

Multi-User Augmented Reality: Achieving Light Weight Synchronization With Embedded Sensors

Zida Liu , *Student Member, IEEE*, Ruiqi Wang , and Gonghong Cao , *Fellow, IEEE*

Abstract—Multi-user Augmented Reality (AR) holds great promise for facilitating collaboration among individuals by allowing them to perceive and interact with virtual objects. To achieve multi-user AR, it is important to keep different users' coordinates well-synchronized so that one user can share the created virtual object with others. However, most existing synchronization methods rely on resource-intensive object detection algorithms or mobile SLAM, which leads to high energy consumption and is not suitable for mobile devices. In this paper, we propose to achieve lightweight synchronization for multi-user AR by leveraging multiple embedded sensors on mobile devices such as IMU sensor, ultra-wideband (UWB) sensor, and camera sensors. Specifically, we propose a novel AR localization algorithm for tracking the relative positions of AR users. It utilizes particle filters to fuse the IMU and UWB sensing data, and employs an adaptive resampling method to improve the localization accuracy. To further enhance the positional accuracy of virtual objects displayed in AR, we design a feature-based position modification algorithm. Extensive evaluations demonstrate that our synchronization method significantly improves accuracy and reduces energy consumption compared to existing baselines.

Index Terms—Multiple users, augmented reality, sensor fusion, energy saving.

I. INTRODUCTION

MOBILE augmented reality (AR) allows for the integration of digital content or virtual objects into the real-world environment, providing an immersive experience on mobile devices. Multi-user AR [1], [2], [3], [4] takes this a step further by enabling multiple users to perceive and interact with virtual objects or digital content within the same physical space. Consider a scenario where FBI agents equipped with AR navigate a building to rescue hostages. In this situation, the lead agent's AR device automatically detects and highlights the hostages and terrorists on its display. Simultaneously, other agents also see these virtual overlays on their own displays. If any of the individuals involved move, the virtual overlays must be updated on the appropriate AR displays. Similar examples exist for an ad-hoc AR game with a group of users, or a group of people exploring a museum using AR for a guided tour.

Received 11 June 2025; revised 6 April 2026; accepted 17 April 2026. Date of publication 22 April 2026; date of current version 6 August 2026. This work was supported by the National Science Foundation under Grant 2125208 and Grant 2215043. Recommended for acceptance by J. Zhang. (*Corresponding author: Zida Liu.*)

The authors are with the School of Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA 16802 USA (e-mail: zjl5310@psu.edu; ruw400@psu.edu; gxc27@psu.edu).

Digital Object Identifier 10.1109/TMC.2026.3686765

TABLE I
ENERGY EXPENDITURES OF DIFFERENT TECHNIQUES

Operation	Power (W)
Mobile-SLAM (ARCore) [9]	2.54
DNN [14]	4.13
IMU-based tracking [10]	0.34
UWB ranging [15]	0.103
Camera-based Modification (Sec. 4)	0.11

To enable multi-user AR applications, AR devices need to determine where to place virtual overlays based on real-world scene analysis, typically by utilizing resource-intensive object detection algorithms based on Deep Neural Networks (DNNs) [4], [5], [6], [7], [8]. However, since AR devices are detecting and tracking the same set of physical objects, it is unnecessary for all devices to perform object detection individually. Instead, a primary user is selected to run the DNN, and the other secondary users can re-project the virtual objects based on their relative positions and orientations to the primary user.

This approach helps save energy since most users can avoid running energy-intensive DNNs themselves. To achieve this, secondary users must know how to transform the primary user's coordinates to their own coordinate system, ensuring accurate placement of virtual overlays. This process is known as *coordinate synchronization*. Conventional approaches [2], [9], [10], [11] involve synchronizing users' coordinates at the start of the AR session and utilizing mobile SLAM (Simultaneous Localization and Mapping) [12], [13] to track their 6 degrees of freedom (6DoFs) movement. However, mobile SLAM consume high amounts of energy [10], negating the energy-saving advantage of multi-user AR.

To overcome this problem, we propose to achieve lightweight synchronization in multi-user augmented reality by leveraging multiple embedded sensors on mobile devices such as IMU sensor, ultra-wideband (UWB) sensor, and camera sensors. These sensors have much less power consumption compared to running DNNs or mobile-SLAM. Table I shows the power consumption of these sensors based on our measurements using a Google pixel 6 Pro smartphone. We used Google ARCore [9] as the SLAM implementation, and Google MediaPipe [14] as the DNN. We empirically evaluated the power consumption of each component using Android BatteryManager API (more details are in Section V-A). As shown in the table, the energy cost of DNN alone is 4.1 W and mobile-SLAM alone is 2.54 W. However, the energy consumption for using embedded sensors is only $0.34 + 0.103 + 0.11 = 0.56$ W.

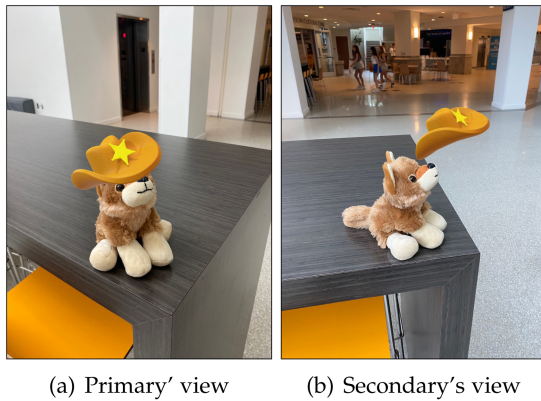


Fig. 1. The UWB error leads to an AR display error in the secondary's view.

Although individual sensors have been utilized for similar purposes, they all have various problems. For example, existing approaches [10], [16], [17], [18] have utilized IMU sensors. However, one major disadvantage of IMU is the accumulated error over time when applied to localization, and it will lead to inaccurate position that eventually results in a huge drift. Another approach is to leverage the UWB sensors. UWB-enabled devices can continuously localize each other's 3D position with reduced power consumption. However, UWB localization accuracy can be affected by interference from the user's body when holding the mobile device [19]. This interference can lead to severe AR display errors, resulting in an unsatisfactory AR experience. For example, as shown in Fig. 1, we conducted some experiments based on two Google Pixel 6 pro smartphones, where the primary user placed a virtual hat on the fox's head through ARCore. After re-projecting the virtual hat from the primary user by UWB ranging, the secondary user saw a virtual hat on the fox's head from a different view. Since UWB introduced some location error, the virtual hat was not placed in the right position.

In this paper, rather than relying solely on individual sensor, we combine multiple sensors to achieve light weight synchronization. We first obtain the original/raw relative 3D position by UWB ranging, and then calibrate it by fusing users' movement calculated by the IMU sensors. We propose a novel IMU/UWB fused AR localization algorithm, which takes advantage of the complementary nature of IMU and UWB to achieve light weight synchronization. We further improve the position accuracy of the virtual object on the display by utilizing the camera sensor, i.e., leveraging image based technique. We propose a feature-based position modification algorithm, which further reduces the rendering discrepancy when overlaying the virtual object on the appropriate AR displays. Different from feature-based methods (relying only on camera sensor) for tracking users' pose movement [10], [12], our method does not involve complex pose estimation calculations and only operates on a small part of the original image, thus the cost is much lower.

The main contributions of the paper are as follows:

- We leverage multiple embedded sensors to synchronize AR users with low energy and better accuracy.
- We propose a novel AR localization algorithm for tracking the relative positions of the AR users. It utilizes particle

filters to fuse the IMU and UWB sensing data, and employs an adaptive resampling method to improve the localization accuracy.

- A feature-based position modification algorithm is developed to further improve the positional accuracy of virtual objects displayed in AR.
- We have built a prototype to evaluate the proposed techniques on smartphones. The evaluation results demonstrate that our method significantly improves accuracy and reduces energy consumption compared to existing baselines.

II. SYSTEM OVERVIEW

To save energy in multi-user AR, rather than each user individually executing resource-intensive object detection algorithms to determine the placement of virtual objects, a primary user¹ is selected to run DNN, while other secondary users can synchronize with the primary and then re-project the virtual object to the right position. Fig. 2 illustrates our system architecture.

Primary User: When an AR session starts, the primary user runs object detection algorithms on his mobile device to determine the specific virtual object's position and orientation in his coordinate system. Subsequently, the rendering pipeline overlays the virtual object onto the display. By leveraging the light-weighted MUSE algorithm [20], we can accurately determine the primary user's 3D orientation based on the IMU sensing data, even when the user is in motion. Following a similar approach as in [10], we estimate the primary's motion during a short-time interval by performing double integration on the linear acceleration. Finally, we extract ORB visual features within the bounding box given by the object detection algorithm, and such information is used to further improve the location accuracy.

The primary transmits synchronization data to the secondaries periodically, based on which the secondaries can track the primary's six degrees of freedom (6DoFs) and re-project the virtual object from the primary's coordinates to their own. Specifically, the synchronization data has the following information: 1) the primary's 3D orientation and motion during the current time slot in the global reference frame (GRF),² calculated using its local IMU sensors. 2) the virtual object's information, including its identification, orientation, and position relative to the primary user. 3) visual features extracted from the image captured by the camera sensor, which is used for improving the accuracy of the virtual overlay's position on the AR device's display.

