Decentralized Visual-Inertial Localization and Mapping on Mobile Devices for Augmented Reality

Kourosh Sartipi, Ryan C. DuToit, Christopher B. Cobar, and Stergios I. Roumeliotis[†]

***Abstract*— In this paper, we present a novel approach to shared augmented reality (AR) for mobile devices operating in the same area that does not rely on cloud computing. In particular, each user's device processes the visual and inertial data received from its sensors and almost immediately broadcasts a partial feature map of its surroundings. In parallel, every device localizes against the broadcasted maps by employing feature observations to determine its 4-DOF transformation to each of the gravity-aligned maps. By doing so, virtual content placed by any of the users is quickly and accurately displayed in all other users' views. Furthermore, to reduce the effect of inconsistency introduced by relocalizing against incrementally created and updated maps, their transformations w.r.t. the device are modeled as random processes driven by white noise instead of constant, unknown parameters. Lastly, we assess the accuracy of our approach for the case of two users in a room-scale environment against VICON ground-truth.**

I. Introduction

With the increasing availability and capability of mobile devices, numerous methods employing online simultaneous localization and mapping (SLAM) to estimate a device's 3D pose (position and orientation) in real-time have been developed [1], [2], [3], [4], [5]. Extending this capability, however, to multiple mobile devices in augmented reality (AR) applications, where the users place virtual objects in the environment (*e.g.*, see Fig. 1), is not trivial. Specifically, shared AR use-cases (*e.g.*, games, collaborative design, and education) require to precisely estimate, in real time, the pose of each device as well as the location of all virtual objects regardless of the frame against which each of them was anchored. In particular, all virtual objects are placed w.r.t. the "owner's" frame of reference (*i.e.*, the frame of the user placing the object). Thus, for the objects to be shared across multiple users, the transformations between their frames need to be determined. This can be achieved in a number of ways.

In particular, Lynen *et al.* [6] and Oleynikova *et al.* [7], first create an offline map of the area of interest w.r.t. a single frame which then the users employ to localize against. Although creating a map offline to be shared in advance has certain advantages (*e.g.*, high accuracy), it causes delays, burdens the users with additional effort, is not resilient to changes (*e.g.*, lighting conditions) and offers no way for updating or expanding their map after the start of the operation. Note that ARKit [8] and HoloLense [9] that compute the shared map on device, as well as ARCore [10] that relies on cloud computing share the same drawbacks of

Fig. 1. Screen captures from two mobile phones in a shared AR experience. The red R2D2 is placed by the first user (left), while the blue R2D2 is placed by second user (right). Both devices, though, view them as if they are sharing the same physical space.

[6] and [7].[1] Another class of approaches employ powerful servers or cloud infrastructure to process data received from all users, to compute and broadcast a unified map [11], [12], [13], [14], [15], [16]. Cloud computing, however, may not always be available and raises privacy concerns.

In the context of cooperative SLAM, Cunningham *et al.* introduced in [17], and later improved upon in [18], a decentralized data fusion smoothing and mapping (DDF-SAM) approach to enable consistent[2] localization against features where each device marginalizes all its other states and broadcasts the resulting dense factor and feature estimates. To reduce DDF-SAM's computational cost from quadratic to linear in the number of features, Paull *et al.* [20] presented a consistent marginalization scheme, where an approximate factor with a predetermined structure is computed to achieve communication cost *linear* in the number of features. The necessary computations, however, are often prohibitive for use in mobile devices, as they involve inversion and eigen-decomposition of the dense marginalized feature Hessian matrix. Alternatively, approaches such as [21], [22], [23], distribute the computations of the multi-user optimization problem across all devices. They require, however, communication synchronization to send the necessary data to other devices at each step of their distributed optimization process. Furthermore, they compute the optimal device poses and therefore incur extra processing because of the large number of states involved.

Approaches that separate SLAM to a fast *Localization Frontend* and a slower, but more accurate *Mapping Backend* (*e.g.*, PTAM [24]) are more suitable for mobile processors. In particular, Schuster *et al.* [25] propose an approach comprising a fast *Frontend*, which creates keyframes and local maps, and a *Backend* that employs pose-graph optimization

[†]Kourosh Sartipi, Ryan C. Dutoit, Christopher B. Cobar, and Stergios I. Roumeliotis were with Google, CA 94043 at the time of submission sarti009@umn.edu, rdutoit@google.com, cobar001@umn.edu, stergios@umn.edu.

[1]Recently ARKit introduced an update that allows users to continuously share maps. The details of their approach, however, is not publicly available.

[2]A state estimator is consistent if the estimation errors are zero mean and have covariance equal to the one calculated by the filter [19].

for processing inter and intra-device measurements. These two components, however, are decoupled as the *Frontend* does not localize against the *Backend's* map, making it susceptible to drift. Alternatively, Zhang *et al*. [26] extend ORB-SLAM [27] to multiple devices where each user shares all keyframes and corresponding feature measurements with every other user, and all the data from all devices are processed to create a single pose graph representing the entire system on every device. Similarly, in [28], the devices involved in inter-device loop-closure continuously broadcast all their keyframes and maps, and then perform independent map merging operations. Both [26] and [28], however, impose a significant communication and computational burden as all keyframes and measurements are shared and processed by each device independently.

In our work, to ensure real-time operation, we introduce a decentralized SLAM system, consisting of, per device: i) A fast *Localization Frontend* (based on a sliding-window filter [2]) and, ii) An incremental *Mapping Backend* [29] (see Fig. 2). Specifically, each user's *Frontend* estimates the host device's pose independently and performs map-based updates (corrections) using *all* available maps, created by the host device's *Backend* or transmitted from other users. In parallel, and as soon as this is available, each user's *Backend* incrementally broadcasts information about a subset of its processed features (3D position, approximate covariance, and a summarization of its visual descriptors). By doing so, each device has (almost) immediate access to a collection of users' maps, which are employed for reducing drift, and more importantly, for estimating the transformations between the users' and the maps' frames. Once these estimates become available, the system can compute the poses of all virtual objects w.r.t. all device's frames.

