UbiPose: Towards Ubiquitous Outdoor AR Pose Tracking using Aerial Meshes

Weiwu Pang
weiwupan@usc.edu
University of Southern California

Chunyu Xia
chunyuxi@usc.edu
University of Southern California

Branden Leong
branden@usc.edu
University of Southern California

Fawad Ahmad
fawad@cs.rit.edu
Rochester Institute of Technology

Jeongyeup Paek
jpaek@cau.ac.kr
Chung-Ang University

Ramesh Govindan
ramesh@usc.edu
University of Southern California

ABSTRACT

Tracking the position and orientation, or pose, of a viewing device enables AR applications to accurately embed virtual content in physical spaces. Mobile OSs track pose by matching device camera images against street-level imagery. Thus, pose tracking is often unavailable at off-street pedestrian locations. UbiPose enables pose tracking at such locations using aerial meshes, generated from satellite imagery, that are likely to be more widely available at these locations. However, matching a camera image against an aerial mesh can be error-prone, even with modern neural matchers. These neural components are also compute-intensive. UbiPose contains a novel pose tracking pipeline that runs entirely on a mobile device using fast-path optimizations designed to accept or reject pose estimates in many cases, without sacrificing accuracy. Experiments on real-world traces show that it achieves tracking accuracy comparable to AR pose tracking in iOS in places where that is available, and is able to track pose accurately in places where it is not.

CCS CONCEPTS

• **Human-centered computing** → **Mixed / augmented reality**; **Ubiquitous and mobile computing systems and tools**;

KEYWORDS

Outdoor Pose Estimation, Augmented Reality, Aerial Mesh

This material is based upon work supported by the National Science Foundation under Grant No. 1956445.

ACM MobiCom '23, October 2–6, 2023, Madrid, Spain

ACM ISBN 978-1-4503-9990-6/23/10.
https://doi.org/10.1145/3570361.3613263

ACM Reference Format:
Weiwu Pang, Chunyu Xia, Branden Leong, Fawad Ahmad, Jeongyeup Paek, and Ramesh Govindan. 2023. UbiPose: Towards Ubiquitous Outdoor AR Pose Tracking using Aerial Meshes . In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3570361.3613263

1 INTRODUCTION

Augmented reality (AR) applications place virtual content in a user's view of the physical world. These applications can revolutionize the way we interact with the physical world. They can deliver contextual cues that enrich our experience of physical spaces, such as museums, monuments and other historical sites. They can also promote safety during walking or driving by alerting us to impending dangers beyond our visual field of view. Motivated by this potential, major mobile OSs, iOS and Android, contain mature AR application development platforms, ARKit [9] and ARCore [36] (respectively).

The central primitive that enables AR is the ability to track viewer *pose* – the position and orientation of the viewing device (a smartphone camera, or a headset), expressed in world coordinates. With accurate pose, applications can correctly align virtual content in the physical space captured within the viewer. Inaccurate alignment can adversely impact the user experience. In this paper, we focus on *outdoor* pose tracking, which both ARKit and ARCore support.

Both have built-in modules that provide continuous *AR pose tracking*. At a high-level these modules contain two components [8, 36]: a *localizer* that estimates the current pose of a view devices, and, because localization can be expensive, a *tracker* that tracks viewer movements over short timescales to update viewer pose in between localizer invocations. ARKit and ARCore use *visual-inertial odometry* (VIO) trackers, which fuse camera images with inertial sensor readings, and *visual localizers* that determine pose of a viewer's camera based on a corpus of pre-collected images.

The visual localizers in ARKit and ARCore use *terrestrial* (or surface-level) imagery. This kind of imagery powers Google Street View and Bing Streetside. At scale, cameras attached to specially-outfitted vehicles periodically collect this imagery by sweeping a large parts of the planet. As such, visual localizers in these AR platforms are likely to be unavailable in locations where at-scale imagery collection is difficult: pedestrian areas in corporate or university campuses, outdoor shopping areas, apartment complexes, and so on (§2).

