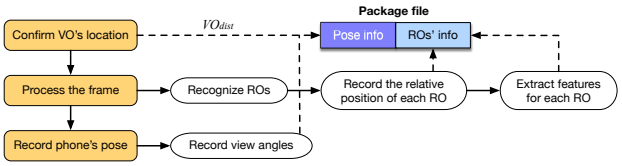

Fig. 4. Program architecture for VO owners

After the user confirms the location of the VO, our system will capture the camera frame at the moment, and process it in the following three steps. (1) We recognize ROs in the frame, and record their relative positions to the VO. (2) For each RO, we conduct a new algorithm based on *distortion tests* to extract the image features. This is our core technical component for the VO owner. (3) Finally, we record the phone's pose information which includes the view angles and the distance between the phone and VO (VO_{dist}). All these data will be bundled into a package file to be delivered to the VO viewer. The major components in the VO owner's process are illustrated in Fig. 4. In the rest of this subsection, we present the details of each step.

1) Recognize reference objects (ROs): In this first step, we recognize the objects in the captured frame as ROs. In our system, we simply use Tensorflow's object detection module to identify the objects. For each recognizable RO, a label of the object and the detection confidence are reported, as well as the rectangle region in the frame. In addition, we record the pixel coordinates of the top-left and bottom-right corners of the rectangle region for each RO. With these information, we can derive the RO's relative position to the VO in the frame.

2) Distortion-based feature extraction: In this step, we extract image features for each RO, and expect these features would help the VO viewers identify the corresponding RO. In practice, the biggest challenge of this module in mobile AR applications is the change of the viewpoints. The VO owner and viewers may not observe the RO from the same angle or distance. As we show in Section III, this change could dramatically affects the matching ratio values.

In PROMAR, we develop a distortion-based feature extraction algorithm aiming to select a set of *robust* FPs. Our main idea is to classify the possible viewpoint changes into multiple types of *distortion*. For each type, the VO owner prepares a set of FPs that are particularly robust to this type of distortion.

Distortion types. In PROMAR, we consider four relevant properties of the viewpoint when defining distortion types: (1) horizontal angle, (2) vertical angle, (3) rotation, and (4) scale (indicating the distance to the scene). For each of these properties, the change would occur in two directions, i.e., 'left' or 'right' for horizontal angle, 'up' or 'down' for vertical angle, 'clockwise' or 'counter-clockwise' for rotation, and 'bigger' or 'smaller' for scale. We consider each directional change of a property as an individual distortion type. Thus, totally we define eight types of distortions in our system.

Distortion tests. The main process is illustrated in Fig. 5. We first use regular feature extraction algorithms such as ORB [19] to extract the feature points of each RO. Then we conduct a distortion test for each type of the distortion. Basically, we generate a set of distorted images of the original RO image mimicing the viewpoint change within a certain range. For each distorted image, we also extract FPs and compare to the FPs from the original RO image. A FP is considered more robust if we can find a match in more distorted images. Our objective is to select a set of N robust FPs for each type of distortion. Fig. 5 illustrates an example with 3 types of distortion (horizontal angle, scale, and rotation), and $N = 5$. The output includes 3 FP sets each including 5 FPs (represented by their indexes). The details of robust FP selection algorithm is presented in the next paragraph.

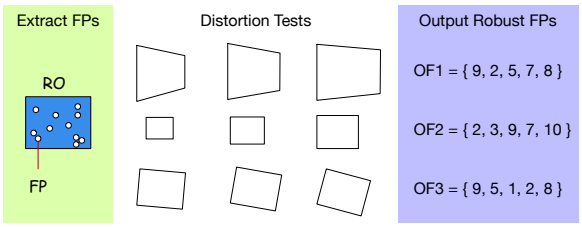

Fig. 5. Distortion-based feature extraction

Robust FP selection. The problem of selecting robust FPs can be formulated as follows. For each identified RO, we extract n FPs $\{p_1, p_2, \ldots, p_n\}$, and apply m types of distortions $\{D_1, D_2, \ldots, D_m\}$. For each distortion type D_j, we generate multiple test cases (TC_j), where $d_{jk} \in TC_j$ indicates the k-th distorted image in this category. We examine each FP and check if it exists (a matching can be found) in each distorted image. We use $x(i, j, k)$ to denote the matching results,

$$x(i,j,k) = \begin{cases} 1 & \text{If } p_i \text{ exists in distorted image } d_{jk}, \\ 0 & \text{Otherwise.} \end{cases}$$

Thus, our goal is to select N most robust FPs for each type of distortion, where N is specified based on overhead constraint.