Key to our system are the state updates against *uncertain* maps, computed incrementally online. Similar to previous approaches (*e.g*., [6], [30], [27]) and in order to reduce processing requirements, we assume that the mapped features' positions are perfectly known. This approximation, however, leads to inconsistency[3] which we partially remedy by increasing the assumed noise covariance of the mapped feature observations. In our case, this inconsistency is further exacerbated by the fact that the maps are *evolving*, *i.e*., both expanding and improving. That is, the assumed perfectly known feature positions *change* between consecutive map broadcasts as more visual observations become available and are processed by the *Backend* to improve each map's accuracy. Note that this is a distinguishing characteristic of our system, where in order to minimize the time for starting a shared AR experience, we choose to almost immediately start sharing incomplete, and often inaccurate maps. To reduce the impact of this new type of inconsistency, without increasing processing, in this work we model the transformations between the users and the maps as random processes driven by white noise. This modeling choice allows the system to compensate for errors incurred due to inconsistency over time, resulting in higher accuracy. In summary, the main contributions and key findings of this paper are:

- To the best of our knowledge, we present the first decentralized 3D SLAM system for mobile devices that does *not* require a cloud infrastructure and where the users can *immediately* start a shared AR experience, provided they observe the same scene. The communication cost of our algorithm is linear per user in the number of users. On the other hand, map-based updates of the *Frontend* for determining the user-to-map transformations, have complexity cubic in the number of users.[4]
- We introduce a novel method for reducing the effect of inconsistency caused by performing map-based updates against incrementally built, inaccurate maps, and verify the improvements in accuracy experimentally.

In what follows, we first provide an overview of the shared AR system in Sect. II, and describe its components in detail in Sect. III. We then experimentally evaluate our algorithm in Sect. IV and provide concluding remarks in Sect. V.

II. System Overview

Consider multiple mobile devices operating in the same area and communicating through a network connection (*e.g*., via WiFi). Each device processes its own measurements, and places virtual objects in the environment anchored to *its own* frame of reference. The objective of this work is to design a system that accurately estimates, in real-time, the location of all virtual objects w.r.t. all users, regardless of which frame of reference they were originally anchored to. Without loss of generality, we consider the case of two users whose devices run the same processes (see Fig. 2). Specifically, our algorithm comprises the following components:

1) *Mapping Backend* (Sect. III-A): At the core of the *Backend* is an approximate, incremental, batch least-squares (BLS) estimator that periodically processes the available inertial and image measurements (both consecutive feature tracks and loop closures) to incrementally create a 3D map. Every time a partial map is computed, information from it (3D feature positions along with their approximate covariance and corresponding descriptors) are provided to the host's relocalization module and broadcast to the other user (Sect. III-A.1).
2) *Relocalizer* (Sect. III-B): This comprises a set of *Relocalizers*, one per map, each of which seeks to find correspondences between the features extracted in the host's current image and those found in a user's map.
3) *Localization Frontend* (Sect. III-C): The *Frontend*'s purpose is to produce accurate, frequent (10 Hz), and low-latency estimates of the device's pose. At its core is the inverse square root filter [2], which processes inertial measurements and consecutive image 2D feature tracks over a *fixed-size sliding window of keyframes*, thus maintaining real-time performance. Additionally, every

[3]To the best of our knowledge, the only approach that consistently uses previously computed maps is that of [31]. Its processing requirements, however, prevent us from employing it in the case of multiple users.

[4]This cost is negligible as compared to that of incremental mapping, especially for the case of a few users operate within a room-size area.

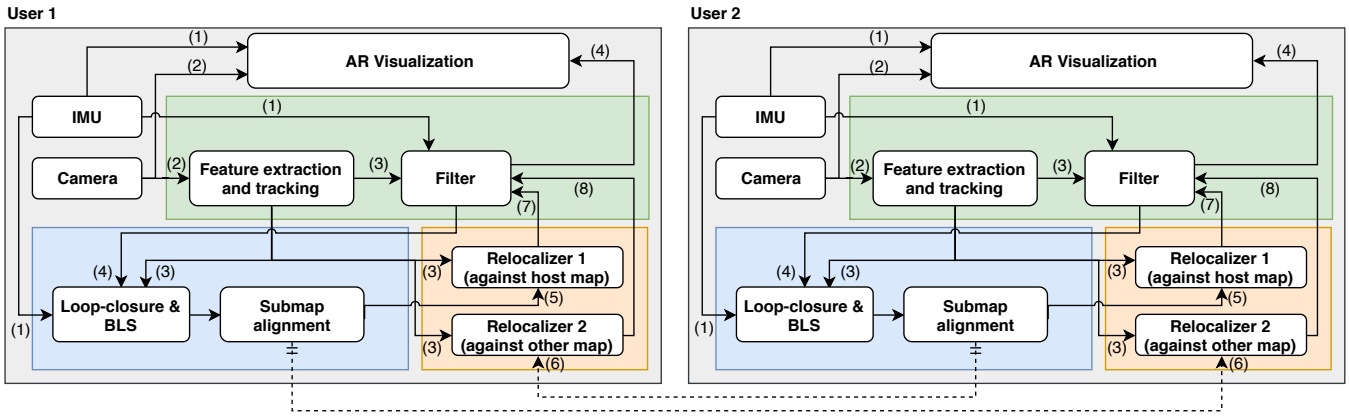

Fig. 2. Overview of the system running online on the mobile devices of two users. Shaded blue denotes the *Mapping Backend* (Sect. III-A), orange the *Relocalizer* (Sect. III-B), and green the *Localization Frontend* (Sect. III-C). Note the number of *Relocalizers* equals the number of users. The numbers on the arrows correspond to the information delivered: (1) IMU measurements, (2) images, (3) feature tracks, (4) filter state estimates, (5) host user's feature position and covariance estimates, (6) other user's feature position and covariance estimates, (7) 2D-to-3D feature matches against the host user's map, (8) 2D-to-3D feature matches against the other user's map.

time the *Relocalizer* finds correspondences between the features extracted in the current image and those in one of the users' maps, the filter processes these 2D-to-3D feature matches to reduce its pose drift and improve the estimate of its transformation w.r.t. the detected map.

