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

ABSTRACT

Augmented reality (AR) is an emerging technology that can weave virtual objects into physical environments, and enable users to interact with them through viewing devices. This paper targets on multi-user AR applications, where virtual objects (VO) 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 can help the viewers accurately identify the same scene tolerating the view angle differences. 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.

1. INTRODUCTION

Smartphones and their apps have dramatically changed the way people communicate and improved our daily life from various perspectives. 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 [3], ARKit [2], Vuforia [5], and Wikitude [6].

In this paper, we target on a framework that supports multi-users interactions for the MAR apps, where one user places a virtual object that can be recognized by other users. Multi-users 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 virtual object submitted with the repair request, and people can play the scavenger hunt game to find the virtual objects placed by other users. In addition, 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 3. Most importantly, we expect the target system to tolerate moderate viewpoint difference between the the user who places the virtual object and the user who views the object. This is a common setting in realistic scenarios, but leads to significant research challenges in system design.

The current AR systems, however, are mostly designed for a single user. They often require additional processes to effectively work in practice, such as preloading high quality 'tag' images, or have constraints on the virtual object placement, e.g., ARCore [3] 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 the detection accuracy is quite low, especially 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. Our source codes and demos can be accessed at https://github.com/PROMAR2019 .

2. RELATED WORK

Mobile augmented reality has been an extensively exploited field for a long time, many companies have released powerful frameworks to free developer's concern on low level implementation, like ARKit 2 [2], ARCore [3], Wikitude [6], Vuforia [5] to name a few. However, to our best knowledge, all existent frameworks suffer from inconsistency, insufficiency of computing resources or have special limitations on practical application. The latest release of ARCore and ARKit announced to support multiple-user AR, but their prerequisite of detecting plane restricted the deployment. To achieve a generally usable AR, the potential bottleneck and promising breakthrough fall in below categories:

Object detection. Multiple different methods, like [13, 16, 19, 24, 41, 44], were used to categorize an object from an input image with given object category. Starting from [23], deep neural networks dominated this field. The later improved neural networks such as [20, 21, 34, 38, 42] could achieve impressive precision even higher than human performance. Even so, because the model training required a massive amount of image data, they fell short of recognizing exact target object in AR scenery, which is usually unable to provide sufficient training data. The traditional image vision methods, such as SIFT [28], SURF [33], ORB [36], and others [8, 11, 25, 35], suffer from the lack of computability or inconsistent performance with changes of viewpoint.

Tracking. One of the most fundamental techniques laying the groundwork for AR applications is tracking. The latest popular tracking technologies can be separated into the following three research aspects: (1) Simultaneous Localization and Mapping: SLAM refers to the techniques of trying to simultaneously localize some sensors with respect to its surroundings, while at the same time mapping the structure of that environment. The history of SLAM can be dated back to 1986 paper [40]. Recent studies [14, 17, 22, 27, 43] taking the advantage of developed visual sensors successfully achieve this task by using only visual input and processing data with sophisticated algorithms. The typical limitations for SLAM come from high computational cost and potential tracking loss caused by fast camera motions. (2) RGB-D Data and Reconstruction: With the maturing of cheap and widely accessible RGB-D camera, many researchers manage to utilize dense depth maps generated by the sensors to solve tracking problem. Prior work [15, 29, 32, 37] presented this approach to map and track complex environment accurately. This method cannot be applicable in our problem as commercial mobile phones are not quipped with RGB-D sensors. (3) Hybrid Tracking and Mobile Platforms: This approach combines both visual-based and sensor-based tracking to achieve low-latency tracking with high precision and accuracy. Prior work [9, 12, 26, 31] exploited camera alongside other sensors such as GPS, and accelerators to address tracking problem.

3. SYSTEM MODEL AND MOTIVATION

We first present our problem formulation and some challenges the existing solutions cannot address.

3.1 Problem Formulation

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. The following Fig.1 illustrates an example.

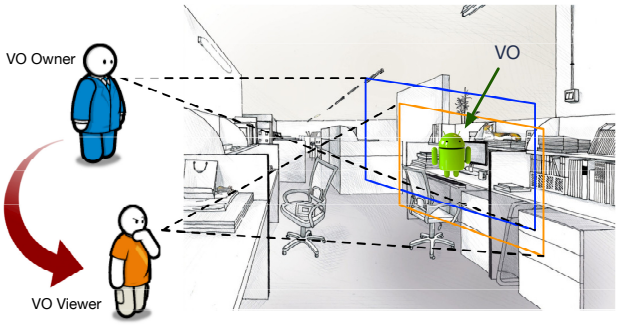

Figure 1: An application example

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

Hardware requirements. Our target applications are based on common off-the-shelf mobile devices with regular cameras and on-board sensors. 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

([2,3]) or attached to a particular object whose image has to be uploaded and processed beforehand ([5,6]).