We define our objective as a max-min problem as follows,

where y_i indicates if p_i is selected or not in the robust FP set.

$$\text{Maximize} \quad \text{Min} \{ \sum_{i \in [1,n]} x(i,j,k) \cdot y_i \mid \forall k \} \tag{2}$$

$$\text{s.t.} \quad \sum_i y_i \leq N, y_i \in \{0,1\}, \forall i \tag{3}$$

Essentially, given a set of FPs, we check the matches we can find in each distorted image. This objective is to maximize the number of matches in the worst case (the distorted image with the fewest matches). The intuition here is that if we assume there are a large set of the distortion test cases, and the future VO viewer's viewpoint change happens to fit in one of them, then the number of matches in this objective is equivalent to the matching ratio for that viewer.

Algorithm 1: Distortion-based Feature Extraction: DFE(D_j, N)

```
1  RET = {}, C = {1,...,n}, minS = {1,...,|TC_j|} ;
2  while |RET| < N do
3      for i ∈ C do
4          c = Σ_{k∈minS} x(i,j,k);
5          if c > max then
6              max = c,  idx = i;
7      RET = RET ∪ idx,  C = C − idx, minS = {};
8      for k ∈ TC_j do
9          c = Σ_{i∈RET} x(i,j,k);
10         if c < min then
11             min = c, minS = {k};
12         else if c == min then
13             minS = minS ∪ k;
14 return RET;
```

In PROMAR, we develop Algorithm 1 to solve the above problem for a given type of distortion. Basically, we apply a greedy algorithm and selects the FPs one by one. We use RET to represent the return set of robust FPs, and C to represent the candidate set of FPs, i.e., n original FPs. In addition, we define a set $minS$, called *minimum set*, to include the distorted cases that yield the minimum number of matches at the moment. Initially, $minS$ includes all the cases because the number of matches are all 0 in the beginning (line 1). The main body of the algorithm is a while loop (lines 2–13) that terminates when RET holds N FPs. In each round, the algorithm enumerates all the candidates (lines 3–6), and picks the FP that appears the most in the minimum set $minS$. Selecting such an FP will evict the most number of distorted images out of the minimum set, and most likely help increase the minimum number of matches in all distorted images. In line 7, the algorithm updates the set RET and C. The minimum set $minS$ is reconstructed in lines 8–13 based on the updated RET.

3) Record phone's pose: Finally, we record the VO owner's phone's pose information as a part of the package file (referring to Fig. 4). We borrow the concept of *pose* from the AR system which defines a relative coordinate system and the corresponding transformation to real world coordinate space. In PROMAR, a pose includes two parts of information. First, we measure the camera's view angles by reading the orientation sensors including values of three axes. Particularly in Android, they are named as { azimuth, pitch, roll }. Second, the pose information also includes the distance between the VO and the camera, i.e., the value of VO_{dist} set by the user when confirming the location of the VO.

Above all, the RO information and the phone's pose information will be packed into a package file (referring to Fig. 4), and delivered to VO viewers.

B. VO Viewer: View a VO

Once a VO viewer receives the package file, her or his phone can load the data and start to recognize the ROs. The user interface is essentially a camera view that keeps capturing real-time frames. Each frame will be handled by the following steps. (1) We recognize the objects in the frame, and if there exist potential matches of the ROs, we further extract the pose of the phone. (2) We apply a pose-assisted feature matching algorithm to determine if the ROs are in the frame. (3) If there are recognized ROs in the frame, we restore the location of the VO and render the AR scene. Fig. 6 illustrates the architecture for VO reviewers, where the solid lines and dashed lines indicate the control flow and data flow respectively.

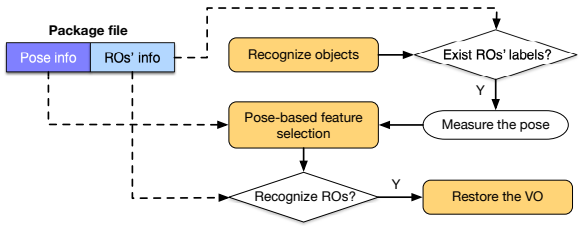

Fig. 6. Program architecture for VO viewers

1) Recognize objects and measure pose: This step is similar to the first step of the OV owner. We use Tensorflow's module to recognize the objects in the frame, and obtain a label and rectangle region for each object. We first compare the labels with the RO's labels in the package file. If there exist label matches, we will measure the phone's pose preparing for the feature matching. Recall that the pose information includes the 3-axis orientation data and the distance to the VO. The orientation data can be obtained from the phone, but it is not straightforward to derive the distance to the VO. PROMAR considers the area of the rectangle region of the recognized object in the viewer's frame, indicated by ARV, and compare to the area of the rectangle region of the matching RO from the VO owner, indicated by ARO. We use a $scale$ to denote the scaling ratio between the viewer's view and owner's view,

$$scale = ARV/ARO. \tag{4}$$

We assume the area of the same object in the frame is inversely proportional to the distance between the camera and the VO. Since we know that the distance between the VO and the owner's phone is VO_{dist} (recorded in the package file), we can estimate the distance between the VO and the viewer's phone as $VO'_{dist} = \frac{VO_{dist}}{scale}$. The pose information including 4 values will be passed to the next step.