III. Technical Approach

Since the communication of each user's partial map to other users is of key importance in our approach, we hereafter describe the mapping *Backend* (Sect. III-A), and then provide a brief overview of the *Relocalizer* (Sect. III-B). Subsequently (Sect. III-C), we present the *Frontend* and the process through which 2D-to-3D correspondences are employed to estimate a user's pose w.r.t the maps.

We hereafter denote the position and orientation of frame $\{F_1\}$ in frame $\{F_2\}$ as ${}^{F_2}\mathbf{p}_{F_1}$ and ${}^{F_2}_{F_1}\mathbf{C}$, respectively, where ${}^{F_2}_{F_1}\mathbf{C}$ denotes the rotation matrix corresponding to the orientation of frame $\{F_1\}$ in frame $\{F_2\}$. Additionally, we define $\mathbf{I}_\ell$, for $\ell = 2,3,4$, as the $\ell \times \ell$ identity matrix. To preserve clarity, we assume the IMU and camera are co-located and there is no time offset between the sensors' clocks. In practice, both of the quantities are estimated online [2].

A. Mapping Backend

The *Backend* employs the IMU and the feature measurements, as well as the camera pose estimates from the *Frontend* (see Fig. 2) to incrementally create a 3D map of the area. To do so, it first employs a bag-of-words approach [32] to detect loop closures and pass the matches through a 3pt+1 RANSAC [33] for outlier rejection. Then, the feature tracks, loop-closures, and IMU measurements are processed in the BLS estimator (see Fig. 2). Specifically, by defining $\{M_1\}$ as the gravity-aligned user 1's map frame of reference (*i.e.*, the z axis of the frame is parallel to the gravity direction), the pose (position ${}^{M_1}\mathbf{p}_{C_{k+1}}$ and quaternion of orientation ${}^{C_{k+1}}_{M_1}\mathbf{q}$) and velocity ${}^{M_1}\mathbf{v}_{C_{k+1}}$ of the IMU-camera pair, as well as the IMU gyroscope bias ${}^{C_{k+1}}\mathbf{b}_g$ and accelerometer bias ${}^{C_{k+1}}\mathbf{b}_a$ at time step $k+1$, form the 16×1 vector ξ_{k+1}, which is computed from ξ_k by integrating the IMU's rotational velocity and linear acceleration measurements $\mathbf{u}_k$. This process is compactly described as

$$\xi_{k+1} = \phi(\xi_k, \mathbf{u}_k) + \mathbf{n}_k^{(I)} \tag{1}$$

where ϕ is a nonlinear function corresponding to the IMU measurement model, and $\mathbf{n}_k^{(I)}$ is a zero-mean Gaussian process noise with covariance $\mathbf{Q}_k^{(I)}$. Further details on the IMU measurement model can be found in [34].

The camera observation of feature j located at position ${}^{M_1}\mathbf{f}_j$ from the pose $\{C_k\}$ is expressed as

$${}^{C_k}\mathbf{z}_j = \pi\left[{}^{C_k}_{M_1}\mathbf{C}({}^{M_1}\mathbf{f}_j - {}^{M_1}\mathbf{p}_{C_k})\right] + \mathbf{n}_{jk}^{(c)} \tag{2}$$

where π represents the camera perspective-projection model and $\mathbf{n}_{jk}^{(c)}$ is the Gaussian zero-mean measurement noise with covariance $\mathbf{Q}_{jk}^{(c)}$.

Computing the optimal BLS estimate of the trajectory consisting of K camera poses and a map of L features requires minimizing the following non-linear cost function:

$$\begin{aligned}\mathscr{C} = &\sum_{k=1}^{K-1} ||\xi_{k+1} - \phi(\xi_k, \mathbf{u}_k)||^2_{\mathbf{Q}_k^{(I)}} + \\ &\sum_{k=1}^{K}\sum_{j=1}^{L} \left\| {}^{C_k}\mathbf{z}_j - \pi\left[{}^{C_k}_{M_1}\mathbf{C}({}^{M_1}\mathbf{f}_j - {}^{M_1}\mathbf{p}_{C_k})\right]\right\|^2_{\mathbf{Q}_{jk}^{(c)}}\end{aligned} \tag{3}$$

by employing Gauss-Newton iterative optimization. Once the map is optimized, we prepare for publication (see Sect. III-A.1) all estimated feature positions ${}^{M_1}\mathbf{f}_j$ and their approximate covariances $\mathbf{P}_{\mathbf{f}_j\mathbf{f}_j}$. The latter are computed, under the assumption that their observing camera poses are perfectly known, by inverting the corresponding 3×3 block-diagonal elements of the Hessian matrix of (3). In order for the *Backend* to provide low-latency (typically within 2 sec) initialization of newly observed features and refinement of reobserved features, this incremental optimization process is repeated whenever ten new keyframes are received, which corresponds to typically a sec of visual and inertial data.

At this point, we should note that minimizing (3) has computational and memory cost between linear and quadratic in the number of states, which increases the latency of estimating the positions of new features as the map grows. For this reason, to ensure timely map publication, when the BLS solve time exceeds 6 sec we finalize the map containing *all* states currently being optimized in the *Backend* and start a new one. Finally, we align all maps corresponding to a user following the approach of [29] (see local-map alignment in Fig. 2). We, eventually stop the *Backend* after processing 2,000 keyframes to prevent ever-increasing memory usage. Investigating selective keyframing approaches to address this limitation is part of our future work.

1) Data Transmission to Other Users: Once a new map is published, it can be immediately employed for relocalization by the host device, as no communication is required. When transmitting information to the other users, though, care must be taken to ensure low communication cost. For instance, the entire image's feature descriptors, as well as their pixel locations and position estimates correspond to approximately 20 KB of data per keyframe per user. Considering the case of 5 users, this could amount to 1 MB of data per second. In our system, in order to preserve bandwidth, we selectively broadcast the features' descriptors, their 3D positions and approximate covariances. Specifically, when transmitting we consider two scenarios: (i) Broadcast a new feature, (ii) Update the information about a previously broadcast feature.