Viewpoint difference. In a realistic setting, our system must tolerate the viewpoint difference of the VO owner and viewers. As illustrated in Fig.1, the VO owner and viewers may view the VO and its surrounding scene from different angles and distances. We expect the VO viewers to be able to view the VO if the viewpoint difference is within a certain range.

Runtime overhead. Our target system 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.

3.2 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.

3.2.1 Recognition accuracy

The first issue is the accuracy of recognizing the extract scene with changes of viewpoint. We have conducted experiments to examine the performance of the traditional approaches in our application scope. We use the images from **ALOI** [18] 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 [36] algorithm. We 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 measure the matching ratio as the metric of accuracy, i.e., the ratio of the matched FPs and the total 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 [10] library). For each object, we enumerate all its images as the VO owner's image, and report the average matching ratios 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.

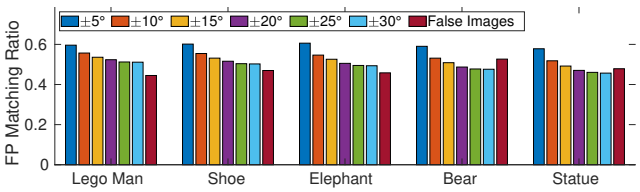

Figure 2: Matching ratios with change of view angles

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. It indicates that either the matching ratios of true images are too low or those of false images are too high in the traditional feature matching algorithms.

3.2.2 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 Moto G6 Android phone based on OpenCV library. We choose ORB [36] feature extraction algorithm in our experiments, and the algorithm allows us to configure a parameter to indicate the number of feature points 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 feature points from both of them for comparison. For example, the first bar in Fig. 3 indicates the comparison of two sets of 100 FPs. The results in the figure are the average values of 100 independent runs for each setting.

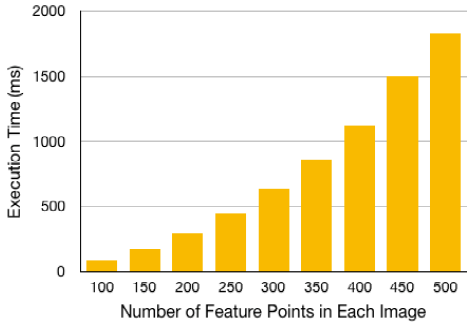

Figure 3: Execution time of comparing two images' feature points measured on an Android phone.

We observe that the computation overhead exponentially increases along the increase of the number of FPs.

When the VO viewer keeps capturing frames and process them at runtime, we can only use a small set of FPs. In our default setting, we only consider 100 FPs for each frame.

4. DESIGN OF PROMAR

In this section, we present the design details of PROMAR. The major objective of the system is to allow the VO viewers to recognize the exact same scene where the VO owner places the VO while their view angles and distances to the scene may be slightly different. Our solution is based on the objects that can be recognized in the scene. We call them reference objects (ROs). In stead 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.

In this section, we first present an overview of our solution, then describe the detailed processes for the VO owner and viewers.

4.1 Sketch of the framework

We first briefly describe the basic steps for both VO owner and viewer, and define *matching ratio* which is an important metric in our system.

VO owner: The main process for a VO owner is to place a VO in a target frame. Through the phone's camera view, the VO owner selects a desired position to place the VO that has been imported as a 3D asset. In our default setting, the VO is placed in the center of the frame. The VO owner can further adjust the size of the VO to roughly indicate the distance. It is an approximated estimation as the regular camera does not possess the depth information. Once the VO owner confirms the position and size of the VO, it will be rendered via the AR platform. In particular, PROMAR is built upon Google ARCore [3]. Then, PROMAR uses the following three steps to prepare a package file for the VO viewers. **(1) Identify ROs:** In our design, the VO is bonded with the objects that can be recognized in the same target frame. We call them reference objects (RO). This step is processed by a regular object detection system such as Tensorflow [4] or YOLO [7] that analyzes the captured target frame. We also records each RO's relative position (pixel coordinates) to the VO on the target frame. **(2) Extract image features:** This is the core step in the VO owner's process. For each RO, PROMAR applies a novel algorithm to extract the image features. These features will later be used by the VO viewers to identify the RO. **(3) Record phone's pose:** Besides the image features, PROMAR also records the *pose* information of the phone, that essentially defines the camera view angle and distance towards a target object. We will present more details in the next subsection. This information is derived from the readings of the onboard sensors such as accelerometer, gyroscope, and magnetometer.

All these pieces of information are stored in a package file that will be delivered to the VO viewers.