2) Pose-assisted feature comparison: The core component is to apply feature matching and determine if the ROs exist in the viewer's frame. In PROMAR, for each object with a matching label, the feature comparison includes two steps: (1) select the FPs of the matching RO from the package file, (2) compare with the FPs extracted from the recognized object.

FP selection: First, for each RO with a matching label found in the viewer's frame, we aim to select N FPs from the package file to represent the robust FPs for feature comparison.

Comparing the VO viewer's and VO owner's poses, if only one of the four properties has changed, we can just pick the robust FPs for that distortion type. However, in practice, it is very common to have multiple distortions involved in the viewpoint change. Our algorithm needs to select FPs from multiple lists of robust FPs to construct a set of N FPs. Intuitively, if the change of one distortion type is more significant, we would prefer selecting more FPs from its list.

In PROMAR, we first assign a weight value (w_i) to each involved distortion type to indicate its significance. We normalize w_i into $[0, 1]$ by considering a *working range* for each properties. Basically, we set an upper bound for the value change for each property in the pose information. If the change reaches or exceeds the upper bound, its weight is 1, otherwise the weight value is proportional to the change. In particular, we set $\pm 45^\circ$ to be the working range for the 3-axis orientation readings (i.e., horizontal angle, vertical angle, and rotation), and $[0.2, 1.8]$ to be the range for scale. Therefore, the weight value for each angle distortion is $\min\{\frac{|diff_{angle}|}{45}, 1\}$, and the weight for scale is $\frac{|scale-1|}{0.8}$, where $scale$ is from Eq. 4.

We develop Algorithm 2 to select N FPs so that the number of FPs chosen from each distortion type is proportional to its weight value. We use RET to hold the selected FPs, and the program terminates when N FPs are chosen. In addition, we use variable c_i as a counter recording the number of FPs selected from the list for distortion type D_i. In every round, we first calculate the *deficit* of each distortion type according to its weight value and current counter value (lines 3–6). The distortion type with the maximum deficit is recorded (idx records its index), and we select a FP from its FP list OF_{idx}. In lines 7–8, nFP is the FP we select, and p_i points to the head of each robust FP list OF_i. Finally, we update the counter values in lines 9–11. The same FP may appear in multiple FP lists. Once it is selected, we increase the counters of all the FP lists containing it. We also remove the selected FP from all these lists because it is no longer a candidate.

FP matching: After selecting N FPs of a RO from the package file, the VO viewer also extract N FPs from the matching object in its own frame, and applies feature matching process. In PROMAR, we develop a new matching algorithm with two steps, strict matching and loose matching. We first use regular matching algorithm to find a set of matches, i.e., $i \rightarrow j$ to indicate that the FP $i \in OF$ can find a similar FP j in the matching object in the VO viewer's frame. In the first round of strict matching, we enforce strong constraints and aim to find a small subset of matches that are very likely to be true matches. And then, we use these matches to derive a coordinate mapping between the two object images. In the second round of loose matching, we relax the constraints on the image properties for a match $i \rightarrow j$, if i and j's coordinates fit in the mapping we derive.

Algorithm 2: FP Selection: FPS(OF, w, N)

```
1  RET = {} ;
2  while |RET| < N do
3      for OF_i do
4          deficit = c_i/sum(c) − w_i;
5          if deficit > max then
6              max = deficit,  idx = i;
7      nFP = OF_idx[p_idx ++];
8      RET = RET ∪ nFP;
9      for OF_i do
10         if nFP ∈ OF_i then
11             c_i ++,  remove nFP from OF_i;
12 return RET;
```

1. Strict matching: Regular feature matching algorithms are often inaccurate including false matches in the results. In this step, we intend to strictly examine the matches and only keep the matches with high confidence to be true. Besides the typical techniques, we add two new processes here. The first is random sample consensus (RANSAC), which is commonly used to eliminate outliers. In the second process, we set a threshold for the *distance* value of the matches. It is a built-in property from the feature matching algorithm measuring the distance of the two multi-dimensional FPs in a match. The distance value approximately indicates the similarity of the two FPs. In this round of strict matching, we eliminate all the matches whose distance values are greater than our threshold. For the remaining matches $i \rightarrow j$, we apply linear regression on FP i and j's coordinates in their images. Our intuition is that if the object recognized by the VO viewer is actually the RO observed by the VO owner, their FPs should be located in a similar way even with a moderate viewpoint change. The result of the linear regression represents a mapping correlation of the coordinates between the two objects.