To address this shortcoming, we explore the design and implementation of UbiPose, an *on-device* AR pose tracker that uses *aerial* imagery collected by earth observation satellites or aircraft (§3). This imagery, pre-processed into a 3D *aerial mesh* is widely available in, for example, Google Earth [38]. Using aerial meshes can increase the availability of AR pose tracking to areas where terrestrial imagery is unavailable. By one estimate [48], Google Earth has 3× the coverage of Google Street View. To our knowledge, no prior work has explored AR pose tracking using aerial meshes (§6).

Challenges and Contributions. Visual localization using aerial meshes presents two challenges. The first challenge is *ensuring accuracy* of the estimated pose. Visual localizers match visual features in a device camera image to visual features in collected imagery [52, 69]. Knowing the 3D positions of features in visual imagery (obtained offline), a localizer can estimate camera pose using the matched features. An aerial mesh is a compact 3D representation of the physical world generated from aerial 2D imagery. As such, it lacks some visual detail, so matching features in a 2D image to features in a 3D mesh can result in significant pose errors (§3.2).

Recent work [67] has shown that modern neural feature extractors and matchers can localize a camera image by matching it to images *rendered* from a *terrestrial* mesh. Aerial meshes are of poorer quality than terrestrial meshes, so UbiPose cannot directly use this approach (§3.2). Matching against features in the entire rendered corpus can result in false positives, leading to poor pose estimates at the tail (*e.g.,* 95th percentile).

To address this, our first contribution is the design of a *VIO-assisted localizer* (§3.3). Instead of pre-rendering images and extracting features offline, UbiPose renders multiple images (for robustness to VIO errors) *at the location determined by VIO*, then extracts features from the rendered images and uses these to match and localize. This significantly improves tail localization accuracy by scoping the feature search.

However, this implies that UbiPose must render, extract, match, and localize on the mobile device. Especially with heavyweight neural extractors and matchers [24, 74], this can easily overwhelm mobile device compute resources.

Our second contribution is the design of optimizations that reduce UbiPose's resource footprint (§3.4). Two optimizations, *Lift-and-Project* and *Early-Exit* respectively allow fast-path acceptance and rejection of pose estimates by reusing successful matches from recent camera frames. A third, *Fused-Match* projects features from multiple rendered images into one to reduce the number of invocations to the matcher when full-path processing is necessary.

Summary of Results. On traces collected in three major metropolitan areas in North America, UbiPose's accuracy is comparable to ARKit's (§4). In many of these locations, ARKit is unavailable: there, UbiPose is able to obtain pose estimates with the same accuracy (about 1 m positioning error and 1° orientation error at the tail) as in locations where ARKit is available. This is encouraging, and suggests that aerial-mesh based pose tracking can increase pose tracking coverage. UbiPose runs on modern mobile device hardware with modest resource footprints and its optimizations reduce latency by 2× and power consumption by 20%.

2 BACKGROUND AND MOTIVATION

We first introduce background and terminology, then describe shortcomings in the state-of-practice in pose tracking.

Pose. This term refers to the position and orientation of an object along six degrees of freedom (6DoF) – three translational (x, y, z) and three rotational (roll, pitch, yaw) axes [28]. Robotics, autonomous driving, and mixed reality applications need to estimate the precise pose of objects for motion planning or for rendering virtual objects in a scene.

Augmented Reality (AR). AR applications place virtual objects or markers in the physical world, as viewed through a device such as a smartphone or a headset. *Indoor* AR applications can enrich the viewing experience in museums, aid navigation in unfamiliar spaces, or enhance shopping in malls and stores [3, 23, 34, 44, 61, 70, 83]. *Outdoor* AR can enhance visitor experience at landmarks, aid pedestrian navigation, or gamify exercise [47, 64, 80, 85, 92].

As an example outdoor AR, at WWDC 2020 Apple first demonstrated an AR art installation in San Francisco (Fig. 1). This installation used AR to create an immersive experience that inserted virtual art into the physical environment. Users could use their iPhones to view the installation and to interact with the virtual art piece. The ability to precisely position and orient the virtual art was crucial to the success of this demonstration. Without the proper alignment of virtual objects relative to the physical environment as seen by the viewing device (the iPhone in this case), users would have had a jarring and disjointed experience.