VO viewer: In PROMAR, a VO viewer receives VO package information from the VO owner. Through her or his phone's camera view, the viewer keeps capturing image frames, and conduct the following steps for each frame. **(1) Identify the objects in it:** This step is the same as identifying ROs in the VO owner's process. After obtaining the recognized objects, the viewer compares their labels with the recorded ROs'. If there is no match, the VO viewer discards this frame. **(2) Obtain phone's pose:** If there exist matching ROs, the viewer will derive the phone's pose information including the view angles and the distance between the phone and recognized ROs. **(3) Extract image features and conduct featuring matching:** In this step, the viewer conducts feature matching for each recognized RO. Based on the viewer phone's pose information, the viewer will select a set of features from the VO owner's package, and compare to the features extracted from viewer's frame. **(4) Restore the VO based on ROs:** If the feature matching for a RO is positive, the viewer will derive the VO's position on its frame according to the relative position information in the package file, and render the VO via the AR platform.

Matching ratio: The core technical solution in PROMAR is identifying a reference object. 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 reference object, and outputs a set of feature points (OF) in the area as well as a guideline threshold for matching ratio (τ). 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,

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

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 the threshold parameter,

[1]Note that this matching relation is directional.

i.e., $MR > \tau$.

4.2 VO Owner: Place a VO

In PROMAR, a user places a VO through camera view with an existing AR framework such as Android ARCore [3]. We have modified the framework so that the VO can be placed in an arbitrary location rather than on a recognizable plane in the default configuration. With such a setting, the underlying AR rendering framework 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 direct distance between the camera and the VO. We use one meter as the default value, and provide a seekbar for the user to adjust the value. We use VO_{dist} to represent this value. Once the user confirms the VO's location, we will record the value of VO_{dist}.

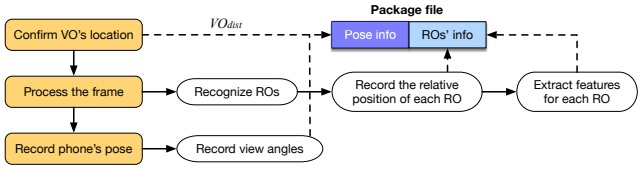

Figure 4: Program architecture for VO owners

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 its 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.

4.2.1 Recognize reference objects (ROs)

In this first step, we recognize the objects in the captured frame, and use them as references to help the viewer restore the AR scene. In our system, we use Tensorflow's object detection module to identify the objects. For each recognizable object, a label of the object and the detection confidence are reported, as well as a rectangle region surrounding the object.

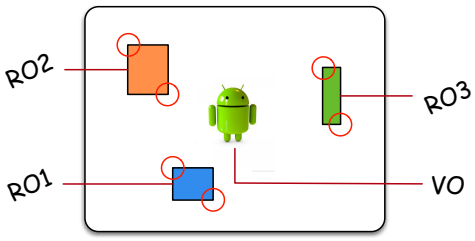

Figure 5: Identify reference objects (ROs)

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. Recall that the VO is placed in the center of the frame, and its pixel coordinates are $(\frac{W}{2}, \frac{H}{2})$, where W and H are the width and height of the frame. Fig. 5 illustrates an example of placing a VO in a frame with three ROs.

4.2.2 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 3, this change could dramatically affects the matching ratio values. Referring to Eq. 1, some FPs in the original OF may not even exist in the VF yielding a low matching ratio.

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.

Definition of 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. In PROMAR, 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. 6. We first use regular feature extraction algorithms such as ORB [36] 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 mimicking the viewpoint change within a certain range. For ex-

ample, for 'rotation: clockwise' distortion, we rotate the original RO image clockwise with a degree ranging from 0° to 30° with an interval of 3°. 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. 6 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).

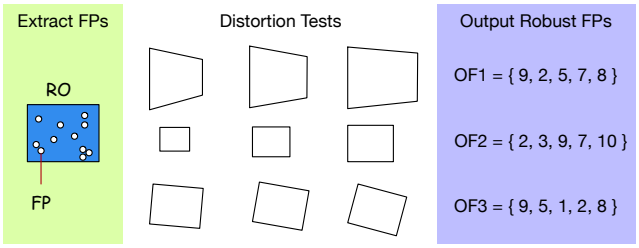

Figure 6: Distortion-based feature extraction

Robust FP selection. The problem of selecting robust FPs can be generally 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 counterpart can be found) in each distorted image. We use $x(i, j, k)$ to denote the matching results,

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

Considering the feature matching performance on the VO viewers, the number of FPs the VO owner supplies is limited to a system parameter N. Thus, our goal is to select N most robust FPs for each type of distortion.

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. Eventually, the recognition result is based on the comparison of the matching ratio and a threshold. Solving this max-min problem can increase the chance for the worst-case distorted RO image in this distortion type.