2. Loose matching: In this round of matching, we examine the original set of matches again. We relax the distance threshold, and check the coordinates mapping in each match. If the mapping is close to our linear regression's result, we will consider it as a match.

Eventually, the matching ratio for this RO is $\frac{|M|}{N}$, M is the set of matches selected after these matching processes.

3) RO-based VO restoration: In the last step, if the VO viewer recognizes at least one ROs, we will restore the VO based on the ROs' information. With the relative location information in the package file and the coordinates mapping correlation, it is straightforward to derive the VO's pixel coordinates in the current frame. However, the AR rendering system require 3-D coordinates in the real world coordination system in the unit of physical distance such as meter.

In PROMAR, we apply a transformation process that can convert pixel distance to physical distance. Due to the page

limit, we use an example in the following Fig. 7 to briefly illustrate the basic steps.

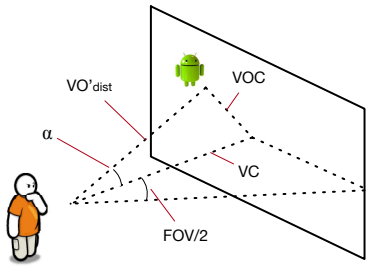

Fig. 7. Restore the VO based on an RO

Assume that we have derived the pixel coordinates of the VO, and the frame represents a virtual panel facing the viewer. We use VC to indicate the distance between the viewer and the center of the frame, and VOC denotes the distance between the VO and the center. In our solution, we use a physical property of the camera, called field of view (FOV). This property defines the view angles the camera supports. The FOV includes two values, one for horizontal angle, and the other for vertical angles. We can retrieve FOV information via Android APIs, and in Fig. 7, we show an angle of $\frac{FOV}{2}$ referring to the horizontal FOV. Since we know the dimension of the frame, the pixel value of VC is calculated as

$$VC = \frac{W}{2} / \tan(\frac{FOV}{2}),$$

where W is the width of the frame. As VOC can be easily calculated in the unit of pixel, we can derive the angle α as $\tan^{-1}(VOC/VC)$. The physical distance between the viewer and the VO has been estimated as VO'_{dist} according to VO_{dist} from the owner and $scale$ derived by the viewer. Thus, we can calculate the physical distance of VC as $VO'_{dist} \cdot \cos(\alpha)$. At this point, we have both pixel and physical distance for VC. Then we can calculate a conversion ratio, and further derive the 3-D physical coordinates for the VO.

V. Performance Evaluation

In this section, we present our performance evaluation. We have implemented PROMAR in Java and incorporated it with Android ARCore [1]. We have built a library for the main functions in PROMAR that can be imported by regular Java programs. The library includes about 5,000 lines of codes. In addition, we created a mobile application based on the ARCore's and Tensorflow's samples [43], [44] with additional 2,000 lines of codes plus adjusted PROMAR Android library. Our implementation incorporates libraries of OpenCV [42], Tensorflow Lite [44], and ARCore. Our experiments are conducted on an OnePlus 7 phone.

A. Settings and Workload

We mainly focus on three performance metrics, (1) accuracy of recognizing the scene, (2) accuracy of the VO restoration, and (3) runtime overhead. In our experiments, we use the ORB implementation in OpenCV [42] as the image feature detector and descriptor. When evaluating the scene recognition, the following image datasets are used.

ALOI [41]: It is a color image collection of one-thousand small objects. For each object, it offers the pictures taken at varying view angles. It represents the test cases with a single object and a single type of distortion (horizontal angle).
BigBIRD [45]: It contains object images taken by 5 cameras fixed on different heights, each neighbor camera generates photos with vertical view angle difference as around 22°. It represents the test cases with a single object and multiple (two) types of distortions (horizontal and vertical angles).
False images: Finally, we download 100 images from the Google Images using the corresponding object title as keyword. They form a dataset of false images for the target object.

B. Accuracy of scene recognition

In this subsection, we evaluate the accuracy of scene recognition in the VO viewer's module. In PROMAR, it refers to the accuracy of recognizing the same ROs. We examine the FP matching ratios, and the precision/recall rate given a threshold τ for the matching ratio. The precision rate is defined as $\frac{TP}{TP+FP}$, recall rate as $\frac{TP}{TP+FN}$, where TP, FP, TN and FN represent true positive, false positive, true negative, and false negative respectively. We present the results for the following three cases: single object/single distortion in image datasets, single object/multiple distortions in image datasets, and mobile phone experiments with two ROs.
Single Object/Single Distortion. Due to the page limit, we only present the results of a single distortion on horizontal angle with 10 objects in ALOI. For each image of an object, we regard it as the owner's saved data and use the other images of the same object as the viewer's image to compare with the owner's image. Specifically, we extract 100 robust FPs from the owner's image in corresponding to the view angle difference between the viewer and owner and 500 regular ORB FPs from the viewer's image, after which our matching is applied. The average matching ratios are listed in Table. I.