AR Pose Tracking. To accurately position a virtual object in the physical world, AR applications need a precise estimate of the pose of the viewing device (*viewer pose*) in the

Figure 1: Virtual art rendered using Apple ARKit, from [4].

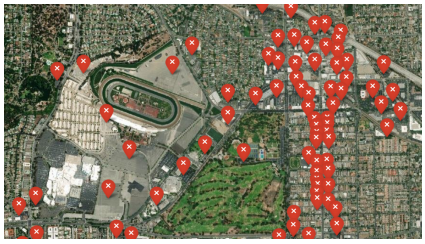

Figure 2: ARKit unavailability in the downtown area of Arcadia, CA, USA.

Figure 3: ARKit unavailability in Pasadena, CA, USA.

global coordinate system. Since the viewer can move through the physical space, AR applications must *track* viewer pose continuously and accurately to enable a smooth viewing experience. We call this the *AR pose tracking* problem.

In this paper, we focus on *outdoor* AR pose tracking. Of particular interest to us is the *ubiquity* of outdoor AR pose tracking. Ubiquitous pose tracking: (a) works on commodity mobile devices without assuming additional sensors or other specialized equipment, and (b) can track pose well in a wide range of outdoor physical spaces.

The State-of-practice in Outdoor AR Pose Tracking. While much research has explored outdoor AR pose tracking (§6), the state of practice in ubiquitous pose tracking is represented by Apple's ARKit [8] and Google's ARCore [36]. Apple used ARKit to develop the virtual art installation described above [4], and we use that as an example to explain how AR pose tracking enables such applications.

(1) In ARKit, the art installation app developer must provide a 3D model of the art piece, and a *geo-anchor* (the position in world coordinates at which to place the art piece).
(2) When a user opens the app on their iPhone, the app invokes ARKit and instantiates an AR session. ARKit then downloads a map for localization that contains visual features corresponding to the surrounding environment, together with feature positions in world coordinates.
(3) As the user moves the camera, ARKit captures camera images, matches them with features in the map, and uses these to estimate the pose of the camera in world coordinates. Using the pose, and a model for the camera and its parameters, ARKit can place the virtual art piece in the corresponding pixel locations on the iPhone display.

ARKit uses *street-level terrestrial imagery* to obtain features of the environment [8].

Gaps in the State-of-practice. To understand the accuracy and ubiquity of outdoor AR pose tracking, we evaluated ARKit in three metropolitan areas (§4).

Accuracy. ARKit achieves 0.5-1.1 m median position error, 0.9-2 m 95th-percentile (*p95*) position error, 0.5-1.2° median position error, and 1.1-2.3° p95 rotation error. To ensure good user experience, it is necessary to have low tail error in addition to having low median error, so we consider p95 errors throughout our work. In general, then, ARKit is remarkably accurate even at the tail.

Ubiquity. While ARKit's GeoTracking works well in many places, it is limited to specific areas. It appears to use terrestrial imagery [4] collected for the Look Around [5] feature in Apple's Maps. This is similar to Google Maps' Street View [43], or Bing Maps' Streetside [56]. At scale, these companies capture most[1] of this kind of imagery using vehicles driving on public streets or other areas accessible to vehicles. This means that ARKit pose tracking is unlikely to be available in pedestrian only areas: outdoor malls, parts of college and corporate campuses, large apartment complexes, vehicle-restricted urban centers, amusement parks, and so on.

Apple's Maps app supports the ability to test for ARKit availability at a given location, by setting a geo-anchor at that location on a map. This allows us to check unavailability by *virtually visiting* the location. To demonstrate how widespread ARKit unavailability can be, we virtually visited offstreet locations (parking lots, side streets) within about 20-30 blocks in the downtown areas of two cities in North America (locations omitted for anonymity). In each block, we tested one location for availability. Figs. 2 and 3 shows that ARKit unavailability can be pervasive in these areas.