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          end
8      end
9      RET = RET ∪ idx,  C = C − idx, minS = {};
10     for k ∈ TC_j do
11         c = Σ_{i∈RET} x(i, j, k);
12         if c < min then
13             min = c, minS = {k};
14         else if c == min then
15             minS = minS ∪ k;
16     end
17 end
18 return RET;
```

In PROMAR, we develop the following Algorithm 1 to select the robust FPs 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, $minS$ include 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 2–17) that terminates when RET holds N FPs. In each round, the algorithm enumerates all the candidates (lines 3–8), and pick 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 9, the algorithm updates the set RET and C. The minimum set $minS$ is reconstructed in lines 10–16 based on the updated RET.

4.2.3 *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 that 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 axises. Particularly in Android, they are named as { azimuth, pitch, roll }. Second, the pose information 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) delivered to VO viewers.

4.3 VO Viewer: View a VO

Once a VO viewer receives the package file, her or his phone can load the data and starts 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 RO in the frame, we restore the location of the VO and render the AR scene. Fig. 7 illustrates the architecture for VO reviewers, where the solid lines and dashed lines indicate the control flow and data flow respectively. In the rest of this subsection, we present the details of each step.

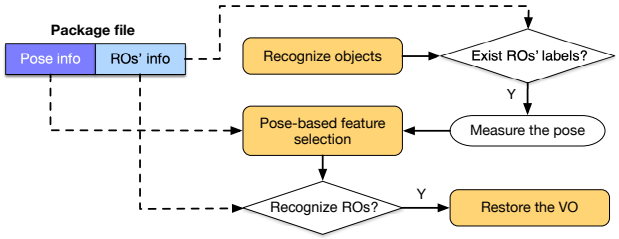

Figure 7: Program architecture for VO viewers

4.3.1 *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 is no match between the two lists, we will immediately discard the frame. If there exist label matches, we will measure the phone's pose preparing for the feature matching algorithm. Recall that the pose information includes the 3-axis orientation data and the distance to the VO. The orientation data can be obtained directly from Android's APIs. But it is not straightforward to derive the distance to the VO. In PROMAR, we consider 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,

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

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

4.3.2 *Pose-assisted feature comparison*

The core component for the VO viewer is to apply feature matching and determine if the ROs exist in the viewer's frame. Matching labels in the previous step is only a quick filter. Image feature comparison is our main approach to make the decision. 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: In the first step, 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. The parameter N is determined by considering the run-time performance of feature matching algorithm.

Comparing the VO viewer's and VO owner's posees, if only one of the four properties has changed, we can just pick the robust FPs for that distortion type. E.g., referring to Fig. 6, if the only difference between the VO viewer's pose and owner's pose is the rotation angle, we can choose OF_3 of the matching RO for feature comparison. 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, based on the pose information of the viewer, we first assign a weight value (w_i) to each involved distortion type. 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 $\pm 45°$ 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. For example, if the VO owner's pose is $\{< 30°, 5°, 0° >, 1 \text{ meter}\}$, and the viewer's pose is $\{< 20°, 10°, 0° >, 1.2 \text{ meters}\}$, three distortion types are involved, and their weights are $\{\frac{|20-30|}{45}, \frac{|10-5|}{45}, \frac{|1.2-1|}{0.8}\} = \{0.22, 0.11, 0.25\}$.

Then we develop Algorithm 2 to select FPs so that the number of FPs chosen from each distortion type is proportional to its weight value. The inputs include the weights (w_i) and the robust FP lists from the package file (OF_i for distortion type D_i). 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 3–8). The distortion type with the maximum deficit is recorded (index number is idx), and the algorithm selects a FP from its FP list OF_{idx}. In lines 9–10, 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 11–15. 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.

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  |   |   end
8  |   end
9  |   nFP = OF_idx[p_idx++];
10 |   RET = RET ∪ nFP;
11 |   for OF_i do
12 |   |   if nFP ∈ OF_i then
13 |   |   |   c_i++,  remove nFP from OF_i;
14 |   |   end
15 |   end
16 end
17 return RET;
```

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.

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 confidences 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, let M be the set of matches selected after these two rounds of matching processes. The matching ratio for this RO is $\frac{|M|}{N}$.

4.3.3 RO-based AR scene

In the last step, if the VO viewer recognizes at least one ROs, i.e., the matching ratio is greater than the system threshold, 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 fol-

lowing Fig. 8 to briefly illustrate the basic steps.

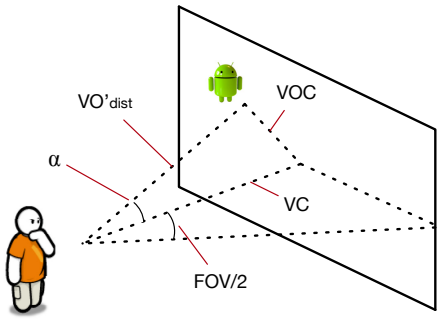

Figure 8: Restore the VO based on an RO

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. 8, we show a angle of $\frac{FOV}{2}$ referring to the horizontal FOV. Since we know the dimension of the frame, it is straightforward to calculate the pixel value of VC 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 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.