TABLE I
Average matching ratio with view angle change

Angle change	±5	±10	±15	±20	±25	±30	±35	±40	false
Lego man	0.65	0.51	0.43	0.37	0.34	0.32	0.28	0.26	0.13
Shoe	0.69	0.51	0.42	0.34	0.27	0.23	0.20	0.18	0.12
Furry elephant	0.68	0.47	0.35	0.27	0.20	0.19	0.15	0.12	0.11
Toy bear	0.71	0.57	0.48	0.42	0.36	0.32	0.27	0.25	0.20
Van Gogh	0.78	0.71	0.65	0.57	0.49	0.45	0.43	0.39	0.21
Furry bear	0.63	0.44	0.33	0.24	0.19	0.13	0.14	0.12	0.11
Duck cup	0.72	0.61	0.49	0.43	0.40	0.36	0.32	0.30	0.21
Furry dog	0.65	0.49	0.41	0.36	0.33	0.34	0.32	0.32	0.18
Baby cream	0.73	0.62	0.49	0.44	0.36	0.32	0.31	0.28	0.25
Girl statue	0.68	0.53	0.40	0.33	0.30	0.27	0.25	0.23	0.12
Average	**0.69**	**0.54**	**0.44**	**0.37**	**0.32**	**0.29**	**0.26**	**0.24**	**0.16**

We observe that within a reasonable range of the angle change, e.g., ±25°, PROMAR works very well yielding a wide gap between the matching ratios of true images and those of false images. In some cases, our algorithms can distinguish them even when the angle change reaches ±40°.

In Fig. 8 we plot the precision and recall rate in accordance with the changing threshold. We compare PROMAR to is a combination of OpenCV's ORB, knnMatcher, ratio test, cross-checking, and RANSAC test, denoted as RANSAC. This combo has been empirically proved to be effective and superior

to other built-in matchers. As illustrated in Fig. 8, the precision of PROMAR is above 0.98 while the recall rate is higher than 0.85 in all the tested cases. It represents an extremely high recognition accuracy that can recognizes most of the true cases and eliminates the false images.

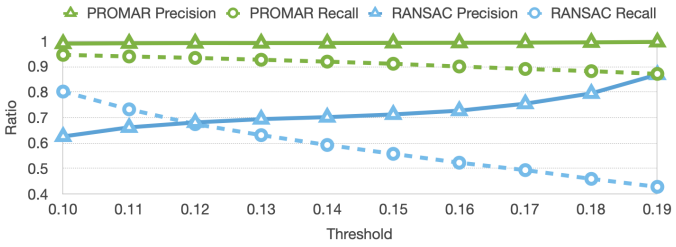

Fig. 8. Precision and recall rate obtained by PROMAR and RANSAC with different thresholds.

Single Object/Multiple Distortion. Next, we present the evaluation results of two cases of multiple distortions: horizontal + vertical angle change, and scale + horizontal angle change. Horizontal + Vertical angle change. BigBIRD dataset provides 5 groups of images spaced horizontally by 3° and for each group, they are vertically spaced by ∼22°. We exploit 3 groups of this dataset to test the performance of multiple (two) distortion types and illustrate the results in Fig. 9. The axis x represents the varying horizontal angle difference between the owner's and viewer's images, and the vertical view angle difference is fixed as 22°in this experiment.

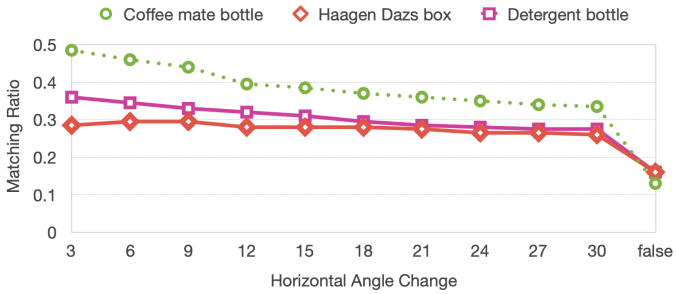

Fig. 9. Matching ratio with two types of distortions (vertical angle change is fixed at 22°)

The results show that PROMAR is still quite effective as the matching ratios of false images are obviously lower than those of true images in Fig. 9. When we set the recognition threshold as 0.15, the precision and recall rate are 0.91 and 0.98 respectively in this case.