- *New features:* In this case, we require that the ratio of the smallest to the largest eigenvalues of the feature's covariance matrix is sufficiently large (> 0.01) so as to ensure the uncertainty is not disproportionately high in one direction. Additionally, the largest eigenvalue should be sufficiently small (< 10). These requirements prevent poor map-based updates caused by features with high uncertainty. If these criteria are met, the feature's 3D position, covariance, and descriptors corresponding to its first observation are broadcast.
- *Existing features:* Over time, subsequent observations to a known feature often occur, which supply additional descriptors and result in improved position estimates. In order to provide users with up-to-date information, but still limit the necessary bandwidth, we only broadcast: (i) Updated position and covariance information when the feature's position estimate has changed by more than 3 cm, and (ii) New descriptors with Hamming distance larger than 40 compared to the previous one.

The aforementioned parameter values are empirically selected to provide a balance of accuracy, initial localization time, and communication cost. In our experiments, this strategy results in less than 1 KB of data transmitted per user per keyframe (*i.e.*, less than 10 KB per user per second).

B. Relocalizer

Each *Relocalizer* seeks to find 2D-to-3D matches between features in a user's current image and a map. In this work, we have chosen to employ a separate *Relocalizer* per user (instead of one *Relocalizer* for all users' maps), so as to be able to process correspondences to each map separately and in parallel. For instance, Fig. 2 shows that each device contains two *Relocalizers* each corresponding to a different map, where one of them uses mapped features from the host device, while the other receives the necessary information (3D feature positions, their approximate covariances, and a subset of their observed binary descriptors) over the network.

Finding a 2D-to-3D match, *i.e.*, the closest image feature descriptor to a mapped one, has complexity linear in the number of features stored in the map, which can be prohibitive for real-time applications. To reduce this cost, we employ a hierarchical k-means tree [35] to find the approximate closest descriptor in the map for each queried image feature.

Once the image-to-map (2D-to-3D) feature matches are determined, inliers are then selected through a hamming-distance ratio test [36], followed by P3P RANSAC [37]. We subsequently apply PnP [38] to compute the camera's pose w.r.t. the map's frame. Lastly, and before we accept this match, we compare the roll and pitch estimates from PnP against those computed from the filter's gravity estimate. The inlier matches are then provided to the *Frontend* to be processed by the filter as described in the next section.

C. Localization Frontend

At each time step k, the *Frontend* extracts FAST [39] corners in the current image and tracks them, by matching their corresponding ORB descriptors [40], to the previous image (see Fig. 2). To estimate the current pose with low latency, we employ the square-root inverse filter of [2], which processes IMU measurements [see (1)] and 2D-to-2D feature tracks across a sliding window of J poses while marginalizing the older ones. Specifically, at each time step k, the filter maintains the following state vector

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_{C_{k-J+1}}^T & \dots & \mathbf{x}_{C_k}^T & \mathbf{x}_{E_k}^T & \mathbf{x}_\tau^{k\,T} \end{bmatrix}^T \tag{4}$$

where $\mathbf{x}_{C_i}$, for $i = k-J+1, \dots, k$, is the camera pose at time step i, $\mathbf{x}_{E_k} = \begin{bmatrix} {}^{C_k}\mathbf{b}_g^T & {}^{C_k}\mathbf{b}_a^T & {}^{G_1}\mathbf{v}_{C_k}^T \end{bmatrix}^T$ contains the bias of the gyroscope, ${}^{C_k}\mathbf{b}_g$, and accelerometer, ${}^{C_k}\mathbf{b}_a$, as well as the velocity ${}^{G_1}\mathbf{v}_{C_k}$ at time step k. Finally, the state comprising the 4-DOF transformations of the filter's global frame w.r.t. each of the maps' frames (i.e., the global-to-map states) is:

$$\mathbf{x}_\tau^k = \begin{bmatrix} \tau_1^{k\,T} & \dots & \tau_N^{k\,T} \end{bmatrix}^T \tag{5}$$

with $\tau_i^k \triangleq \begin{bmatrix} {}^{G_1}\theta_{M_i}^k & {}^{G_1}\mathbf{p}_{M_i}^{k\,T} \end{bmatrix}^T$, for $i = 1, \dots, N$, where N is the number of users, and ${}^{G_1}\theta_{M_i}^k$ and ${}^{G_1}\mathbf{p}_{M_i}^k$ are the relative yaw and position of the map frame $\{M_i\}$ w.r.t. the filter's global frame $\{G_1\}$ at time step k, respectively. Note that all the filters' and maps' frames are *gravity aligned*, therefore, we only need to estimate the yaw for their relative attitude. Furthermore, we should emphasise that there is no system-wide global frame among all users. Instead, each user defines its own global frame and estimates the transformation to the other users' maps.

1) Global-to-map Transformation Propagation Model: Estimating the global-to-map states τ_i^k is achieved through map-based updates. These, as mentioned earlier, cause the estimator to be inconsistent, because the mapped features are treated as known parameters and thus their cross-correlations with the filter states are ignored. Furthermore, and most importantly in the context of this work, the mapped features' position estimates may *change* over time. Specifically, and in order to expedite the start of a shared AR experience, partial and often inaccurate feature maps are broadcast by the users almost immediately. As more feature observations are processed by the users' *Backends* and re-broadcast to the *Relocalizers*, the *Frontend* repeatedly updates its state using *different* feature position estimates. This process exacerbates the effect of inconsistency and causes the filter to be overly confident about its global-to-map relative pose estimates. Eventually, as a result, the filter will start to absorb fewer, or even consistently reject, corrections offered by subsequent updates against higher accuracy maps.

To reduce the effects of inconsistency, in this work, we model each global-to-map-transformation state as a random process driven by white noise, *i.e.*,

$$\dot{\tau}_i = \eta_i^c \tag{6}$$

where η_i^c is zero-mean white Gaussian noise. The equivalent discrete-time state propagation equation is

$$\tau_i^{k+1} = \tau_i^k + \eta_i \tag{7}$$

where τ_i^{k+1} and τ_i^k are the global-to-map transformations of map i at steps $k+1$ and k, respectively, and η_i is the discrete-time zero-mean Gaussian noise. The effect of this model is to increase over time the filter's uncertainty about the location of a map, thus reducing its overconfidence about the positions of the features comprising it. Note that if a map is not reobserved for a long time, this model will lead to a very large uncertainty about its transformation w.r.t. the filter's global. In such an event, we will consider the map as "lost" and re-initialize it only when re-discovered.