5. PERFORMANCE EVALUATION

In this section, we present our performance evaluation. We have implemented PROMAR in Java, and incorporated it with Android ARCore [3]. Our mobile application codes are based on the ARCore's sample application [1] with additional 7,200 lines of codes. Our experiments are conducted on a mid-range Moto G6 phone. In addition, 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. All the source codes can be obtained from an anonymous github link `https://github.com/PROMAR2019`.

5.1 Settings and Workload

Our solution mainly includes a new feature extraction algorithm and a new feature matching algorithm. We will first examine them individually, and then evaluate the overall performance combining both of them. In our experiments, we use the ORB implementation in OpenCV [10] as image feature detector and descriptor.

Several image datasets are used for evaluating the accuracy of our system.

ALOI [18]: 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).

3D objects on Turntable [30]: It includes the images of 100 objects taken from different horizontal angles with an interval of 5°. Each image includes multiple objects with background noise. It represents the cases with multiple objects and a single type of distortion.

BigBIRD [39]: 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 Google image using the corresponding object title as keyword. They form a dataset of false images for the target object. The false image set is only used in the cases with a single object.

The main performance metrics we evaluate is the recognition accuracy and runtime overhead. For accuracy, we consider the precision rate defined as $TP/(TP+FP)$, recall rate as $TP/(TP+FN)$, and true negative rate as $TN/(TN/FP)$, where TP, FP, TN and FN represents true positive, false positive, true negative and false negative respectively.

5.2 Evaluation of Feature Extraction

In this subsection, we evaluate our distortion-based feature extraction algorithm. The alternative we compare to is the original ORB feature extraction. For the feature matching, we use the same Brute-Force matching with cross-checking for both feature extraction algorithms.

We conduct the evaluation with 5 objects from ALOI image set. For each object, there are 72 images taken at different horizontal view angle with an interval of 5°. In our feature matching algorithm, we generate 10 distorted images for distortion tests, and extract 100 robust FPs for the horizontal angle distortion. For each test, we pick an image as the VO owner's image, and compare to the images taken from adjacent angles ($<$ 40°). We repeat this process for all the images of the object, and record the matching ratio for each case. The

same workflow is again applied to regular ORB feature extraction as the comparison.

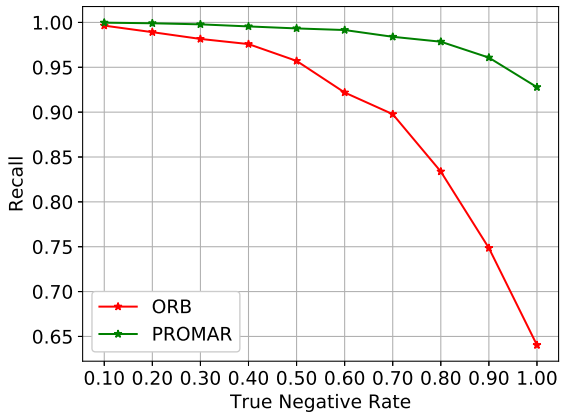

Figure 9: Comparison of feature extraction.

Fig. 9 illustrates the comparison of accuracy. In this test, we adjust the matching ratio threshold to yield a given true negative (TN) rate, i.e., axis x, and then examine the recall rate (axis y). Apparently, PROMAR is superior to ORB with a much higher recall rate under the same TN rate. Even when TN=1 (rejecting all false images), our feature extraction algorithm yields a recall rate of 0.93 with the built-in Brute-force matching algorithm, while ORB's recall is only 0.64. It shows that Algorithm 1 is effective finding a set of robust FPs.

5.3 Evaluation of Feature Matching

In this subsection, we individually evaluate the feature matching algorithm in PROMAR. We use the regular ORB algorithm to extract 100 FPs from each image. The alternative matching algorithm we compare to is a combination of OpenCV's 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.

Our evaluation is also based on ALOI image dataset. We select 10 objects, and use each of them images as the base image comparing to the adjacent images. We confine the degree change in $\pm 40°$. We also apply the matching algorithm with false images. In total, more than $83,000$ matching results are recorded and processed. The average matching ratios are shown in Fig. 10.

We observe that with the same set of FPs, our feature matching algorithm achieves considerably higher matching ratios. Especially when the angle change is large, our algorithm significantly improves the matching ratio. Foe example, when the angle change is 25°, RANSAC yields a matching ratio of 0.16, and our algorithm achieves 0.21, and improvement of 31%.