Android ARCore's pose tracking is also unavailable in some locations. ARCore exports an interface to test for availability of its visual positioning system (VPS), but in our experience, this interface is unreliable: it reports availability even in areas that we verified were unavailable. Instead, we physically visited four qualitatively different locations to determine ARCore availability. Tbl. 1 reports position and heading errors at these locations as reported by ARCore, as well as the corresponding errors at a street-side location near these locations. In each of these cases, errors at these locations are up to 10× greater than at the street-side — the latter has accuracy consistent with ARKit (§4). At these locations, ARCore appears to be estimating pose from GPS and its compass

[1]Some of this imagery is user-contributed [43]; such imagery is unlikely to be dense enough to permit accurate AR pose tracking.

Sample locations	**At the location**	**Nearest street-side**
Housing Complex	9.31m, 23.9°	0.77m, 1.7°
University Campus	8.92m, 11°	0.85m, 1.4°
Church	6.47m, 24.7°	0.80m, 1.5°
High School	3.76m, 3.7°	1.45m, 2.4°

Table 1: ARCore reported position (in m) and heading (in °) errors at the location and on the nearest street.

because imagery is unavailable for visual positioning. GPS is ubiquitous, but its position error ($\leq$ 8m 95% horizontal error) [31] is an order of magnitude larger than ARKit's.

Towards Ubiquitous AR Pose Tracking. This analysis suggests that outdoor AR pose tracking is not as ubiquitous yet as needed for widespread use of AR applications, both on iOS and Android. In this paper, we explore techniques to significantly increase the coverage of AR pose tracking beyond locations where street-level imagery is available.

3 UBIPOSE DESIGN

We now describe the design and implementation of UbiPose.

3.1 Increasing Availability of Pose Tracking

Goal. Truly ubiquitous AR pose tracking would mean the availability of AR pose tracking at any location on the planet accessible to pedestrians or vehicles, public or private. UbiPose takes a step towards this goal by expanding the coverage of AR pose tracking beyond areas where street-level imagery is available. §7 discusses what it would take, beyond UbiPose, to achieve true ubiquity.

Requirements. To this end, we impose three requirements:

(1) Where street-level imagery is available, UbiPose should achieve pose tracking accuracy comparable to ARKit. This would allow mobile apps to track pose without relying on street-level imagery.
(2) Where street-level imagery is *not* available, UbiPose should still be able to achieve pose tracking accuracy comparable to that achieved by ARKit in areas where street-level imagery *is* available. This ensures that AR apps can have a uniform expectation of accuracy, regardless of where the user uses the app.
(3) UbiPose should run entirely on the mobile devices with a modest resource footprint. ARKit, after downloading its map, runs entirely on the mobile device, so on-device operation for UbiPose can lower the barrier to adoption.

Key Insight. Instead of relying on street-level imagery, UbiPose achieves AR pose tracking using *aerial meshes*.

A mesh is a compact representation of a three-dimensional object – it approximates object surfaces using a mesh of polygons. The size of the polygons (most meshes use triangles or quadrangles) represents a trade-off between accuracy and compactness: smaller polygons result in a high resolution, more accurate, mesh but the resulting mesh can be larger in size than those using larger polygons. Beyond capturing the surface geometry of an object using polygons, mesh representations usually contain associated surface color and texture attributes. As such, meshes are often used to compactly store, and accurately render, 3D objects, for a wide variety of uses.

Figure 4: An screenshot of an aerial mesh from Google Earth.

To generate a mesh, one can use sensors ranging from 2D cameras to 3D sensors such as stereo or RGB-D cameras and LiDARs. Today, most mesh generation at scale relies on 2D images. *Photogrammetry* [76] can generate a mesh representation of a (static) 3D object from a sequence of 2D images. Color mapping and texturing algorithms automatically project color and texture from images to the mesh surfaces.