2) Inflated Camera Measurement Noise Model: Once the *Relocalizer* determines the 2D-to-3D feature correspondences to map i, these are provided to the filter (see Fig. 2), where an approach similar to that of [30], [6] is employed to process the following measurement:

$$\mathbf{z}_{ij} = \pi \left[{}^{C_k}_{G_1}\mathbf{C} \left({}^{M_i}_{G_1}\mathbf{C}({}^{G_1}\theta_{M_i}^k)\, {}^{M_i}\mathbf{f}_j + {}^{G_1}\mathbf{p}_{M_i}^k - {}^{G_1}\mathbf{p}_{C_k} \right) \right] + \nu_{ij} \tag{8}$$

where $\mathbf{z}_{ij}$ are the pixel coordinates of the image measurement to feature j of map i, ${}^{M_i}\mathbf{f}_j$ denotes the 3D position of the feature, and ν_{ij} is the zero-mean Gaussian noise of the measurement with covariance $\sigma^2\mathbf{I}_2$.

Specifically, the 2D-to-3D matches are processed by linearizing (8), around the camera pose $\mathbf{x}_{C_k}$ and global-to-map τ_i^k estimates, while treating the feature position ${}^{M_i}\mathbf{f}_j$ as perfectly known. To mitigate the effects of inconsistency arising from this approximation, we employ two more noise-weighting factors in addition to the measurement noise covariance $\sigma^2\mathbf{I}_2$. Specifically, we inflate the noise first as dictated by the feature's approximate covariance, $\mathbf{P}_{\mathbf{f}_j\mathbf{f}_j}$, and then according to the filter's covariance, $\mathbf{P}_{\mathbf{xx}}$ (originally adopted by NASA's Apollo program [41]). Summing these terms with the pixel noise yields the following measurement covariance:

$$\mathbf{R}_{ij} = \alpha \mathbf{H}_\mathbf{f} \mathbf{P}_{\mathbf{f}_j\mathbf{f}_j} \mathbf{H}_\mathbf{f}^T + \beta \mathbf{H}_\mathbf{x} \mathbf{P}_{\mathbf{xx}} \mathbf{H}_\mathbf{x}^T + \sigma^2 \mathbf{I}_2 \tag{9}$$

where $\mathbf{H}_\mathbf{f}$, $\mathbf{H}_\mathbf{x}$ are the Jacobians of the measurement function (8) with respect to the feature position and the camera pose, respectively, and α and β are tuning parameters (in this work we have selected $\alpha = 1.0$ and $\beta = 0.5$, which in our experiments produced better initial localization accuracy when two users look at the same scene).

3) Computational Considerations: Note that by including the global-to-maps transformations $\mathbf{x}_\tau^k$ in the filter's state vector, instead of the *Backend*, we trade a temporary loss of accuracy, for responsiveness and processing savings. That is, by not estimating the 4 DOF transformations in the *Backend*, we avoid expensive matching between the features of all the maps, thus resulting in significant processing savings. Instead, the transformation between a map and a device's global frame is determined *immediately* after the first successful filter update against that map. This estimate is then improved as more measurements become available.

The computational cost of the filter is cubic in the size of the state vector and thus in the number of users, N. We typically, however, consider a small number of users ($N = 2, \ldots, 5$), hence this part of the state vector corresponds to a small fraction (8-20 elements) of the entire state (100-150 elements). Additionally, since the filter maintains the square root of the information matrix, the memory cost of maintaining the global-to-map states in the filter is quadratic in N. Lastly, the communication cost per user is linear in N, as each user receives the partial maps of other users and broadcasts its own map.

We finally note that in real-time AR, the *Frontend*'s accuracy and latency are significantly more important than those of the *Backend*, since the users are immediately able to observe any errors, or latency, of the *Frontend*. Hence, to compute the current pose estimate for rendering virtual objects, the AR Visualizer (see Fig. 2) extrapolates the latest filter estimate by integrating IMU measurements.

IV. Experimental Results

In what follows, we first briefly describe our experimental setup and then present the evaluation results for the proposed approach. In particular, we start by providing quantitative evidence in support of our proposed global-to-map propagation model described in Sect. III-C. Subsequently, we present the results from multiple experiments where two devices collect data inside a room with different starting pose configurations and evaluate the accuracy of their estimates against VICON ground-truth. Note that in a shared AR experience, the virtual objects are rendered in the user's current image by employing the most recent device pose estimate. Hence, the *Backend* poses, while affecting the accuracy of the *Frontend*, are not directly utilized for computing the virtual objects' poses w.r.t. the camera at the current time. Therefore, all results presented hereafter consider the *first available Frontend pose*

Dataset	$\sigma_\theta = 10^{-4} rad$ $\sigma_p = 10^{-3} m$	$\sigma_\theta = 10^{-5} rad$ $\sigma_p = 10^{-4} m$	$\sigma_\theta = 10^{-6} rad$ $\sigma_p = 10^{-5} m$
D3	5.8	4.7	5.4
D4	7.6	7.2	8.0

TABLE I

RMSE ERRORS (CM) OF TWO DATASETS WITH DIFFERENT GLOBAL-TO-MAP PROPAGATION NOISE VALUES.

Dataset	w/o propagation	w/ propagation
D_1	**6.2**	6.3
D_2	7.7	**6.5**
D_3	4.6	**4.4**
D_4	4.6	**4.2**

TABLE II

RMSE ERRORS (CM) WITH AND WITHOUT GLOBAL-TO-MAP TRANSFORMATION PROPAGATION WHEN RELOCALIZING AGAINST THE MAP OF DATASET D_5 CREATED OFFLINE.