Scale + Horizontal angle change. In another test setting for multiple distortions, we pick 10 objects in ALOI which already includes the images with horizontal angle changes. For every image, we additionally generate a set of new images by manipulating the scale from 30% to 95%. After the scaling process, we use images of 65% scale as owner's image and compare it with images on all scales whose horizontal angle differences are less than 40°. The results are illustrated in Fig. 10, where the horizontal angle change lies on the x-axis, scale change on y-axis, and z-axis values indicate the matching ratios. For the simplicity of the plot, we do not include the matching ratios of false images whose average value is 0.17.

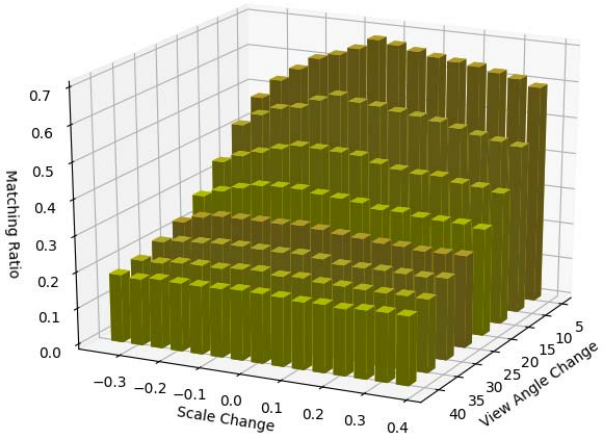

Fig. 10. Matching ratio with two types of distortions (scale and horizontal angle).

We can observe that for all cases, including the worst case where scale change is 35% and view angle change is 40°, the matching ratios are adequately higher than the average matching ratio of false images. It would be straightforward to distinguish true images from false images with moderate viewpoint changes. We calculate the precision and recall rates assuming a recognition threshold of 0.15, and their values are 0.92 and 0.73 respectively. The results are sufficiently accurate for moderate distortions in two domains.

Mobile Phone Experiments. Finally, we deploy our on an Android phone and conduct experiments in a realistic setting. Specifically, we place virtual objects on a laptop, chair, and bed. Then we play the role of a VO viewer and randomly move the camera around the objects for 10 minutes. During the process, we also point the camera to different laptops, chairs and beds (include the original ones). We develop a logger running on the phone that records the real-time matching ratios as well as the distortion types involved for each frame processed by PROMAR. Overall we record more than 300 frames that can be recognized by TensorFlow, and their matching ratios are illustrated in Fig. 11. The axis z shows the matching ratio, while axis x and y indicate the horizontal and vertical angle change respectively, and the color represents the absolute value of scale change (illustrated on the color bar).

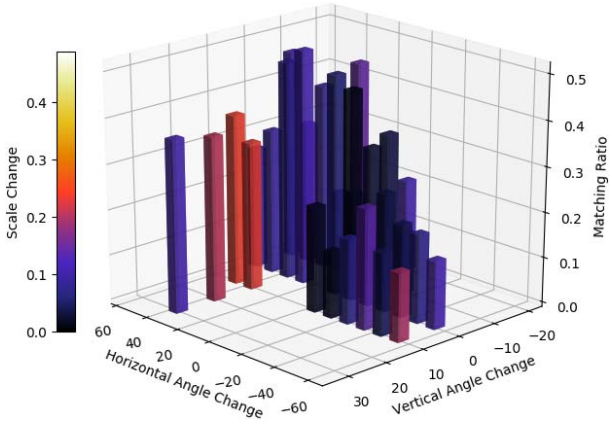

Fig. 11. Matching ratio in experiments on a mobile phone.

With all three distortion types involved, the average of

matching ratio in this test is 0.28 which is still consistent with our previous evaluation with image data sets. We further plot the recall and precision rate with different recognition thresholds in Fig. 12. The precision is as high as 0.98 when setting the threshold as 0.15, which means almost all false cases are filtered out. In the meantime, the recall rate is more than 0.7, i.e., most of the positive cases are correctly recognized. These results confirm that PROMAR can achieve remarkable accuracy performance in real world tests, even when the viewpoint distortion is significant, e.g., shrinking size to about half or changing view angle for more than 45°.

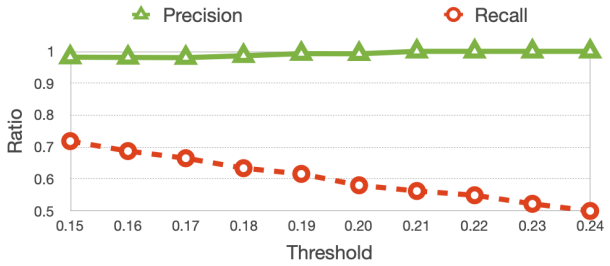

Fig. 12. Precision and recall rate obtained by PROMAR on a mobile phone with different thresholds.