Secondary Users: Before re-projecting the virtual object from the primary's coordinate system to his own coordinate system, the secondary user needs to track the primary's 3D orientation and position relative to him. For the relative 3D orientation, the secondary can compare the primary's orientation in GRF with his own orientation in GRF and then calculates the difference.

¹The primary role can be dynamically rotated among users to distribute energy consumption and achieve better detection accuracy.

²In GRF, the Y-axis is tangential to the ground at the device's current location and points towards the magnetic north pole. The Z-axis points towards the sky and is perpendicular to the ground. The X-axis is defined as the cross product of Y and Z.

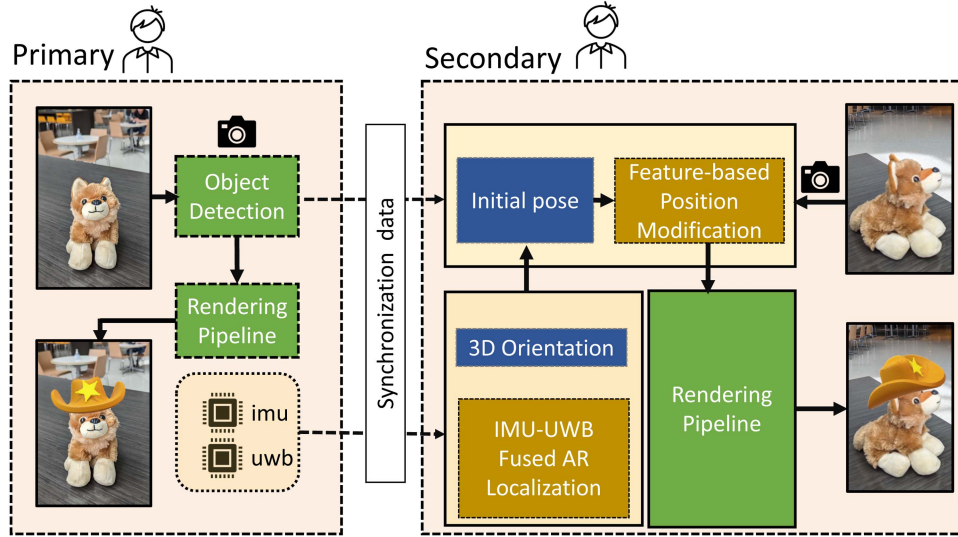


Fig. 2. System overview.

For 3D position, although UWB ranging can provide such measurement, it is inaccurate and cannot be used directly. Thus, we propose an IMU/UWB fused AR localization algorithm to improve its accuracy by combining IMU and UWB techniques and leveraging the complementary characteristics of IMU and UWB. Specifically, the position movement calculated by IMU dead-reckoning exhibits high accuracy within a short time duration, but suffers from significant drift over time. Conversely, the positional information obtained from UWB ranging is subject to noise but the noise is independent and does not accumulate over time. The details of the algorithm will be presented in Section III.

After obtaining the primary's relative position, orientation and the virtual object's information, the secondary determines where to overlay the virtual object by transforming its position from the primary's coordinate system to his own. That is,

$$[x_s, y_s, z_s]^T = [R|p][x_p, y_p, z_p]^T \quad (1)$$

Here, $\{x_p, y_p, z_p\}$ denotes the 3D position of the virtual object in the primary's coordinate, R and p denote the primary's 3D orientation and position relative to the secondary.

To further improve the position accuracy, we leverage the camera sensor and propose a feature-based position modification algorithm, which further reduces the rendering discrepancy when overlaying the virtual object on the appropriate AR displays. The details of the algorithm will be presented in Section IV.

III. IMU/UWB FUSED AR LOCALIZATION

The IMU-UWB fused AR localization is designed to track the primary's 3D position within the secondary's local reference frame. Although previous approaches [21], [22] have combined IMU and UWB technologies for indoor localization, their focus has been on determining the user's absolute position. However, in a multi-user AR scenario, the secondary does not need to know the absolute positions of the primary and himself. Instead, our main focus lies in the secondary's capability to obtain the

primary's relative position to himself, which we refer to as *AR localization*. With this information, the secondary can re-project the virtual object from the primary's coordinate system into his coordinate system.

A. Problem Formulation of AR Localization

In AR localization, our goal is to enable the secondary to track the primary's position relative to himself. By leveraging UWB ranging, the secondary can obtain the primary's initial position in his own reference frame. However, this measurement is inaccurate and noisy, which is represented as $\tilde{p}^t = p^t + \sigma^t$, where $\tilde{p}^t$ denotes the measured position of the primary at time t , p^t denotes the ground truth position and σ^t represents the position error. The primary and secondaries can utilize their IMU sensors to track their movements over short time intervals. The primary's movement sequence is denoted as $M_p = m_p^0, m_p^1, \dots, m_p^t$ and the secondary's movement sequence is denoted as $M_s = m_s^0, m_s^1, \dots, m_s^t$. Our goal is to utilize these movement sequences, along with the UWB ranging sequence $\tilde{P} = \tilde{p}^0, \tilde{p}^1, \dots, \tilde{p}^t$, to estimate p^t .

To formulate the problem, we consider the joint posterior probability $\mathbb{P}(p^t | \tilde{P}, M_p, M_s)$. Since the UWB ranging and IMU tracking data originate from different hardware units, $\tilde{P}$ is independent of M_p and M_s . Drawing on Bayesian theory and the Markov assumption, we have the following:

$$\begin{aligned} & \mathbb{P}(p^t | \tilde{P}, M_p, M_s) \\ & \propto \underbrace{\mathbb{P}(\tilde{p}^t | p^t)}_{\text{UWB Ranging}} \int \underbrace{\mathbb{P}(p^t | M_p, M_s, p^{t-1})}_{\text{IMU dead reckoning}} d\mathbb{P}(p^{t-1}) \end{aligned}$$

Although various algorithms [21], [23] exist for maximizing the joint posterior probability $\mathbb{P}(p^t | \tilde{P}, M_p, M_s)$, such as the Kalman filter, unscented Kalman filter, and extended Kalman filter, they all assume that the observation noise follows a zero-mean Gaussian distribution. However, in NLOS (non-line-of-sight) scenarios, the UWB position error does not adhere

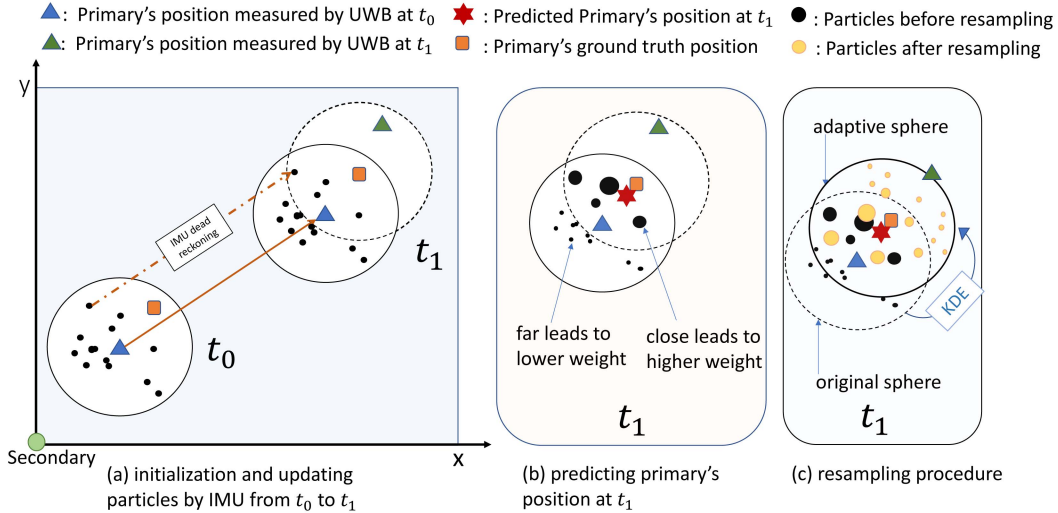


Fig. 3. IMU/UWB fused AR localization.

to a Gaussian distribution [19]. Unlike Kalman filter, particle filters can handle nonlinear and non-Gaussian systems through a sampling-based approach. Thus, we employ a particle filter to fuse measurements from the IMU and UWB sensors. A basic particle filter consists of the following steps:

- **Initialization:** spreading a set of particles where each particle represents a hypothesis of the system.
- **Updating particles:** updating the particles in the next state of the system based on the system transformation function.
- **Importance weighting:** each particle is assigned a weight based on how well it fits the current measurements.
- **Prediction:** making a prediction by calculating the weighted mean of all particles.
- **Resampling:** creating a new set of particles by choosing particles from the current set with probabilities proportional to their weights.

B. IMU/UWB Fused AR Localization

Although previous methods [22], [24], [25], [26] have employed particle filters to integrate IMU and UWB for indoor localization, their focus lies in localizing the user's absolute 3D position within the indoor environment using distance measurements from the mobile device to pre-installed UWB anchors. Consequently, during the initialization stage, these methods disperse particles across the entire area where the user can move, requiring a large number of particles for efficient position tracking. However, this approach results in high computational overhead and slower convergence of the localization process. In contrast, in AR localization, the goal is to find the primary's 3D position relative to the secondary. To address this specific relative position estimation problem in AR, we propose a new algorithm.