An *aerial mesh* is a mesh of an outdoor space captured from aerial 2D imagery, obtained either using earth-observation satellites or specially equipped aircraft. (In contrast, a *terrestrial mesh* is captured using surface-level imagery). Fig. 4 shows a picture of an aerial mesh obtained from Google Earth [38]. As such, aerial meshes can capture the structure of locations inaccessible to at-scale methods for capturing surface-level imagery, such as vehicles used to capture Google Street Views [43]. Examples of such locations include off-street plazas and malls, as well as privately-owned residential, corporate, and industrial complexes. By one estimate, aerial meshes from Google Earth cover up to 36 million square miles [48] (3$\times$ more than Street View) of the Earth's surface, or 98% of the human-inhabited regions of the Earth.

Challenges. While aerial meshes potentially provide greater coverage, we know of no work that has demonstrated accurate AR pose tracking using these. So, UbiPose must address two important challenges: (a) how to track AR pose using aerial meshes (§3.2) and (b) how to do so accurately (§3.3), and with minimal resources on mobile devices (§3.4).

3.2 Aerial Mesh Based Pose-Tracking

UbiPose's approach to pose-tracking using an aerial mesh is qualitatively different from that of prior work. Pose-tracking requires a fundamental primitive, image *localization* (estimating the pose of a given image), for which prior work has considered two broad approaches (Fig. 5).

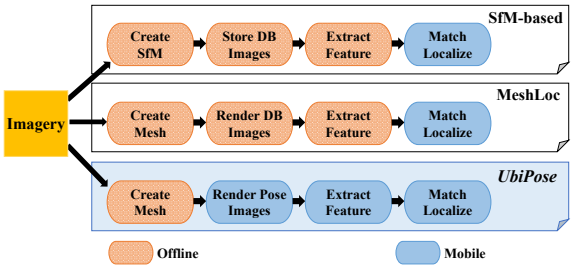

Figure 5: Different approaches to AR pose tracking at scale.

Figure 6: Cross-modality matching between a camera image and an image from an aerial mesh results in relatively few matches.

Figure 7: An image rendered from a terrestrial mesh (left) and from an aerial mesh (right). Notice the distorted arch on the right side of the aerial mesh image.

Imagery-based Localization. Both ARKit and ARCore track pose using terrestrial imagery. While we do not know the details of their approach, other recent work has described a commonly used approach for large-scale localization [52, 69] that relies on *Structure-from-Motion* (SfM).

We begin by briefly describing SfM; [33] has more detail. In its simplest form, SfM, given a sequence of images from a camera, finds matching features between each camera pair, and uses this to estimate the 3D positions of each feature point in each image. These feature points form a localization *map*. Then, given a query image, imagery-based localization matches features in the image with features on the map, and uses the 3D positions of those feature points to localize the position of the query image.

Practical systems construct these SfM maps offline [52, 69], but because SfM is compute-intensive, they scale map generation by tiling space and constructing SfM maps for each tile separately. They perform localization either on the mobile device, or offload localization to the cloud.

SfM from Aerial Mesh. As discussed in §3.1, UbiPose exploits the broad availability of aerial meshes to enable ubiquitous pose tracking. We designed a plausible approach for UbiPose that builds upon SfM, consisting of the following steps (top panel in Fig. 5): (a) Render images from different viewpoints on the aerial mesh; (b) Use these rendered images to obtain an SfM model; (c) Use the SfM model to estimate the camera pose for AR pose tracking by matching camera image features to those in the SfM map.

Unfortunately, this strawman does not perform well (§4): it has high median and p95 error, for three reasons. First, the quality of the SfM map depends on the positions on the mesh of the rendered images in step 1. Second, SfM does not exploit the fact that pixels in the rendered images already have associated 3D points (obtained from the aerial mesh); instead it infers their 3D positions, and these estimates are likely to increase error. Finally, and perhaps most important, images from a camera are different from images rendered by a mesh, since the latter represent a fundamentally different *modality*; sometimes, feature matching can find very few matches between a camera image and a rendered image (Fig. 6).