C. Accuracy of VO restoration

The other important metric is the accuracy of the restored VOs, including their location and size. Since it is a quite subjective metric, we design an anonymous survey and recruit 75 volunteers on social media to fill it after watching the our demo video clips of placing and restoring VOs with PROMAR. We also show a demo video of ARCore [43] with the same functions (placing and restoring VOs on a plane surface) as a comparison, especially for the drifting issues existing in the original ARCore framework.

In the clips, we use PROMAR to place VOs in two different scenes (with a laptop and bike as the RO respectively), and then PROMAR restores the VOs with different view angles and distances. For testing the framework in complex scenarios, we also place the VO on a desk with multiple objects (a laptop, two monitors, lines, a keyboard, and shell). The survey includes 5 questions asking users' subjective perception on the location and the size of restored VO comparing to the original VO, the comparison between PROMAR and ARCore and the background knowledge of the respondent on AR/VR. We draw the response data in Fig. 13.

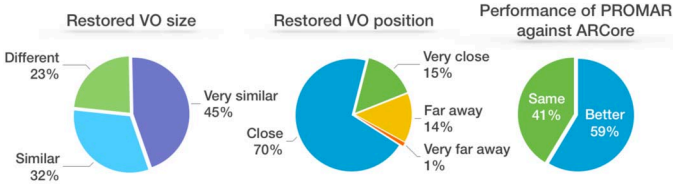

Fig. 13. The first two plots are users' perception of the restored VO's position and size. The last plot is the users' comparison PROMAR and ARCore

According to the collected data, most respondents think PROMAR can reproduce the virtual object on a near position (85%) with similar size (74%). In comparison with ARCore, 59% of the users consider PROMAR's performance is better than ARCore, and the rest of them think PROMAR's accuracy is as good as ARCore. None of the respondents think that PROMAR is inferior to ARCore. The survey results are substantial evidences that PROMAR has significantly improve the multi-user AR experiences in a practical setting.

D. Runtime overhead

The last metric we examine is the runtime overhead which is critical for real time mobile AR applications. We conduct experiments on a OnePlus 7 phone equipped with a Snapdragon 855 processor and 8GB RAM and running Android 10 system. We measure the time overhead of each key step in our system, and the average values of 50 independent trials are illustrated in Fig 14.

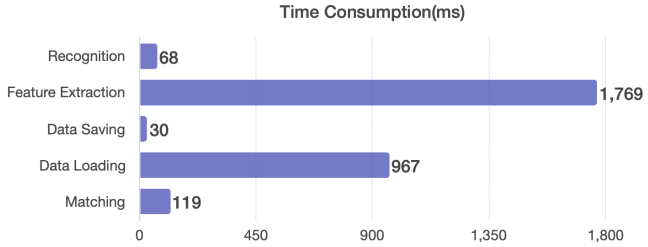

Fig. 14. The average time consumption of each step

'Recognition' refers to the process of detecting the ROs in a frame which is the first step for both VO owner and viewers. PROMAR uses Tensorflow Lite in this step and it takes 68ms to finish on average. For the VO owner, the next step is to apply distortion-based tests and save a set of robust FPs. This process is the most time-consuming step (1799ms on average). But it is a one-time operation for the VO owner and can be conducted in the background.

Our main focus is the run-time processing rate for VO viewers because they need to keep capturing frames from a camera view and process those frames at a high rate. In PROMAR, the VO viewers need to first load the package file delivered by the VO owner taking 967ms on average in our experiments. This step is a one-time operation for the viewers. Then, the VO viewers will recognize the objects in each frame (68ms), and then apply our matching algorithm (119ms). In total, it takes 187ms to process one frame. These experimental results indicate that PROMAR is quite efficient as it can process more than 5 frames per second in such a complex application setting.

In conclusion, our evaluation has validated that with the novel techniques presented in this paper, PROMAR is a practical and effective multi-user AR framework.

VI. Conclusion

In this paper, we develop a framework that supports multi-user AR applications. We present novel algorithms for image feature extraction and feature matching process. Our design is based on reference objects combining image features with the mobile phone's pose information. We have implemented the entire system and evaluated with a prototype and experiments. The results show that our system is effective and practical.

REFERENCES

[1] "Google ARCore," https://developers.google.com/ar/.

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

[3] "Vuforia," https://www.vuforia.com/.

[4] "Wikitude," https://www.wikitude.com/.

[5] Y. Xiang and S. Savarese, "Estimating the aspect layout of object categories," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3410–3417.