In our algorithm, as illustrated in Fig. 3, rather than spreading particles over the whole space, we only spread the particles near the measured position by UWB ranging. This selective approach enables the particle filter to disregard irrelevant space where the ground truth is unlikely to appear, focusing on the region where

the ground truth position is expected to be found. The initialization stage, as shown in Fig. 3(a), involves obtaining the primary's 3D position in the secondary's coordinates using UWB ranging. A sphere is then constructed with the current position as its center, and a large radius is set to ensure the inclusion of the ground truth position within the sphere, and particles with equal weights are uniformly sampled inside. From t_0 to t_1 , all particles are updated based on the primary's movement calculated by IMU dead reckoning. At time t_1 , the weight of each particle is updated according to its Euclidean distance to the new measured position, and the predicted position is obtained by calculating the weighted average of all updated particles, as shown in Fig. 3(b).

Although the initialization of particles within the proximity area of the measured position can reduce the hypothesis space. As time goes, the hypothesis space inevitably diverges from the ground truth position due to the substantial drift of IMU dead reckoning. As a result, the ground truth position may eventually fall outside the hypothesis space. Although the drifted ground truth position may be covered in prior indoor localization work [24], [25], they distribute particles across the entire space (the large light blue square in Fig. 3(a)), introducing significant computational overhead, which may fail to meet the real-time requirement in our multi-user AR scenario.

To overcome this limitation, instead of relying solely on the original particle set, in each state, we resample a set of particles with equal weights to form a new sphere (see Fig. 3(c)). The sphere's center is the predicted position and its radius is the euclidean distance between the predicted position and the measured UWB position. The new sphere's center is generated by incorporating information related to current UWB measurements, effectively eliminating the drifting problem associated with the original sphere. Additionally, the radius of the new sphere can be adaptively adjusted to account for estimated UWB errors, further improving the accuracy of the resampling process.

After determining the new sphere for resampling, how to assign weights to the particles becomes crucial. Assigning equal weights to all particles would eliminate the posterior distribution

Algorithm 1: IMU/UWB Fused AR localization.

Input: P', M_h, m_s, R_s
Parameters: number of particles n , initial sphere radius r
Output: predicted primary's 3D position p^t ;
Init: $p^0 \leftarrow \tilde{p}^0$; $Z \leftarrow \text{sample}(n, \tilde{p}^0, r)$; $W \leftarrow \frac{1}{n}$;

```

1 for each time  $t = 1, 2, 3, \dots$  do
2   for each particle  $i = 1, 2, 3, \dots, n$  do
3      $z_i = r_s^{t-1} \times z_i$ ;
4      $z'_i = (r_s^t)^{-1} \times (z_i + m_p^t - m_s^t)$ ;
5   end
6    $W \leftarrow \text{SIS}(\tilde{p}^t, Z', W)$ ;
7    $p^t \leftarrow \text{WeightedAverage}(W, Z')$ ;
8    $f \leftarrow \text{KDE}(Z', W)$ ;
9    $Z \leftarrow \text{sample}(n, p^t, \text{dist}(p^t, \tilde{p}^t))$ 
10  for each particle  $i = 1, 2, 3, \dots, n$  do
11     $w_i = \frac{f(z_i)}{\sum f(z_i)}$ 
12  end
13 end

```

of the ground truth within the original sphere. To preserve this distribution, we utilize Kernel Density Estimation (KDE). This estimation ensures that the data distribution implied by the particle set in the new sphere approximates the one implied in the original sphere. Specifically, we first estimate the data distribution from the weighted particles in the original sphere using KDE, then assign weights to particles in the new sphere based on how well they match the estimated distribution. Mathematically, KDE is adopted as follows:

$$f(z_i) = \frac{1}{n} \sum_{i=1}^n \frac{1}{h} \cdot w_i \cdot K\left(\frac{z_i - z'_i}{h}\right) \quad (2)$$

where n denotes the total number of particles in the original sphere, and h is a parameter to determine the width of the kernel function. The variables z'_i and w_i correspond to the position and the weight of a particle in the original sphere, respectively. The function $K(\cdot)$ represents the Gaussian kernel used in the three-dimensional space. By utilizing the particles from the original sphere along with their respective weights, we can calculate the weight $f(z_i)$ for a particle in the new sphere whose position is z_i .

Algorithm 1 shows the details of our algorithm for calculating the primary's relative 3D position, where $\text{sample}(n, p^t, r)$ represents that we uniformly sample n particles from a sphere with center p^t and radius r . $Z = \{z_1, z_2, \dots, z_n\}$ denotes the positions of the n sampled particles after $\text{sample}(n, p^t, r)$. $W = \{w_1, w_2, \dots, w_n\}$ denotes the weights of the sampled particles. The algorithm initializes with $\text{sample}(n, \tilde{p}^0, r)$, and all samples have equal weight.

In lines 2-5, we calculate the positions of the particles in the secondary's coordinate in a new time slot. First, we transform the particles' positions in GRF. Here, $R_s = \{r_s^0, r_s^1, \dots, r_s^t\}$ denotes the rotation matrix transforming the secondary's current orientation to GRF. $Z' = \{z'_1, z'_2, \dots, z'_n\}$ denotes the positions of n particles after being updated by IMU-dead reckoning. With the rotation matrix, we can transform the particles to the GRF,

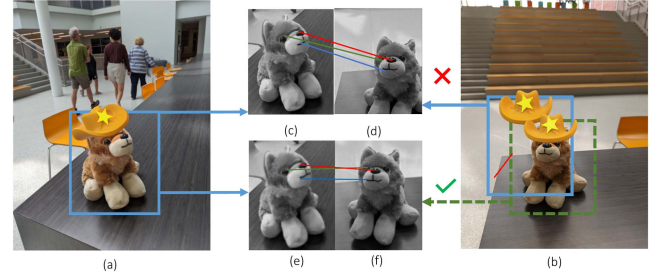


Fig. 4. Feature-based position modification: (a) Shows the image in the primary user's view, (b) shows the image in the secondary user's view. Due to the inevitable 3D position error between the primary and secondary, the target object may appear in the wrong area in the secondary's view (i.e., blue bounding box in (b)), and thus the virtual overlay (hat) is shifted. The problem can be reflected in (c) and (d); the corresponding points in the real world (one from the primary's view and the other one from the secondary's view) will have different pixel coordinates in their respective bounding box. Our feature-based modification will adjust the bounding box to the right position (green one in (b)), and then the pixel coordinates of the corresponding points will not differ too much, as shown in (e) and (f).

update the particles' position through IMU-dead reckoning from z_i to z'_i (line 4), and then transform them back to the secondary's coordinate (line 5).

In lines 7-8, we predict the primary's current position. We first assign different weights to the updated particles. Here, SIS denotes the sequential importance sampling [27], which assigns weights to updated particles based on their distances to the current measure position $\tilde{p}^t$. Subsequently, we calculate the predicted position by taking a weighted average of the positions of all particles.

Lines 9-13 represent the resampling process for predicting the primary's position in the next time slot. We first estimate the posterior distribution of the original particles by the KDE method (see (2)). Then, we generate a set of new particles with $\text{sample}(n, p^t, \text{dist}(p^t, \tilde{p}^t))$, where $\text{dist}(p^t, \tilde{p}^t)$ denotes the Euclidean distance between p^t and $\tilde{p}^t$. Finally, we assign weights to the new particles based on their proximity to the posterior distribution.

IV. POSITION MODIFICATION BASED ON CAMERA SENSORS

Although our IMU/UWB fused AR localization algorithm helps mitigate the 3D position error, some deviation remains. This deviation can still result in the virtual object not appearing in the correct position on AR device's display. To further decrease this display error, we propose a feature-based position modification from an image perspective.

Fig. 4 illustrates the basic idea of our feature-based position modification. Fig. 4(a) shows the image in the primary user's view and Fig. 4(b) shows the image in the secondary user's view. The blue bounding box in Fig. 4(a) illustrates the area where the fox toy is detected by the object detection algorithm. Due to the inevitable 3D position error between the primary and secondary, the target object may appear in the wrong area in the secondary's view (i.e., blue bounding box in 4(b)), and then the virtual overlay (hat) is shifted.

Our goal is to correct the position of the virtual overlay based on the position of the bounding box. The bounding box in the

Algorithm 2: Feature-Based Position Modification.