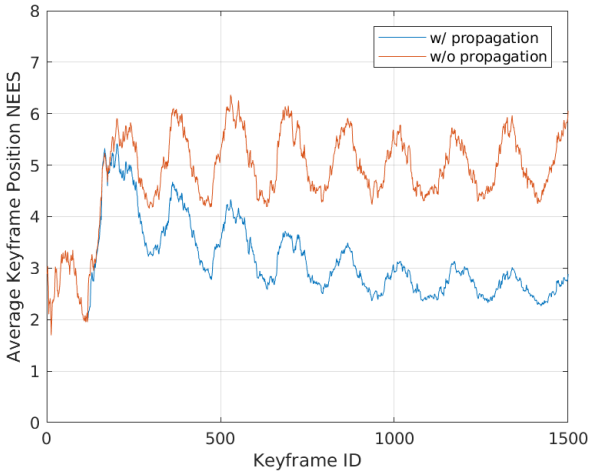

Fig. 3. Simulation results: Average keyframe position NEES.

estimate (*i.e.*, the filter's estimate for the most recent image, computed by processing IMU, visual, and relocalization measurements), and not those from the *Backend* after refinement.

All six visual and inertial datasets (*i.e.*, $D_1 - D_6$) were collected and evaluated using two Google Pixel phones, where greyscale images of resolution 640×480 were saved at 30 Hz, along with consumer-grade, MEMS-based, IMU data at 200 Hz. The collected datasets are between five to seven minutes long and correspond to 100-200 m trajectories inside a room. Specifically, the datasets $D_1 - D_3$ and $D_5 - D_6$ are collected by starting at different locations and moving inside the room so as to visit most areas multiple times with nominal (D_1, $D_3 - D_6$) or fast (D_2) motion profiles. On the other hand, D_4 is collected by focusing on parts of the room that are typically not covered by the other datasets.

A. Impact of Global-to-Map Transformation Propagation

We start by first assessing the impact of the global-to-map transformation noise standard deviation (std) on the position root mean square error (RMSE) of our system on datasets D_3 and D_4. As evident from Table I, the RMSE is fairly insensitive to σ_θ (yaw noise std) and σ_p (position noise std) over a wide range of values.

Next, we evaluate the normalized estimation error squared (NEES) in Monte-Carlo simulations consisting of 30 trials, where the device moves on a circle of radius 5 m and observes 40 features. Fig. 3 depicts the *Frontend*'s keyframe position NEES with and without global-to-map transformation propagation [see (7)]. As evident, the global-to-map transformation propagation reduces the NEES, and hence, improves the estimator's consistency.

Lastly, we evaluate the effect of the proposed global-to-map transformation propagation when a complete map of the environment is available beforehand (*i.e.*, the map does *not* evolve). In particular, we evaluate the *Frontend* running on datasets D_1 through D_4 while performing map-based updates against the map of D_5 computed offline, with and without employing global-to-map transformation propagation. The results of Table II show that when the map is built beforehand, and thus has higher accuracy as compared to an incrementally built and shared one, transformation propagation yields only a slight increase in accuracy except for dataset D_1, for which the difference in RMSE errors is within the ground-truth evaluation error. On the other hand, when we evaluate our system using datasets D_1 through D_5 with a map of the environment that is incrementally built (*i.e.*, evolving map), the global-to-map transformation propagation significantly improves accuracy (see Table III).

These findings support our hypothesis that a significant additional source of inconsistency in map-based updates is due to employing different feature position estimates across time while treating the map as perfect. As our results demonstrate, this can be considerably mitigated by employing the proposed global-to-map transformation propagation.

Dataset	w/o propagation	w/ propagation
D_1	7.2	**6.9**
D_2	16.7	**14.8**
D_3	6.2	**4.7**
D_4	8.4	**7.2**
D_5	6.6	**6.0**

TABLE III

RMSE ERRORS (CM) WITH AND WITHOUT GLOBAL-TO-MAP TRANSFORMATION PROPAGATION WHEN RELOCALIZING AGAINST THE INCREMENTAL MAP OF THE SAME DATASET.

B. Decentralized SLAM Evaluation

We hereafter evaluate our proposed algorithm for the case of two users using combinations of datasets $D_1 - D_6$ (*i.e.*, 15 dataset pairs in total) on two mobile devices. Note that alternative approaches such as [15], [23] could be employed for solving the multi-device localization problem. These methods, however, focus on finding the optimal trajectories (and maps) of all devices, thus they are computationally expensive and not suitable for mobile devices. For this reason, we do not include them in our evaluations.

As mentioned earlier, a key requirement for shared AR is that each user can localize accurately w.r.t. the other user's map. To evaluate this, in Fig. 4 we present the box-and-whisker plot of the RMSE w.r.t. the user's own map (host) and the RMSE w.r.t. the other user's map (others). Specifically, consider the following two position estimates w.r.t. $\{M_1\}$ (*i.e.*, the host map) and $\{M_2\}$ (*i.e.*, the other user's map) for user i at time-step k:

$$ {}^{M_1}\mathbf{p}_{C_k}^{(i)} = {}^{M_1}_{G_i}\mathbf{C}\left({}^{M_1}\theta_{G_i}^{k}\right){}^{G_i}\mathbf{p}_{C_k}^{(i)} + {}^{M_1}\mathbf{p}_{G_i}^{k} \tag{10} $$

$$ {}^{M_2}\mathbf{p}_{C_k}^{(i)} = {}^{M_2}_{G_i}\mathbf{C}\left({}^{M_2}\theta_{G_i}^{k}\right){}^{G_i}\mathbf{p}_{C_k}^{(i)} + {}^{M_2}\mathbf{p}_{G_i}^{k} \tag{11} $$

where ${}^{G_i}\mathbf{p}_{C_k}^{(i)}$ is the *Frontend* estimated position of user i at time step k w.r.t. its own global frame, ${}^{M_j}\theta_{G_i}^{k}$ and ${}^{M_j}\mathbf{p}_{G_i}^{k}$ are the yaw and position, respectively, of the global-to-map transformation of user i w.r.t. map $\{M_j\}$, and ${}^{M_1}\mathbf{p}_{C_k}^{(i)}$ and ${}^{M_2}\mathbf{p}_{C_k}^{(i)}$ are the positions of user i w.r.t. $\{M_1\}$ and $\{M_2\}$, respectively. The RMSE of each user's trajectory w.r.t. each map [see (10), (11)] assesses the ability of the system to accurately render virtual content placed by either user.[5] Additionally, Fig. 5 shows the number of feature measurements involved in the map-based updates against the host and the other user's maps.