[6] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[7] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[8] B. Leibe, A. Leonardis, and B. Schiele, "Combined object categorization and segmentation with an implicit shape model," in *Workshop on statistical learning in computer vision, ECCV*, vol. 2, no. 5, 2004, p. 7.

[9] H. Su, M. Sun, L. Fei-Fei, and S. Savarese, "Learning a dense multi-view representation for detection, viewpoint classification and synthesis of object categories," in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 213–220.

[10] C. Gu and X. Ren, "Discriminative mixture-of-templates for viewpoint classification," in *European Conference on Computer Vision*. Springer, 2010, pp. 408–421.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[12] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016. [Online]. Available: http://arxiv.org/abs/1602.07261

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Computer Vision (ICCV), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2980–2988.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[17] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[18] E. Oyallon and J. Rabin, "An analysis of the surf method," *Image Processing On Line*, vol. 5, pp. 176–218, 2015.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, 2011, pp. 2564–2571.

[20] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

[21] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*. Springer, 2010, pp. 778–792.

[22] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "Kaze features," in *European Conference on Computer Vision*. Springer, 2012, pp. 214–227.

[23] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 IEEE international conference on computer vision (ICCV)*. Ieee, 2011, pp. 2548–2555.

[24] "Stereo camera wiki," https://en.wikipedia.org/wiki/Stereo_camera.

[25] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with microsoft kinect sensor: A review," *IEEE transactions on cybernetics*, vol. 43, no. 5, pp. 1318–1334, 2013.

[26] S. Milani and G. Calvagno, "Joint denoising and interpolation of depth maps for ms kinect sensors," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 797–800.

[27] M. Schmeing and X. Jiang, "Color segmentation based depth image filtering," in *International Workshop on Depth Image Analysis and Applications*. Springer, 2012, pp. 68–77.

[28] F. Qi, J. Han, P. Wang, G. Shi, and F. Li, "Structure guided fusion for depth map inpainting," *Pattern Recognition Letters*, vol. 34, no. 1, pp. 70–76, 2013.

[29] S. B. Gokturk, H. Yalcin, and C. Bamji, "A time-of-flight depth sensor-system description, issues and solutions," in *2004 Conference on Computer Vision and Pattern Recognition Workshop*. IEEE, 2004, pp. 35–35.

[30] A. Oliva and P. G. Schyns, "Diagnostic colors mediate scene recognition," *Cognitive psychology*, vol. 41, no. 2, pp. 176–210, 2000.

[31] L. Fei-Fei and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 524–531.

[32] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "Netvlad: Cnn architecture for weakly supervised place recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5297–5307.

[33] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in neural information processing systems*, 2014, pp. 487–495.

[34] Y. Yuan, L. Mou, and X. Lu, "Scene recognition by manifold regularized deep learning architecture," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 10, pp. 2222–2233, 2015.

[35] J. Huang, D. Millman, M. Quigley, D. Stavens, S. Thrun, and A. Aggarwal, "Efficient, generalized indoor wifi graphslam," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1038–1043.

[36] B. Ferris, D. Fox, and N. D. Lawrence, "Wifi-slam using gaussian process latent variable models." in *IJCAI*, vol. 7, no. 1, 2007, pp. 2480–2485.

[37] H. Abdelnasser, R. Mohamed, A. Elgohary, M. F. Alzantot, H. Wang, S. Sen, R. R. Choudhury, and M. Youssef, "Semanticslam: Using environment landmarks for unsupervised indoor localization," *IEEE Transactions on Mobile Computing*, vol. 15, no. 7, pp. 1770–1782, 2015.

[38] Y. Chen, Q. Yang, J. Yin, and X. Chai, "Power-efficient access-point selection for indoor location estimation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 7, pp. 877–888, 2006.

[39] D. Han, S. Jung, M. Lee, and G. Yoon, "Building a practical wi-fi-based indoor navigation system," *IEEE Pervasive Computing*, vol. 13, no. 2, pp. 72–79, 2014.

[40] C. Wu, J. Xu, Z. Yang, N. D. Lane, and Z. Yin, "Gain without pain: Accurate wifi-based localization using fingerprint spatial gradient," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, p. 29, 2017.

[41] J.-M. Geusebroek, G. J. Burghouts, and A. W. Smeulders, "The amsterdam library of object images," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 103–112, 2005.

[42] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[43] "ARCore sample application: hellosceneform," https://github.com/google-ar/sceneform-android-sdk/tree/master/samples/hellosceneform.

[44] "Tensorflow object detection," https://github.com/tensorflow/models/tree/master/research/object_detection.

[45] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel, "Bigbird: A large-scale 3d database of object instances," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 509–516.