Input: $F_p, F_s, (u, v)$
Output: $\{u', v'\}$

```

1  $\{f_{p,i}, f_{s,i}\}_{i=1}^k \leftarrow \text{BruteForce}(F_p, F_s)$ 
2 for each feature point pair  $i = 1, 2, \dots, k$  do
3    $(x_{p,i}, y_{p,i}) \leftarrow \text{GetCoordinate}(f_{p,i})$ 
4    $(x_{s,i}, y_{s,i}) \leftarrow \text{GetCoordinate}(f_{s,i})$ 
5    $\Delta x_i = x_{p,i} - x_{s,i}, \Delta y_i = y_{p,i} - y_{s,i}$ 
6 end
7  $\bar{\Delta x} = \frac{\sum_{i=1}^k \Delta x_i}{k}, \bar{\Delta y} = \frac{\sum_{i=1}^k \Delta y_i}{k}$ 
8  $(u', v') = (u + \bar{\Delta x}, v + \bar{\Delta y})$ 

```

primary coordinate is generated after running the object detection algorithm. Based on (1), we can re-project the bounding box into the secondary's coordinate. For a pair of the corresponding points in the real world (one from the primary's view and the other one from the secondary's view), their pixel coordinates in their respective bounding box should not differ too much. As shown in Fig. 4(e) and (f), the fox toy's nose, right eye, and mouse appear in the upper center of the primary's bounding box, and they also appear in similar positions in the secondary's bounding box. However, when the virtual object deviates from the ground truth, as shown in Fig. 4(c) and (d), the fox toy's nose, right eye, and mouse appear in the upper center of the primary's bounding box, but they appear in the right center of the secondary's bounding box. This problem can be fixed by moving the secondary's bounding box towards the bottom right direction in the secondary's view until the fox's nose, right eye, and mouse also appear in the upper center of the bounding box. The virtual object should follow the same movement as the bounding box, and then correct its position error.

To identify the correspondence points, we first extract ORB features [28] in the primary and secondary's bounding boxes. Then, a brute-force approach is used to select the feature points with high matching degrees. Once the feature point pairs are obtained, their pixel coordinates within their respective bounding boxes are determined. By calculating the average pixel coordinate differences between the matched points, we can calculate the horizontal and vertical offset of the virtual object on the screen. Finally, we adjust the virtual object's position according to the offset.

Although our algorithm has some similarity to visual SLAM [12], [13] that they all extract and match the feature points from the images, they have key differences. Our algorithm is much more light weight, since it only extracts the ORB features within the bounding box, which represents a small proportion of the overall image. This is different from visual SLAM which extracts ORB features from the entire image. Moreover, we only use the feature points to adjust the position of the virtual objects, rather than constructing the 3D map of the environment in SLAM. Thus, our algorithm does not involve computational intensive modules for constructing 3D map of the environment such as backend optimization.

Algorithm 2 shows the details of our feature-based position modification. The algorithm has the following inputs: 1) two sets

of ORB feature points F_p and F_s extracted from the primary's and secondary's bounding box, each containing 20 ORB feature points. 2) the virtual object's position coordinate (u, v) on the secondary's display. In line 1, we use the brute-force method to find k pairs of matched feature points. For each pair, $f_{p,i}$ represents a feature point from the primary user, and $f_{s,i}$ represents a feature point from the secondary. In lines 2-6, we extract each feature point's pixel coordinates and calculate the coordinate difference within each paired feature point, where $(x_{p,i}, y_{p,i})$ and $(x_{s,i}, y_{s,i})$ denote the pixel coordinates of $f_{p,i}$ and $f_{s,i}$ within their respective bounding boxes. In lines 7-8, we average the coordinate difference among all matched feature points to obtain the virtual object's horizontal offset $\bar{\Delta x}$ and vertical offset $\bar{\Delta y}$. Then, we calculate the modified virtual object's position (u', v') .

Since the FPM algorithm relies on feature correspondences, it is applied only when sufficient matched feature points are available. When objects are partially occluded, the number of matched feature points may decrease, which can affect the accuracy of the FPM algorithm. We evaluate this impact in Section V. In cases of full occlusion, the secondary user cannot observe the object, resulting in no matched feature points (i.e., $k = 0$). In such situations, the system automatically bypasses the FPM algorithm and renders the virtual object directly based on the IMU/UWB fused AR localization results.

V. PERFORMANCE EVALUATIONS

We have implemented our multi-user AR system on six smartphones connected via WiFi, including Google Pixel 6 Pro, Google Pixel 10, and Samsung Galaxy S25+. The system implementation is built upon Google ARCore and Google MediaPipe [14]. For the primary user, we have integrated MediaPipe's Objectron as the object detection algorithm, while ARCore's rendering pipeline is employed to overlay the 3D virtual object on the screen.

Upon system initialization, we leverage the Android UWB API [15] to enable UWB ranging between the primary and secondary users. The primary user sends synchronization data to the secondary users via TCP sockets 30 times per second. For secondary users, we have implemented our AR localization algorithm and feature-based position modification algorithm using Android Studio and Android OpenCV SDK. This enables the secondary users to render virtual objects on their displays using ARCore's rendering pipeline.

Based on our implementation and testbed, we conduct extensive experiments to evaluate the performance of our system under a large number of users. To further demonstrate system scalability, we also perform simulations. We first evaluate the energy consumption of different approach in Section V-A, and measure the latency for synchronizing the primary and secondary users in Section V-B. We then evaluate the performance of IMU/UWB fused AR localization algorithm and Feature-based position modification algorithm. For the IMU/UWB fused AR localization algorithm, we simulate users' movements using a well-developed inertial navigation simulator [29]. This allows us to evaluate the scalability of our localization algorithm in

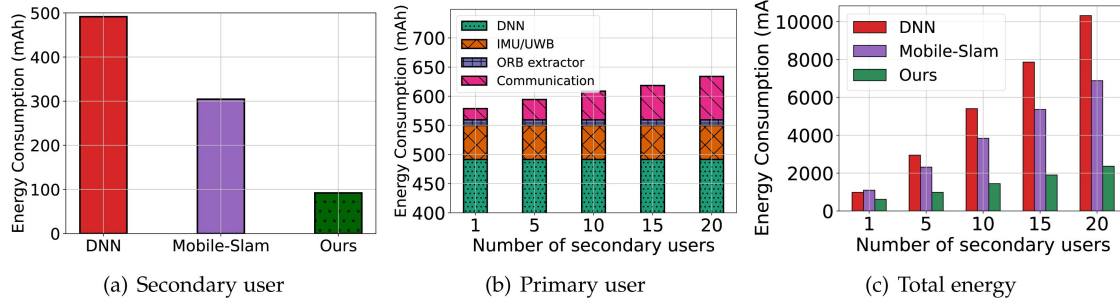


Fig. 5. Energy consumption comparisons. For a single secondary user, our method can reduce the energy consumption by 81% compared to the DNN-based object detection method and by 69% compared to using mobile-SLAM. As more AR users are involved, our method results in much less energy consumption increase than DNN and mobile-SLAM.

multi-user cases, and acquire ground truth data for the users' movement while simulating the UWB position error by incorporating the UWB error model. Further details of this evaluation will be presented in Section V-C. For evaluating the performance of the feature-based modification algorithm, instead of collecting images ourselves from both primary and secondary users, we utilize an existing multi-view image dataset specifically designed for AR research. The details of this evaluation will be presented in Section V-D.

After obtaining the localization data and image data, we proceed to evaluate the multi-user AR system using our prototype. Initially, the primary user creates a virtual object in his own coordinate system using ARCore. We then input the simulated UWB position data and IMU-deadreckoning data into our IMU/UWB fused localization algorithm, which allows us to obtain the predicted primary user's position. Subsequently, we derive the virtual object's position in the secondary user's coordinate system. Finally, we apply the feature-based modification algorithm to adjust the position and overlay the virtual object on the secondary user's screen. The overall performance evaluation will be presented in Section V-E.

A. Energy Consumption

In the evaluation, the primary user places a virtual object. By synchronizing with the primary, the secondary users can see the object in their screens. For comparison, we have implemented three methods for the secondary to overlay the object.

- **DNN:** The secondary user runs a DNN-based object detection algorithm (MediaPipe Objectron [14]) to decide where to overlay the virtual object.
- **Mobile-SLAM:** The primary runs DNN-based object detection algorithm and mobile-SLAM, and the secondary overlays the object by synchronizing with the primary through mobile-SLAM.
- **Ours:** The primary runs the DNN-based object detection algorithm, and the secondary overlays the virtual object through our proposed synchronization method.

The three methods are implemented on Google Pixel 6 pro smartphones with Android Studio and run for 30 minutes. We log the total energy consumption by using Android's Battery Manager. Since our focus is to evaluate the energy consumption of the AR application, we ignore the basic energy consumption

such as the energy consumed by Android OS, camera, and the screen display (brightness set to 70%). The energy consumption shown in Fig. 5 reflects the overall energy usage, which includes computation, communication, and sensor power, after subtracting the baseline energy consumption.

Fig. 5(a) compares the energy consumption of the secondary user in three different mobile AR methods. Compared to running DNN-based object detection for overlaying the virtual object, our method can reduce the energy consumption by 81%. This is because our method does not involve running the computational intensive DNN models for object detection. Instead, through our method, the virtual object can be re-projected from the primary's coordinate to the secondary's coordinate system. Compared to running mobile-SLAM for overlaying the virtual object, our method can reduce the energy consumption by 69%. This is because mobile-SLAM needs to constantly construct the 3D map of the environment and track the 6DoFs in the 3D map. However, our method does not involve the 3D map construction and thus saves a lot of energy.