First, we note that the accuracy of the *Frontend* depends on the motion difficulty, as fast motions result in short feature tracks and inaccurate maps (*e.g.*, see dataset D_2 in Fig. 4 and Fig. 5). The errors in the global-to-map transformations also depend on the number of map-based updates as well as the map's accuracy. Specifically, the updates are required not only to compute a reliable global-to-map estimate, but also to compensate for the pose drift. Hence, in general, small number of map-based updates results in lower accuracy (*e.g.*, see dataset D_1 in Fig. 4). Furthermore, there are cases where the RMSE of a user against the other user's map is lower than that of the host map, this is due to the fact that the other user has produced a higher quality map as compared to the host at the time the map-based updates are applied.[6] As expected, these findings imply that through a decentralized localization and mapping algorithm as the one described in this work, a precise map will improve the localization accuracy of all users viewing it.

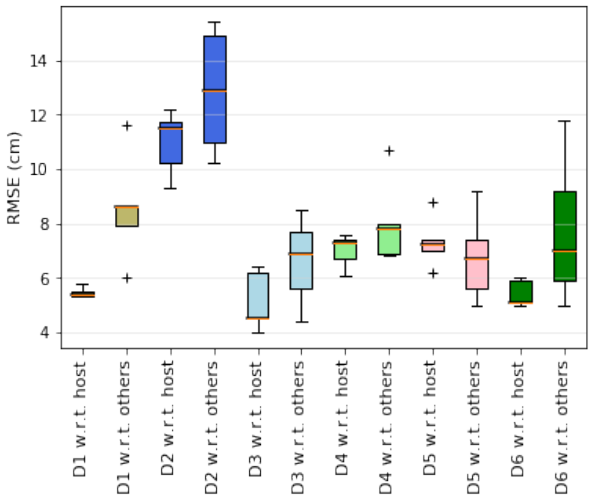

Fig. 4. RMSE of each dataset w.r.t. its own map (host map), and the other user's map frame for all dataset pairs.

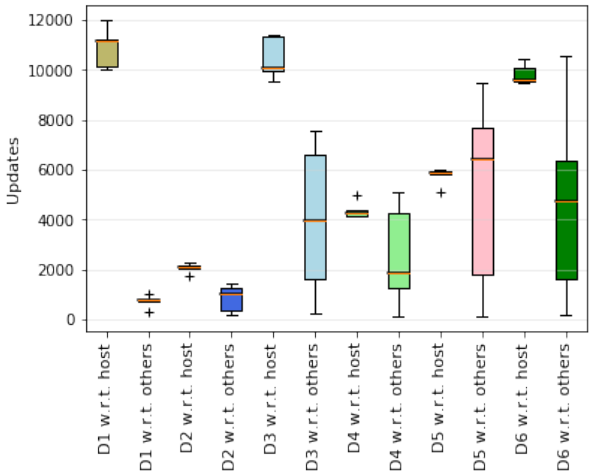

Fig. 5. The number of map-based updates from the user's own map (host map) and the other user's map for all dataset pairs.

V. Conclusion and Future Work

In this paper, we presented an approach for enabling two or more users to perform shared augmented reality without the need for external servers or cloud infrastructure. Specifically, each user broadcasts their incrementally-built map as soon as it becomes available, and continuously communicates improved feature position estimates. Additionally, and through map-based updates, the 4-DOF relative transformation between the users' global and map frames are determined, which allows computing the pose of all virtual objects in all users' frames. Furthermore, and in order to compensate for the additional inconsistency caused by the map-based updates against features whose position estimates change with time, we model the global-to-map transformations as random processes driven by noise. Finally, we experimentally verified the gains in accuracy due to employing this model and demonstrated the performance of our approach in the case of two users operating in room-size areas.

As part of our future work, we will investigate strategies for adjusting the transformation's propagation uncertainty based on the user's and map's estimated accuracy. Additionally, we will focus on feature selection methods so as to reduce the communication and processing cost, thus enabling long-term mapping on extremely constrained platforms, such as wearables or embedded devices.

[5]Note that in addition to RMSE, other factors such as jitter play an important role in the quality of the AR experience. These issues, however, are related to the rendering of the virtual objects and can be compensated for by employing *e.g.*, low-pass filters and patch trackers, in addition to using the latest camera pose from the *Frontend*, to reduce the reprojection errors. Addressing these considerations is outside the scope of this work.

[6]Due to space limitations, specific values for all trials' RMSE and number of map-based updates are included in [42].

REFERENCES

[1] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proc. of the IEEE International Conference on Robotics and Automation*, Rome, Italy, Apr. 10–14 2007, pp. 3482–3489.

[2] K. J. Wu, A. M. Ahmed, G. A. Georgiou, and S. I. Roumeliotis, "A square root inverse filter for efficient vision-aided inertial navigation on mobile devices," in *Proc. of Robotics: Science and Systems*, Rome, Italy, July 12–16 2015.

[3] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial odometry using nonlinear optimization," *International Journal of Robotics Research*, vol. 34, no. 6, pp. 314–334, Dec. 2015.

[4] H. Liu, M. Chen, G. Zhang, H. Bao, and Y. Bao, "ICE-BA: Incremental, consistent and efficient bundle adjustment for visual-inertial SLAM," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, June 18–22 2018, pp. 1974–1982.

[5] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, Aug 2018.

[6] S. Lynen, T. Sattler, M. Bosse, J. Hesch, M. Pollefeys, and R. Siegwart, "Get out of my lab: Large-scale, real-time visual-inertial localization," in *Proc. of Robotics: Science and Systems*, Rome, Italy, July 12–16 2015.

[7] H. Oleynikova, M. Burri, S. Lynen, and R. Siegwart, "Real-time visual-inertial localization for aerial and ground robots," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hamburg, Germany, Sept. 28 – Oct. 2 2015, pp. 3079–3085.