5.4 Overall Evaluation

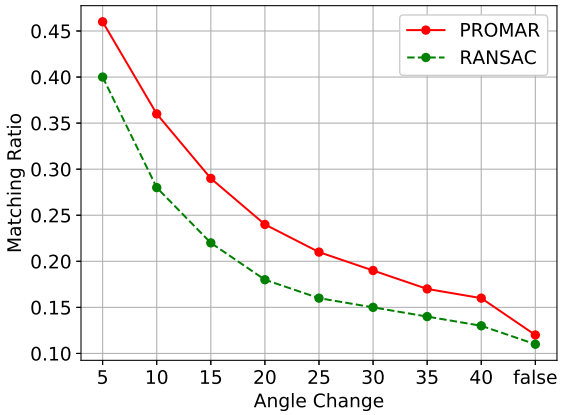

Figure 10: Comparison of feature matching methods.

In this subsection, we evaluate the overall system with both our feature extraction and matching algorithms.

(a) The left is query image while the right is template image.

(b) Matching between each instances.

Figure 11: Our system separates each recognized object out and then use our feature point detection and matching algorithm to decide if the scenery is matched.

Overall process. We first use Fig 11 to illustrate how our system works when comparing the *template image* (the image from the VO owner) and *query image* (the image captured y the VO viewer). We use TensorFlow [4] to recognize each RO in the template image and save the robust FPs. When the query image is captured, PROMAR processes it in a similar way, recognizing objects and further extracting regular FPs, e.g., 500 ORB feature points for each object. Then we apply our matching algorithm to compare those recognized objects in query image to the recorded ROs.

In the example given in Fig. 11, PROMAR firstly

recognizes the three ROs in template image as person(the cartoon character), fire extinguisher and person(the blue monster). Next, PROMAR crops those three instances out and extract robust feature points from each of them respectively (by the VO owner). When a query image is captured (by the VO viewer), PROMAR uses a similar way to crop the three objects and then extract a specific number of regular ORB FPs. Then, PROMAR compares the RO and the matching objects recognized in the query image. Fig. 11b shows the output matching after the whole workflow. The matchings are both persuasive and accurate.

Next, we present the results for different cases with varying numbers of objects and distortion types.

Single Object/Single Distortion. Due to the page limit, we only present the results of a single distortion on horizontal angle and scale. For horizontal angle distortion, we also test it with 10 objects in ALOI. Specifically, we extract 100 template robust FPs in corresponding to the view angle difference of query image to template image and 500 regular ORB FPs from query image, after which our matching is applied. The average matching ratios are listed in Table. 1. Similarly, we repeat the previous process on scaled images. The output is recorded in Table. 2.

We observe that within a reasonable range of the angle change, e.g., $\pm 25°$, 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 $\pm 40°$. The performance for scale distortion is even better. In all the tested cases, the matching ratios of the false images are quite low (< 0.05) while those of the true images keep above 0.28.

Table 1: Avg matching ratio for each categories on different 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

Single Object/Multiple Distortion. Here we show the evaluation results of two cases: 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 three groups of this dataset to test the performance of multiple (two) distortion types. We pick 4 objects in this test, and show the results in Fig. 12. The x value stands for the horizontal angle difference between template and query images, y value represents a matching ratio. Since the vertical view angle difference is identical (22°) in this experiment, we do not explicitly show it in the chart.

Table 2: Avg matching ratio for different scale change

Scale change	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	false
Lego man	0.47	0.46	0.46	0.43	0.40	0.36	0.32	0.28	0.01
Shoe	0.48	0.47	0.46	0.45	0.40	0.38	0.35	0.30	0.02
Furry elephant	0.45	0.46	0.45	0.44	0.39	0.38	0.33	0.31	0.02
Toy bear	0.46	0.48	0.47	0.45	0.43	0.40	0.39	0.34	0.02
Van gogh	0.47	0.46	0.46	0.42	0.39	0.38	0.34	0.31	0.01
Furry bear	0.44	0.42	0.43	0.43	0.40	0.37	0.33	0.30	0.02
Duck cup	0.46	0.46	0.47	0.45	0.43	0.42	0.37	0.33	0.05
Furry dog	0.45	0.45	0.45	0.42	0.38	0.36	0.31	0.28	0.01
Baby cream	0.42	0.44	0.45	0.40	0.40	0.37	0.33	0.29	0.03
Girl statue	0.44	0.44	0.43	0.41	0.39	0.37	0.34	0.32	0.04
Average	0.45	0.45	0.45	0.43	0.40	0.37	0.34	0.30	0.02