In our proposed method, the energy consumption comes from three modules: 1) our IMU/UWB based localization algorithm (IMU/UWB ARL), 2) our feature-based position modification algorithm (FPM), and 3) the communication with the primary user (Communication). Among them, most of the energy consumption comes from IMU/UWB ARL, which takes about 66% of the total energy consumption. This is due to the continuous operation of the IMU and UWB sensors throughout the AR application. The second largest energy consumption is from the communication module, as the primary has to continually send synchronization data to the secondary users. Our feature-based position modification algorithm consumes the least amount of energy since it only uses a few feature points to adjust the positions of the virtual objects without the need for complex computations such as constructing a 3D map of the environment in SLAM.

Fig. 5(b) plots the energy consumption of the primary when connecting to various number of secondary users. The energy consumption comes from four modules: 1) DNN-based object detection (DNN), 2) IMU and UWB sensors' operation (IMU/UWB), 3) ORB feature extraction (ORB), and 4) the communication with the secondary users (Communication). When there is only one secondary user, the majority of energy consumption comes from DNNs, accounting for 85% of total

TABLE II
ENERGY CONSUMPTION (MAH) ACROSS DIFFERENT SMARTPHONES

Device	Secondary User			Primary User		
	DNN	Mobile-SLAM	Ours	$N=1$	$N=10$	$N=20$
Pixel 6 Pro	491.5	304.5	92.0	578.3	612.6	637.7
Pixel 10 Pro	452.3	282.4	86.5	538.5	569.0	593.2
Galaxy S25+	444.0	242.7	82.4	526.4	547.8	577.1

energy usage (with DNN of 491 mAh and the total energy consumption of 578 mAh). When the number of users increases, the primary only needs to transmit the synchronization data to multiple secondary users without executing DNN or ORB extractor multiple times. Thus, the additional energy consumption comes from the communication module (i.e., parallel threads for transmitting the synchronization data). Given that the primary's 3D orientation, motion, and the virtual object's information can be represented by several floating point numbers, the primary component of the synchronization data is the visual features. To enhance the position accuracy of the virtual object, we extract 20 ORB feature points. As each ORB feature point occupies 32 Bytes [28], the size of the visual feature is $32 \times 20 = 640$ Bytes. Thus, the total size of the synchronization data is about 700 Bytes. Due to the small size of the synchronization data, the communication overhead (73 mAh) is small, i.e., it is less than **12%** of the total energy consumption (637 mAh) even when there are 20 secondary users.

In Fig. 5(c), we evaluate the total energy consumption (i.e., the sum of the energy consumption by the primary and secondary users) of the three methods when there are one primary user and multiple secondary users. As shown in the figure, our synchronization method always outperforms the other two in terms of energy consumption. Moreover, our method is much more scalable. As the number of secondary users increases from one to twenty, the energy consumption of our method increases about $2363-614 = 1749$ mAh. However, the energy consumption of the DNN method increases about $10321-982=9339$ mAh, which is over $4 \times$ **higher** than our method. This is because our method only runs the computationally intensive DNN model on the primary user, while others just re-project the virtual object in their own coordinate system. Similar trend can be found for mobile-SLAM.

To assess the generalizability of our system across different smartphones, we conducted the same energy measurements on two additional devices: Google Pixel 10 Pro and Samsung Galaxy S25+, as shown in Table II. When the device is used as a secondary user, our method reduces energy consumption by approximately 80% compared to DNN and over 65% compared to mobile-SLAM across all tested smartphones. When the device is used as the primary user, the energy consumption shows a similar scaling trend across devices, increasing by only about 10% as the number of secondary users grows from $N = 1$ to $N = 20$.

B. Synchronization Delay

Both communication and computation latency affect the synchronization delay, which represents the time our system takes from the primary user being aware of the virtual object's position

TABLE III
SYNCHRONIZATION DELAY (MS)

Content	Device	Latency (ms)	
		single-secondary	20-secondary
Data transmission	Pixel 6 Pro	6.4	9.3
	Pixel 10 Pro	6.2	9.8
	Galaxy S25+	6.3	9.4
Feature-based modification	Pixel 6 Pro	7.6	
	Pixel 10 Pro	3.9	
	Galaxy S25+	2.9	
IMU/UWB ARL	Pixel 6 Pro	7.1	
	Pixel 10 Pro	3.9	
	Galaxy S25+	2.2	
End-to-End	Pixel 6 Pro	21.2	24.0
	Pixel 10 Pro	14.1	17.6
	Galaxy S25+	11.4	14.5

to the secondary knowing its position in his own view. There are three phases in a single end-to-end synchronization:

- Data transmission delay: the primary transmits the synchronization data (introduced in Section II) to the secondary.
- IMU/UWB ARL: the computation latency to update the primary's relative 3D position in the secondary's view by our AR Localization algorithm.
- Feature-based modification: the process of modifying the virtual object's position in the secondary's view, which includes the time for ORB feature extraction, points matching, and position adjustment.

In our experiment, the primary and the secondary users are connected through a single-hop Wi-Fi network in our lab. We measure the latency when using 5 GHz wireless channels. Table III shows the communication and computation latency of synchronizing two different numbers of secondary users with the primary user, one secondary user and 20 secondary users.

First, due to the small size of the synchronization data, the transmission delay only takes about 6 ms in the single-secondary scenario and about 9 ms in 20-secondary scenario across different smartphones. The slight increase comes from the overhead of multithreading TCP transmission.

Second, Feature-based position modification only takes less than 8 ms depending on the device. This is because our system only extracts the ORB features within the bounding box, which represents a small proportion of the overall image, and does not need to construct the 3D map of the environment in SLAM. Increasing the number of secondary users does not affect this latency since each secondary user modifies the virtual object's position within his own view. As expected, more advanced devices such as the Galaxy S25+ achieve this in only 2.9 ms, due to more powerful CPUs for ORB feature extraction.

Third, our IMU/UWB Fused AR Localization algorithm updates the primary's 3D position with short overhead, i.e., 2.2 ms on Galaxy S25+. By only spreading the particles near the measured position, we effectively decrease the number of particles in each iteration and thus the algorithm is lightweight. Increasing the number of secondary users does not affect this latency since each secondary user calculates the primary's relative 3D location, without involving other secondary users.

TABLE IV
UWB ERROR MODELS

	Distribution	Parameters	Scene
Scenario 1	$\mathcal{N}(\mu, \sigma)$	$\mu = 0.1, \sigma = 0.05$	LOS/QLOS
Scenario 2	$\mathcal{N}(\mu, \sigma)$	$\mu = 0.2, \sigma = 0.1$	LOS/QLOS
Scenario 3	$\mathcal{N}(\mu, \sigma)$	$\mu = 0.3, \sigma = 0.15$	LOS/QLOS
Scenario 4	$\Gamma(\alpha, \lambda)$	$\alpha = 1.5, \lambda = 1.3$	NLOS

Overall, the end-to-end latency, from the primary user determining the virtual object's location to the secondary user rendering it in their own view, is below 22 ms in the *single-secondary* scenario and under 25 ms in the *20-secondary* scenario. This enables frequent and continuous synchronization (i.e., 41–88 fps) between the primary and secondary users. This low latency allows the primary user to run advanced DNN-based AR applications [14] without degrading their frame rate (approximately 36 fps on modern smartphones), ensuring that synchronization is not a performance bottleneck.

C. Multi-User AR Localization

We simulate users' movements using the Aceinna Inertial Navigation simulator [29], where one primary and five secondary users move within a 10 m $\times$ 10 m $\times$ 3 m space. Similar to [2], all users (both primary and secondary) move around a fixed location and face the location where the virtual object should be overlaid. We consider three user movement patterns: (1) a rectangular path with 90° turns, (2) a circular path with a radius of approximately 5 m, and (3) a back-and-forth straight-line path of approximately 6 m. The movement speed ranges from 0 to 1.5 m/s, representing a typical human walking pace. Each movement is repeated 50 times with different initial user locations, and each trial lasts 48 seconds. The simulator's IMU381ZA model provides all necessary IMU data, including acceleration and orientation, associated with the users' movement.

Since UWB localization accuracy is known to degrade due to body-induced interference when a user holds the mobile device [19], we incorporate this effect into our simulation by modeling the UWB distance error based on the empirical error distributions measured in [19]. This model incorporates two distinct distributions: 1) the Gaussian distribution, which represents line-of-sight (LOS) and quasi-line-of-sight (QLOS) scenarios, and 2) the Gamma distribution, which represents non-line-of-sight (NLOS) scenarios. We consider four scenarios with varying noise scales and distributions as shown in Table IV.

We compare our algorithm with the following baselines:

- *UWB-only*: The secondary obtains the primary's relative 3D position only through UWB ranging.
- *EKF*: the Extended Kalman Filter, which is a sensor fusion algorithm used for indoor localization [21].
- *Naive PF*: Most existing work [22], [24], [25], [26] leverages particle filters to integrate IMU and UWB for indoor localization. However, their focus is to find the user's absolute 3D position and hence need some anchor nodes. However, our focus is to find the primary's position relative

to the secondary. As a result, their solutions can not be directly applied, and have to be modified. In Naive PF, the particles are spread into the area around the measured position by UWB ranging. Once the primary's position is predicted, a set of new particles is resampled around the measured position to enable the next round of prediction, which follows the resampling procedure in the initialization stage.