[8] Apple, "ARKit," https://developer.apple.com/arkit.

[9] Microsoft, "HoloLens," https://www.microsoft.com/en-us/hololens.

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

[11] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, "Multiple relative pose graphs for robust cooperative mapping," in *Proc. of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, May 3–8 2010, pp. 3185–3192.

[12] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, "Collaborative monocular SLAM with multiple micro aerial vehicles," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, Nov. 3–7 2013, pp. 3963–3970.

[13] L. Riazuelo, J. Civera, and J. M. Montiel, "C2TAM: A cloud framework for cooperative tracking and mapping," *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401–413, 2014.

[14] J. G. Morrison, D. Gálvez-López, and G. Sibley, "MOARSLAM: Multiple operator augmented RSLAM," in *Distributed Autonomous Robotic Systems*. Springer Japan, 2016, vol. 112, pp. 119–132.

[15] C. X. Guo, K. Sartipi, R. C. DuToit, G. A. Georgiou, R. Li, J. O'Leary, E. D. Nerurkar, J. A. Hesch, and S. I. Roumeliotis, "Resource-aware large-scale cooperative three-dimensional mapping using multiple mobile devices," *IEEE Transactions on Robotics*, vol. 34, no. 5, pp. 1349–1369, Oct. 2018.

[16] M. Karrer, P. Schmuck, and M. Chli, "CVI-SLAM-collaborative visual-inertial SLAM," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2762–2769, Oct. 2018.

[17] A. Cunningham, M. Paluri, and F. Dellaert, "DDF-SAM: Fully distributed SLAM using constrained factor graphs," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, Oct. 18–22 2010, pp. 3025–3030.

[18] A. Cunningham, V. Indelman, and F. Dellaert, "DDF-SAM 2.0: Consistent distributed smoothing and mapping," in *Proc. of the IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, May 6–10 2013, pp. 5220–5227.

[19] Y. Bar-Shalom, X. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation: Theory, Algorithm, and Software*. New York, NY: John Wiley and Sons, 2004.

[20] L. Paull, G. Huang, M. Seto, and J. J. Leonard, "Communication-constrained multi-AUV cooperative SLAM," in *Proc. of the IEEE International Conference on Robotics and Automation*, Seattle, Washington, May 26–30 2015, pp. 509–516.

[21] E. D. Nerurkar, S. I. Roumeliotis, and A. Martinelli, "Distributed maximum a posteriori estimation for multi-robot cooperative localization," in *Proc. of the IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 12–17 2009, pp. 1402–1409.

[22] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1286–1311, 2017.

[23] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-efficient decentralized visual SLAM," in *Proc. of the IEEE International Conference on Robotics and Automation*, Brisbane, Australia, May 21–25 2018, pp. 2466–2473.

[24] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nara, Japan, Nov. 13–16 2007, pp. 225–234.

[25] M. J. Schuster, C. Brand, H. Hirschmüller, M. Suppa, and M. Beetz, "Multi-robot 6D graph SLAM connecting decoupled local reference filters," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hamburg, Germany, Sept. 28 – Oct. 2 2015, pp. 5093–5100.

[26] H. Zhang, X. Chen, H. Lu, and J. Xiao, "Distributed and collaborative monocular simultaneous localization and mapping for multi-robot systems in large-scale environments," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, June 2018.

[27] R. Mur-Artal, J. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Trans. on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[28] R. Egodagamage and M. Tuceryan, "Distributed monocular visual SLAM as a basis for a collaborative augmented reality framework," *Computers & Graphics*, vol. 71, pp. 113–123, 2018.

[29] K. Sartipi and S. I. Roumeliotis, "Efficient alignment of visual-inertial maps," in *Proc. of International Symposium on Experimental Robotics*, Buenos Aires, Argentina, Nov. 5–8 2018.

[30] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. E. Johson, A. Ansar, and L. Matthies, "Vision-aided inertial navigation for spacecraft entry, descent, and landing," *IEEE Trans. on Robotics*, vol. 25, no. 2, pp. 264–280, Apr. 2009.

[31] R. C. Dutoit, J. A. Hesch, E. D. Nerurkar, and S. I. Roumeliotis, "Consistent map-based 3D localization on mobile devices," in *Proc. of the IEEE International Conference on Robotics and Automation*, Singapore, Singapore, May 29 – June 3 2017, pp. 6253–6260.

[32] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, New York, NY, June 17 – 22 2006, pp. 2161–2168.

[33] O. Naroditsky, X. S. Zhou, S. I. Roumeliotis, and K. Daniilidis, "Two efficient solutions for visual odometry using directional correspondence," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 34, no. 4, pp. 818–824, Apr. 2012.

[34] A. B. Chatfield, *Fundamentals of High Accuracy Inertial Navigation*. American Institute of Aeronautics and Astronautics, 1997, vol. 174.

[35] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proc. of the International Conference on Computer Vision Theory and Applications*, Lisboa, Portugal, Feb. 5–8 2009, pp. 331–340.

[36] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[37] T. Ke and S. I. Roumeliotis, "An efficient algebraic solution to the perspective-three-point problem," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, July 22 – 25 2017, pp. 4618–4626.

[38] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An accurate O(n) solution to the PnP problem," *International Journal of Computer Vision*, vol. 81, no. 2, p. 155, July 2008.

[39] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proc. of the European Conference on Computer Vision*. Graz, Austria: Springer, May7 – 13 2006, pp. 430–443.

[40] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. of the IEEE International Conference on Computer Vision*, Barcelona, Spain, Nov. 6–13 2011, pp. 2564–2571.

[41] D. S. Bayard and P. B. Brugarolas, "An estimation algorithm for vision-based exploration of small bodies in space," in *Proc. of the IEEE American Control Conference*, Portland, OR, June 8–10 2005, pp. 4589–4595.

[42] K. Sartipi, R. C. Dutoit, C. B. Cobar, and S. I. Roumeliotis, "Decentralized visual-inertial localization and mapping on mobile devices for augmented reality," Google, Tech. Rep., Aug. 2019. [Online]. Available: http://mars.cs.umn.edu/tr/decentralizedvi19.pdf