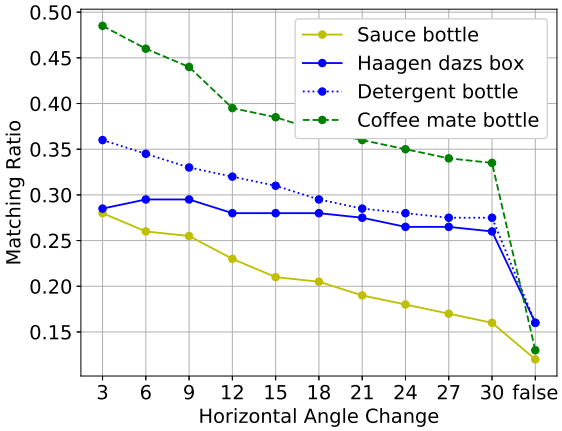

Figure 12: Average matching ratio with two types of distortions (vertical angle change is fixed at 22°)

The results show that PROMAR is quite effective even when both horizontal and vertical angles are changed. The matching ratio of false images is obviously lower than the true images in Fig. 12. Sauce bottle is actually an exceptional case because it does not have sufficient number of FPs. It is sensible to believe the gap between false and true images in practice can be larger.

Scale + Horizontal angle change. In this test, 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 template image and compare it with images on all scales whose horizontal angle differences are less than 40°. The results are illustrated in Fig. 13, where horizontal angle change lies on the x-axis, scale change on y-axis, and z values as well as the heat map indicate the matching ratios. For the simplicity of the plot, we do not include matching ratio of false

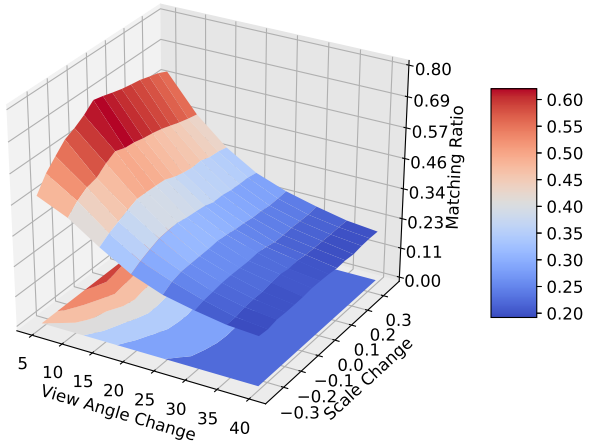

Figure 13: The matching ratio with two types of distortions (scale and horizontal angle).

images, of which the average is 0.17.

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.

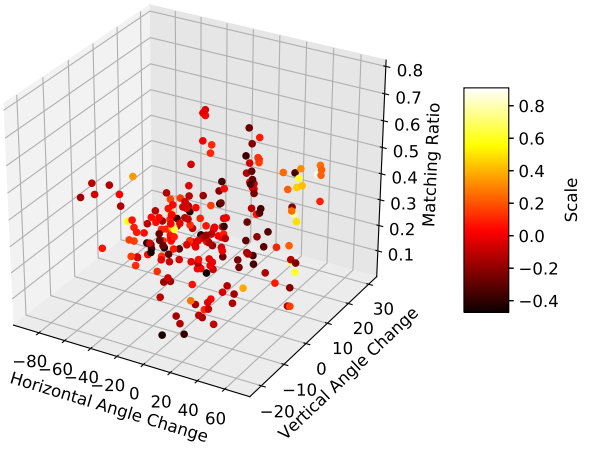

Figure 14: The real-time matching ratio on mobile.

Mobile Phone Experiments: We implement PROMAR on Android, and conduct experiments to test the performance in realistic scenarios. We develop an additional logger that records the real-time matching ratios as well as the distortion types involved for each frame processed by PROMAR. Specifically, we place virtual objects on laptop, chair, and bed. Then we play the role of a VO viewer and move the camera around for 10 minutes. The data recorded by the logger is illustrated in Fig. 14. The axis z shows the matching ratio, while axis x indicates the horizontal angle change, axis y indicates the vertical angle change, and the color represents

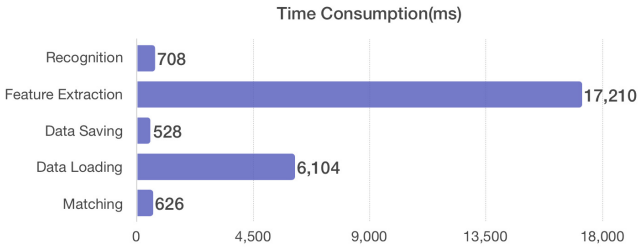

Figure 15: The average time consumption of each processing.

the scale change (illustrated on heat map).

The average of matching ratio in our test is 0.28 which is consistent with our previous test with image data sets. We can observe that most of the points reside around that level. These results confirm that PROMAR can achieve remarkable performance in real world tests, even when the distortion is significant, e.g., shrinking size to about half or changing view angle for more than 45°.