- *Ours w/o KDE*: This is based on our IMU/UWB fused AR Localization algorithm. However, all particles in the new sphere are assigned equal weight, which does not use our KDE based weight reassignment method.

Evaluation Results: As shown in Fig. 6, across all four scenarios (based on Table IV), our algorithm outperforms all baselines and can reduce the 3D position error by over 50% compared to the original UWB. Specifically, in the four scenarios, our algorithm outperforms EKF by 45.8%, 46.1%, 43.4%, and 50.3%, respectively. This is because EKF assumes both the movement calculated by IMU dead-reckoning and the position measured by UWB exhibit zero-mean Gaussian noise. However, the movement derived from IMU dead-reckoning involves the double integration of linear acceleration, which includes Gaussian white noise as well as random walk noise. As a result, the movement calculated by IMU dead-reckoning in each time slot does not conform to Gaussian distribution. In contrast, our algorithm, by using a particle filter, exhibits superior performance in handling non-Gaussian noise.

When comparing our algorithm to the Naive PF method, we observe improvements of 44.1%, 40.1%, 39.8%, and 51.8% in the four scenarios. This enhancement can be attributed to the resampling stage of our algorithm, where new particles are selected around the predicted position that is closer to the ground truth compared to the measured position. In addition, our algorithm preserves the posterior predictive distribution well for particles from previous time slots by using KDE, while Naive PF loses such important information by assigning resampled particles with equal weights. Compared to Ours w/o KDE, our algorithm can improve performance by 9.9%, 13.2%, 15.7%, and 18.7% in the four scenarios.

Here, we did not compare our algorithm with techniques solely based on IMU, where the accumulated errors can lead to a huge drift in localization. Based on our measurements, it can be several magnitudes of high error than UWB. Note that the IMU sensor in our algorithm is only used for calculating the user's movement in a short duration (i.e., in the level of 0.1 s).

Fig. 7 plots the relative 3D position error of a secondary user over time during a 48-second movement. Since the initial 3D position is only determined by UWB ranging, all methods share the same initial 3D position error, which is about 26 cm. When users move, relying solely on IMU leads to significant position error over time due to error accumulation. Compared to only using UWB ranging, leveraging both UWB and IMU (i.e., EKF and ours) can effectively reduce the 3D position error. Furthermore, our method decreases the mean 3D position error by 42.3% and decreases the variance by 70% compared to EKF, which demonstrates better stability compared to EKF.

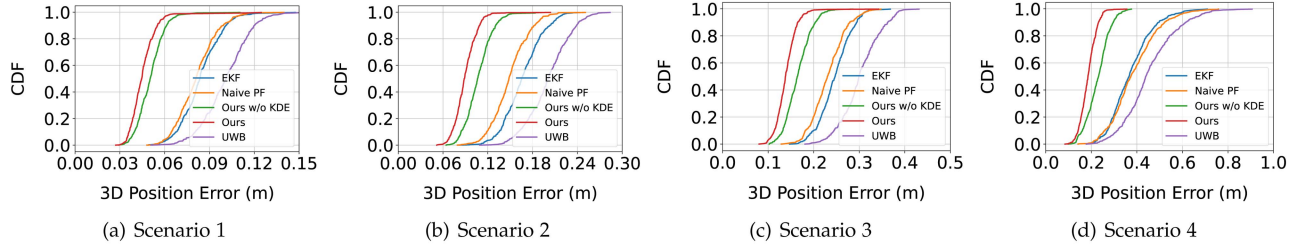


Fig. 6. 3D position error comparisons in different scenarios.

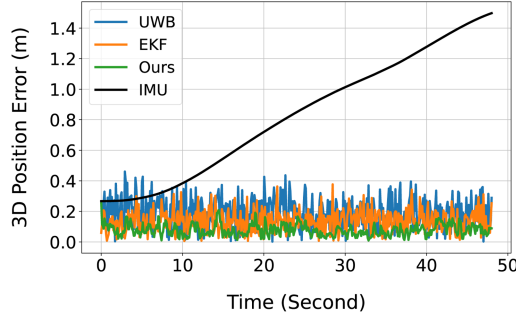


Fig. 7. The stability of our method compared to other methods during movement.

D. Feature-Based Position Modification

Since the evaluation requires a large amount of images, we use two multi-view image datasets, especially collected for mobile AR research - one is MVMDD dataset [3] and the other one is Objectron dataset [30]. In the two image datasets, each object is photographed from different perspectives, thus can simulate the perspectives of different users in the multi-user AR scenario. For each object in the two datasets, we choose 12 pairs of images. Within each pair, one is for the primary's view, and the other is for the secondary's view. The view difference within each pair is either 60 or 120 degrees.

To use these datasets, we should identify the target's position in the users' views using bounding boxes so that the virtual object can be overlaid at the right place. As the original dataset lacks bounding box annotations, we manually label a bounding box for each image as the ground truth. To simulate the secondary's view after projection, the bounding box is shifted for some distance. Such distance shift, i.e., the difference between the bounding boxes in the secondary's view and the ground truth, is referred to as *AR-Error*. This metric includes horizontal and vertical differences, for example, 0.2 horizontal AR-Error indicates that the bounding box (or virtual object) has shifted from its ground truth horizontally by 20% of the image's width.

Evaluation Results: Fig. 6 shows the evaluation results based on the aforementioned datasets. In this figure, the x-axis denotes the AR-Error in the secondary user's view before running our feature-based algorithm and the y-axis denotes the AR-Error after running the algorithm. We also draw a baseline, which illustrates there is no difference before and after running the algorithm, i.e., points under this line mean that running our algorithm improves the performance.

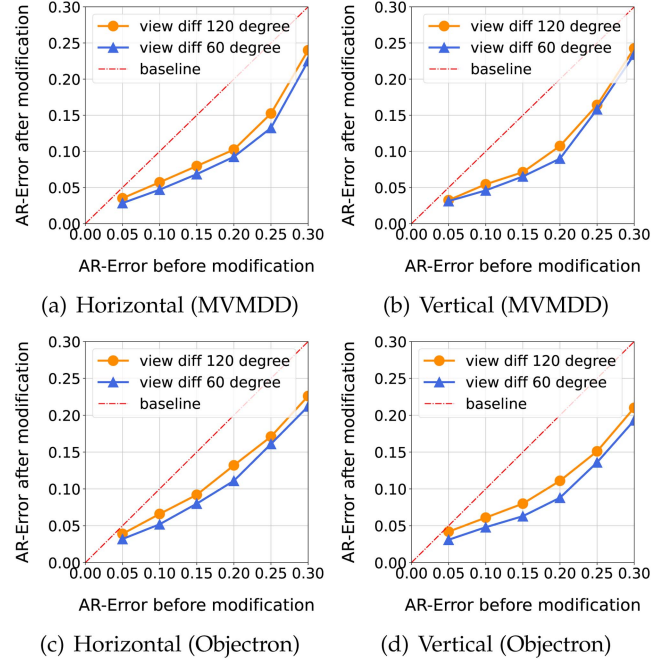


Fig. 8. AR-error reduction with our feature-based approach.

In this evaluation, we choose the view difference of either 60 or 120 degrees. As shown in Fig. 8(a) and (c), by running our algorithm, the horizontal AR-Error can be reduced. For example, with a view difference of 60° and the AR-Error of 0.20, our algorithm can reduce the AR-Error to 0.10 in MVMDD dataset and 0.11 in Objectron dataset. When the AR-Error before modification is small, i.e., from 0.05 to 0.20, our algorithm achieves 40% to 50% improvement on both of the two datasets. However, as the AR-Error keeps increasing, i.e., from 0.2 to 0.3, the overlap between the secondary's bounding boxes and the ground truth becomes smaller, such that fewer feature points can be matched between the primary's and secondary's view. Since our position modification is based on the feature points, fewer matched feature points result in less accuracy, reducing the performance improvement from 50% to 25%. Similar phenomena can be seen for the vertical AR-Error, as shown in Fig. 8(b).

We also observe that our algorithm is slightly affected by the view difference. Specifically, when the view difference is large (120°), our algorithm exhibits slightly lower performance compared to that with small view difference (60°). This is because more featured points with high matching degrees can

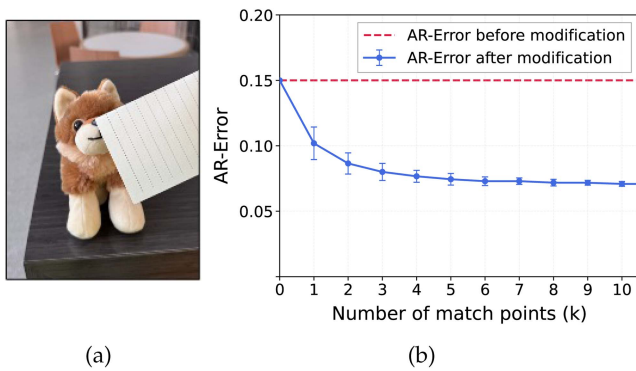


Fig. 9. (a) Occlusion of secondary's view. (b) AR-Error under varying numbers of matched feature points.

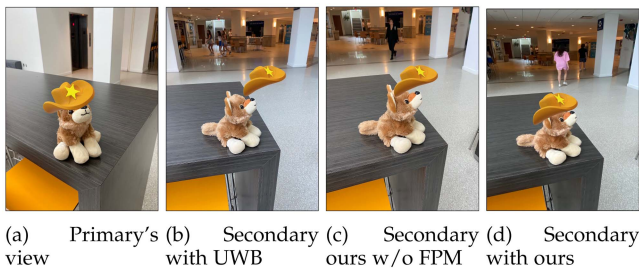


Fig. 10. Demo based on the proposed synchronization method.

be found with a smaller view difference, resulting in higher accuracy.

Occlusion of secondary's view: In practice, the object in the secondary user's view may be partially or fully occluded by other objects or people in the environment, leading to fewer matched feature points within their bounding boxes. To show how the number of matched feature points affects the accuracy of our FPM algorithm, we conducted an additional experiment, where the object in the secondary user's view is partially occluded, as shown in Fig. 9(a). By adjusting this occlusion, different numbers of matched feature points can be found. The FPM's performance is shown in Fig. 9(b), where k is the number of matched feature points after the occlusion. Generally, with more matched feature points, our FPM achieves lower AR-Error since averaging over more matched feature points is more robust to noises and variance, yielding a more accurate offset estimation. In some cases, even there are severe occlusions ($k = 1$), FPM still reduces the AR-Error significantly. This is because a single matched feature point can provide a valid estimate of the bounding box offset in the secondary user's view.

E. End-to-End Evaluation

After demonstrating the superior performance of our IMU/UWB fused algorithm and our feature-based modification algorithm, we focus on evaluating the performance of our whole system in this subsection.

We first use Fig. 10 to demonstrate the effects of our light weight synchronization. In this demo, as shown in Fig. 10(a), the primary user places a virtual hat on the fox's hat through ARCore. By synchronizing with the primary with our system,

the secondary can see a virtual hat on the fox's head from a different view. Since UWB may introduce some error, Fig. 10(b) illustrates the effect when the position of the primary is shifted 10 cm due to UWB measurement error. Due to this error, the virtual hat cannot be placed in the right position in the secondary user's view. Fig. 10(c) illustrates the effect after running our IMU/UWB fused localization algorithm, which can correct some of the UWB position error. As a result, the virtual hat is closer to the right position in the secondary's view. Fig. 10(d) illustrates the effect after running our feature-based algorithm, which can further correct the position error. As a result, the virtual hat is overlaid to the right position in the secondary's view.

To quantify the performance improvement of the proposed techniques, we conduct an end-to-end assessment by feeding IMU/UWB data and real-world image data into our AR system and then measure the AR-Error of the virtual object displayed on the secondary user's screen.

The evaluation has several steps. Initially, the primary user places a virtual object in his own coordinate system based on the detection result of the objects from the multi-view image dataset [3]. Following the same setup as Section V-C, we simulated multiple AR users' trajectories using the inertial navigation simulator and generated UWB ranging data with different error models (Scenarios 1–4). By leveraging IMU/UWB AR localization algorithm, we obtain the estimated relative locations between primary and secondary users. Then, we project the virtual object from the primary's coordinate to the secondary user's coordinate system. Since the simulator also provides ground truth, we can quantify the AR-Error associated with UWB ranging as well as our localization algorithm. Finally, to further reduce the AR-Error, we use the feature-based algorithm to adjust the position of the virtual object in the secondary's view.

In real-world multi-user AR scenarios, the 3D position error of the virtual object can result in varying AR-Errors on the secondary user's screen, depending on the distance between the virtual object and the secondary user. Specifically, when the virtual object is closer to the user, the AR-Error tends to be larger, and vice versa. To take this into consideration, we incorporate varying distances between the virtual object and the secondary user, and consider three different distances: 1 m, 2 meters, and 3 meters.

Evaluation Results: In Fig. 11, the x-axis denotes the distance from the secondary user to the virtual object and the y-axis denotes the horizontal AR-Error. We compare three methods: *UWB*, *Ours w/o FPM*, and *Ours*. The UWB method represents that the secondary obtains the primary's relative 3D position only through UWB ranging. The Ours method represents our approach which incorporates both our IMU/UWB fused algorithm and the Feature-based Position Modification (FPM) algorithm. Ours w/o FPM refers to our algorithm without the FPM component.

As can be seen from the figure, when the distance is one meter, compared to synchronizing the primary and secondary user only by UWB ranging, ours w/o FPM can decrease the AR displaying error by 62.7%, 58.5%, 56.8%, 57.2% in the four different scenarios (based on Table IV). By incorporating FPM, the AR-Error

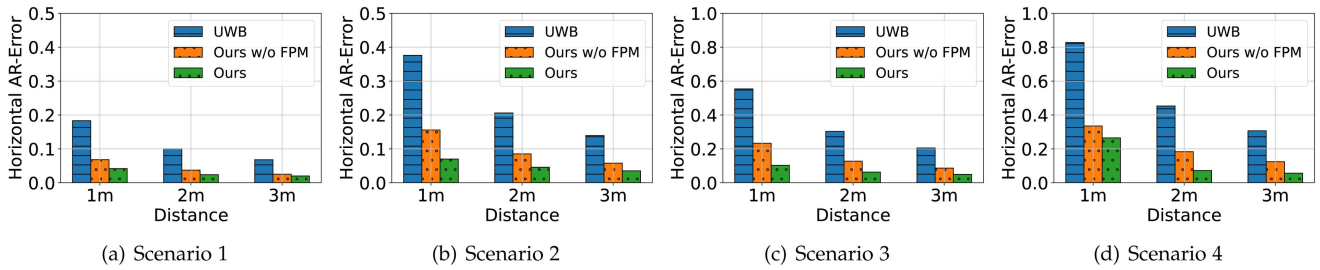


Fig. 11. End-to-end performance evaluations.

can be reduced by 77.0%, 82.4%, 81.3%, and 67.9%. As the distance between the virtual object and the secondary increases, i.e., from one meter to three meters, all methods achieve lower AR-Error, while Ours always outperforms the other two.

VI. RELATED WORK

Multi-user augmented reality: Multi-user AR has received considerable attention in recent years. To synchronize the coordinates of different users, various methods have been proposed and can be classified into two categories. The first category assumes that users have the same offline 3D map of the environment and they can track their own orientations and positions in the same 3D map by SLAM [31], [32], [33], [34]. However, these methods will not work if the environment is not pre-mapped. Our proposed method does not have this limitation since it calculates the relative position and orientation between multiple users without relying on the absolute position and orientation information in the environment. The secondary category is to perform coordinates synchronization between multiple users at the start of the AR session [2], [3] and utilize SLAM to track user movements. Due to the high energy consumption of SLAM, FreeAR [10] leverages IMU sensors for tracking the secondary user's position although it still uses SLAM for tracking the primary user's position. Furthermore, SLAM is constrained by static environment assumptions and performs poorly in real-world dynamic or textureless scenarios [35]. Although recent research [36], [37], [38], [39] has applied deep learning models such as Mask-RCNN [40] to fix the drawbacks, it also significantly increases the energy consumption, which is not suitable for mobile AR. Different from them, instead of relying on SLAM for synchronization, we leverage the embedded IMU and UWB sensors to save energy and achieve better synchronization.

IMU/UWB fused localization: Given the complementary nature of IMU and UWB, many recent studies have focused on combining these sensor modalities to achieve precise indoor localization [21], [22], [24], [25], [26], [41] or gesture recognition for the AR scenario [42]. Some of them have employed particle filters to fuse the IMU and UWB sensing data [22], [24], [25], [26]. However, their main focus is on localizing the user's precise position within an indoor environment by utilizing distance measurements from the mobile device to pre-installed UWB anchors. As a result, they disperse a large number particles across the entire area where the user can move, resulting in high computational overhead and slower convergence of the

localization process. In multi-user AR, our objective is to determine the primary's position relative to the secondary. Hence, we only spread a small number of particles near the measured position and then design an adaptive re-sampling method to further improve the localization accuracy.

VII. CONCLUSION

In this paper, we proposed a lightweight synchronization approach for multi-user AR by leveraging multiple embedded sensors present in mobile devices, such as IMU, UWB, and camera sensors. Our proposed solution introduces a novel AR localization algorithm for tracking the relative positions of AR users. This algorithm utilizes particle filters to fuse the data from IMU and UWB sensors, while incorporating an adaptive resampling method to enhance localization accuracy. To further improve the positional accuracy of virtual objects displayed in AR, we have developed a feature-based position modification algorithm. Extensive evaluation results show that the proposed AR localization algorithm can reduce the position error by over 50% compared to other UWB methods. Moreover, we have developed a prototype based on smartphones, and demonstrate that our synchronization approach enables precise virtual object overlaying while reducing the energy consumption by 69% compared to SLAM-based synchronization methods.