In addition, we illustrate the computation overhead measured on a Moto G6 Android phone in Fig 15. Our first step is to recognize objects in each frame, which takes 708ms to finish it on average. During VO placement we would need to save distortion-based robust feature points. This process is rather time-consuming which takes up to 17210ms. However, feature extraction is a one-time operation and can be done in background, since all the necessary data required to retrieve the scenery can be obtained at the instant when user clicks save button. Furthermore, we observe the data saving operation can cost averagely 528ms on our experiment phone, of which the length is a trifle here. As for VO viewer, they need to load those data stored in previous process before they do matching. The data loading process costs about 6104ms and our matching algorithm only takes 626ms to complete one comparison, which means it can be done in real time without the user's perception of delay.

In conclusion, our evaluation validates that PROMAR is a practical and effective multi-user AR framework. The feature extraction, and matching algorithms outperform state-of-the-art counterpart algorithms.

6. CONCLUSION

In this paper, we develop a framework that support multi-user AR applications. We present novel algorithms for image feature extraction and feature matching process. We basically combine the image feature properties with the mobile phone's pose information throughout the system design. This unique property leads to significant performance improvement. We have implemented the entire system, and evaluated with a prototype and experiments. The results show that our system is effective and practical.

7. REFERENCES

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

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

[3] Google ARCore. https://developers.google.com/ar/.

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

[5] Vuforia. https://www.vuforia.com/.

[6] Wikitude. https://www.wikitude.com/.

[7] YOLO. https://pjreddie.com/darknet/yolo/.

[8] Alcantarilla, P. F., Bartoli, A., and Davison, A. J. Kaze features. In *European Conference on Computer Vision* (2012), Springer, pp. 214–227.

[9] Arth, C., Klopschitz, M., Reitmayr, G., and Schmalstieg, D. Real-time self-localization from panoramic images on mobile devices. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on* (2011), IEEE, pp. 37–46.

[10] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

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

[12] Concha, A., Loianno, G., Kumar, V., and Civera, J. Visual-inertial direct slam. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on* (2016), IEEE, pp. 1331–1338.

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

[14] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 6 (2007), 1052–1067.

[15] Dou, M., Fuchs, H., and Frahm, J.-M. Scanning and tracking dynamic objects with commodity depth cameras. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on* (2013), IEEE, pp. 99–106.

[16] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence 32*, 9 (2010), 1627–1645.

[17] Gauglitz, S., Sweeney, C., Ventura, J., Turk, M., and Hollerer, T. Live tracking and mapping from both general and rotation-only camera motion. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on* (2012), IEEE, pp. 13–22.

[18] Geusebroek, J.-M., Burghouts, G. J., and Smeulders, A. W. The amsterdam library of object images. *International Journal of Computer Vision 61*, 1 (2005), 103–112.

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

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

[21] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR abs/1512.03385* (2015).

[22] Klein, G., and Murray, D. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on* (2009), IEEE, pp. 83–86.

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

[24] Leibe, B., Leonardis, A., and Schiele, B. Combined object categorization and segmentation with an implicit shape model. In *Workshop on statistical learning in computer vision, ECCV* (2004), vol. 2, p. 7.

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

[26] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research 34*, 3 (2015), 314–334.

[27] Liu, H., Zhang, G., and Bao, H. Robust keyframe-based monocular slam for augmented reality. In *Mixed and Augmented Reality (ISMAR), 2016 IEEE International Symposium on* (2016), IEEE, pp. 1–10.

[28] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110.

[29] McIlroy, P., Izadi, S., and Fitzgibbon, A. Kinectrack: 3d pose estimation using a projected dense dot pattern. *IEEE transactions on visualization and computer graphics 20*, 6 (2014), 839–851.

[30] Moreels, P., and Perona, P. Evaluation of features detectors and descriptors based on 3d objects. *IJCN* (2007).

[31] Mur-Artal, R., and Tardós, J. D. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters 2*, 2 (2017), 796–803.

[32] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S., and Fitzgibbon, A. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on* (2011), IEEE, pp. 127–136.

[33] Oyallon, E., and Rabin, J. An analysis of the surf method. *Image Processing On Line 5* (2015), 176–218.

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

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

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

[37] Salas-Moreno, R. F., Glocken, B., Kelly, P. H., and Davison, A. J. Dense planar slam. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on* (2014), IEEE, pp. 157–164.

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

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

[40] Smith, R. C., and Cheeseman, P. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research 5*, 4 (1986), 56–68.

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

[42] Szegedy, C., Ioffe, S., and Vanhoucke, V. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR abs/1602.07261* (2016).

[43] Tan, W., Liu, H., Dong, Z., Zhang, G., and Bao, H. Robust monocular slam in dynamic environments. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on* (2013), IEEE, pp. 209–218.

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