REFERENCES

- [1] C. Slocum, Y. Zhang, E. Shayegani, P. Zaree, N. Abu-Ghazaleh, and J. Chen, "That doesn't go there: Attacks on shared state in Multi-User augmented reality applications," in *Proc. USENIX Secur. Symp.*, 2024, pp. 2761–2778.
- [2] X. Ran, C. Slocum, Y.-Z. Tsai, K. Apicharttrisor, M. Gorlatova, and J. Chen, "Multi-user augmented reality with communication efficient and spatially consistent virtual objects," in *Proc. Int. Conf. Emerg. Netw. Exper. Technol.*, 2020, pp. 386–398.
- [3] Z. Liu, G. Lan, J. Stojkovic, Y. Zhang, C. Joe-Wong, and M. Gorlatova, "CollabAR: Edge-assisted collaborative image recognition for mobile augmented reality," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2020, pp. 301–312.
- [4] W. Zhang, B. Han, and P. Hui, "SEAR: Scaling experiences in multi-user augmented reality," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 5, pp. 1982–1992, May 2022.
- [5] Y. Guan, X. Hou, N. Wu, B. Han, and T. Han, "DeepMix: Mobility-aware, lightweight, and hybrid 3D object detection for headsets," in *Proc. Annu. Int. Conf. Mobile Syst., Appl. Serv.*, 2022, pp. 28–41.
- [6] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 756–764.
- [7] T. Tan and G. Cao, "Deep learning on mobile devices through neural processing units and edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1209–1218.

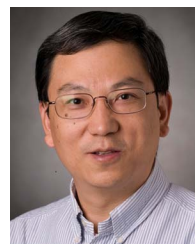
- [8] K. Yang, J. Yi, K. Lee, and Y. Lee, "Flexpatch: Fast and accurate object detection for on-device high-resolution live video analytics," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1898–1907.
- [9] "Google ARCore," 2026. [Online]. Available: <https://developers.google.com/ar>
- [10] K. Apicharttrisor, J. Chen, V. Sekar, A. Rowe, and S. V. Krishnamurthy, "Breaking edge shackles: Infrastructure-free collaborative mobile augmented reality," in *Proc. 20th ACM Conf. Embedded Netw. Sensor Syst.*, 2022, pp. 1–15.
- [11] Y. Zhang, X. Weng, and K. Ling, "Poster Abstract: Logging measurements from UWB and AR sensors on iOS devices," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2024, pp. 293–294.
- [12] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," in *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.
- [13] P. Li, T. Qin, B. Hu, F. Zhu, and S. Shen, "Monocular visual-inertial state estimation for mobile augmented reality," in *Proc. IEEE Int. Symp. Mixed Augmented Reality*, 2017, pp. 11–21.
- [14] T. Hou, A. Ahmadyan, L. Zhang, J. Wei, and M. Grundmann, "MobilePose: Real-time pose estimation for unseen objects with weak shape supervision," 2020, *arXiv:2003.03522*.
- [15] "Android UWB ranging," 2026. [Online]. Available: <https://developer.android.com/develop/connectivity/uwb>
- [16] A. Solin, S. Cortes, E. Rahtu, and J. Kannala, "Inertial odometry on handheld smartphones," in *Proc. Inf. Fusion*, 2018, pp. 1–5.
- [17] J. Gong, X. Zhang, Y. Huang, J. Ren, and Y. Zhang, "Robust inertial motion tracking through deep sensor fusion across smart earbuds and smartphone," in *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 5, pp. 1–26, 2021.
- [18] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao, "A reliable and accurate indoor localization method using phone inertial sensors," in *Proc. ACM Conf. Ubiquitous Comput.*, 2012, pp. 421–430.
- [19] T. Otim, A. Bahillo, L. E. Diéz, P. Lopez-Iturri, and F. Falcone, "Impact of body wearable sensor positions on UWB ranging," *IEEE Sensors J.*, vol. 19, no. 23, pp. 11449–11457, Dec. 2019.
- [20] S. Shen, M. Gowda, and R. R. Choudhury, "Closing the gaps in inertial motion tracking," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 429–444.
- [21] L. Yao, Y.-W. A. Wu, L. Yao, and Z. Z. Liao, "An integrated IMU and UWB sensor based indoor positioning system," in *Proc. Int. Conf. Indoor Positioning Indoor Navigation*, 2017, pp. 1–8.
- [22] Y. Zhang, X. Tan, and C. Zhao, "UWB/INS integrated pedestrian positioning for robust indoor environments," *IEEE Sensors J.*, vol. 20, no. 23, pp. 14401–14409, Dec. 2020.
- [23] W. Xiao, W. Ni, and Y. K. Toh, "Integrated Wi-Fi fingerprinting and inertial sensing for indoor positioning," in *Proc. Int. Conf. Indoor Positioning Indoor Navigation*, 2011, pp. 1–6.
- [24] Q. Tian, I. Kevin, K. Wang, and Z. Salcic, "Human body shadowing effect on UWB-based ranging system for pedestrian tracking," *IEEE Trans. Instrum. Meas.*, vol. 68, no. 10, pp. 4028–4037, Oct. 2019.
- [25] M. Khalaf-Allah, "Particle filtering for three-dimensional TDOA-based positioning using four anchor nodes," *Sensors*, vol. 20, no. 16, 2020, Art. no. 4516.
- [26] Q. Tian, I. Kevin, K. Wang, and Z. Salcic, "A resetting approach for INS and UWB sensor fusion using particle filter for pedestrian tracking," *IEEE Trans. Instrum. Meas.*, vol. 69, no. 8, pp. 5914–5921, Aug. 2020.
- [27] A. Doucet, S. Godsill, and C. Andrieu, "On sequential monte carlo sampling methods for Bayesian filtering," *Statist. Comput.*, vol. 10, pp. 197–208, 2000.
- [28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vis.*, 2011, pp. 2564–2571.
- [29] "Inertial Navigation simulator," 2024. [Online]. Available: <https://github.com/Aceinna/gnss-ins-sim>
- [30] A. Ahmadyan, L. Zhang, A. Ablavatski, J. Wei, and M. Grundmann, "Objectron: A large scale dataset of object-centric videos in the wild with pose annotations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 7818–7827.
- [31] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan, "AVR: Augmented vehicular reality," in *Proc. ACM Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2018, pp. 81–95.
- [32] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz, "Marvel: Enabling mobile augmented reality with low energy and low latency," in *Proc. ACM Embedded Netw. Sensor Syst.*, 2018, pp. 292–304.
- [33] A. Dhakal, X. Ran, Y. Wang, J. Chen, and K. Ramakrishnan, "SLAM-share: Visual simultaneous localization and mapping for real-time multi-user augmented reality," in *Proc. 18th Int. Conf. Emerg. Netw. EXperiments Technol.*, 2022, pp. 293–306.
- [34] X. Zhang et al., "Map++: Towards user-participatory visual SLAM systems with efficient map expansion and sharing," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2024, pp. 633–647.
- [35] T. Hu, T. Scargill, Y. Chen, G. Lan, and M. Gorlatova, "Demo abstract: DNN-based SLAM tracking error online estimation," in *Proc. Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–3.
- [36] J. Xu et al., "Edge assisted mobile semantic visual SLAM," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1828–1837.
- [37] J. Zhang et al., "Edge assisted real-time instance segmentation on mobile devices," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst.*, 2022, pp. 537–547.
- [38] M. Gonzalez, E. Marchand, A. Kacete, and J. Royan, "TwistSLAM: Constrained SLAM in dynamic environment," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 6846–6853, Jul. 2022.
- [39] E. Dong, J. Xu, C. Wu, Y. Liu, and Z. Yang, "Pair-Navi: Peer-to-peer indoor navigation with mobile visual SLAM," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1189–1197.
- [40] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2980–2988.
- [41] J. Yu, P. Wang, T. Koike-Akino, and P. V. Orlik, "Multi-modal recurrent fusion for indoor localization," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2022, pp. 5083–5087.
- [42] H. Oh, S. Bhattacharya, and S. Seo, "UWB/IMU-assisted gesture recognition using learning approaches for VR/XR applications," in *Proc. IEEE Int. Conf. Commun.*, 2024, pp. 4838–4844.



Zida Liu (Student Member, IEEE) received the BE degree from the Huazhong University of Science and Technology and the MS degree in electronic and computer engineering from Duke University. He is currently working toward the PhD degree with Pennsylvania State University. His research interests include mobile computing, augmented reality, and edge-assisted system.



Ruiqi Wang received the BE degree from the University of Electronic Science and Technology of China and the MS degree in computer science from the University of Delaware. He is currently working toward the PhD degree with Pennsylvania State University. His research interests include mobile computing, edge computing, and vehicular networks.



Guohong Cao (Fellow, IEEE) received the BS degree in computer science from Xi'an Jiaotong University, and the PhD in computer science from Ohio State University in 1999. Since 1999, he has been with the Department of Computer Science and Engineering, Pennsylvania State University, where he is currently a distinguished professor. He directs the Mobile Computing and Networking Lab and co-directs the Institute for Networking and Security Research. He has authored or coauthored more than 250 papers in his areas of research, which include wireless networks, mobile computing, machine learning, wireless security and privacy, and Internet of Things, which have been cited more than 25000 times. He has served on the editorial board of *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Wireless Communications*, and *IEEE Transactions on Vehicular Technology*, and has served on the organizing and technical program committees of many conferences, such as the TPC chair/co-chair of IEEE SRDS, MASS, and INFOCOM. He was the recipient of the several best paper awards, IEEE INFOCOM Test of Time Award, and NSF CAREER Award. He is a fellow of the AAAS.