Mesh-Based Localization. MeshLoc [67] is a recent approach to localizing a camera image directly on a dense mesh. It circumvents the problem identified in Fig. 6 by leveraging the observation that modern neural local feature extractors, such as SuperPoint [24], are robust enough to match camera image features to features in rendered mesh images. At a high level, MeshLoc works as follows (middle panel in Fig. 5):

(1) Extract features offline by rendering images from the mesh. Obtain each feature's 3D position from the mesh.
(2) Given a query image, match that image's features with those obtained from step 1, then use the 3D locations of the matched features to localize the camera image.

MeshLoc uses state-of-the-art neural feature matching [74] and achieves, on a terrestrial mesh, p95 position and orientation error of about 0.5 m and 5° respectively [67].

MeshLoc on an Aerial Mesh. We evaluated MeshLoc on an aerial mesh. While it achieves low median error, it exhibits high p95 position and orientation error (§4). Images from an aerial mesh have visibly poorer visual quality than those from a terrestrial mesh in some places (Fig. 7), resulting in fewer matches. Because of the height at which they are captured, aerial meshes have fundamentally lower resolution and can contain distortions because they view vertical surfaces at an angle (unlike terrestrial images).

UbiPose's Approach. UbiPose builds upon these two approaches, but deviates from them in a fundamental way (bottom panel of Fig. 5): rather than extract features offline, it extracts features *online* entirely on the mobile device. More precisely, using a coarse pose estimate, it extracts features

from the mesh that are *likely visible at the estimated pose*. In contrast, approaches described above (SfM, MeshLoc) must search for features to match across *all* features extracted offline. This can result in false positives, especially for aerial meshes. UbiPose's design results in more robust feature matching, because the coarse pose scopes feature generation.

UbiPose uses visual-inertial odometry (VIO) to continuously obtain coarse pose estimates. VIO systems integrate measurements from visual sensors, such as cameras, with inertial sensors, such as accelerometers and gyroscopes, to produce the pose of a camera relative to a starting or (*anchor*) pose [45]. VIO is a mature technology and modern mobile OSs have integrated highly optimized VIO capabilities in the last few years [26].

UbiPose's AR pose tracking works, at a high-level as follows. It uses, as the camera pose, the VIO estimate relative to an anchor with known world coordinate position. VIO generates estimates at the frequency of the camera (30 fps). On a longer time-scale (*e.g.,* once a second), it repeatedly invokes its *localization pipeline* to obtain a current estimate of the camera pose in world coordinates. If it is able to obtain a high quality pose estimate, it uses this as the VIO anchor for subsequent frames.

3.3 UbiPose Tracker

Alg. 1 describes UbiPose's (un-optimized) pose tracker. Invoked periodically every T seconds (T=1 in our implementation) with the camera image captured at that instant as the query image, it first (Line 2) obtains the current pose estimate using VIO (§3.2). It treats this as a coarse pose estimate of the mobile device; VIO can drift over time, so by itself it is insufficient for accurate AR pose tracking. Moreover, as described above, VIO by itself produces poses relative to an anchor; in Line 2, P_V is a global pose relative to some anchor. We describe below how we obtain anchor poses.

Rendering. Line 3 renders K images from the aerial mesh at the VIO estimated pose. We discuss the details of rendering in §3.5, but rendering is fast, requiring only 30 ms on a mobile device. UbiPose renders K images, since rendering a single image produces poor results (§4), both due to errors in VIO, and because an aerial mesh has lower resolution and is often distorted (Fig. 7) so it produces fewer matches. UbiPose could have sampled K images at slightly different angles relative to the VIO pose P_V. This strategy increases the effective field-of-view (FoV), but does not lead to increased matches, since the query image has a fixed FoV, and these extra images overlap little with the query image since their orientation is different. UbiPose renders K images slightly differently: one with the virtual camera at P_V, one by moving the virtual camera forward 1 m along the viewing direction, and one by moving it backward by 1 m along the viewing direction. Since the virtual camera resolution is fixed, this results in more detail in the rendered image and/or additional features. Even though rendering is fast, K should be as small as possible to minimize resource usage; UbiPose uses $K = 3$. This strategy, empirically, produces accurate tracking while keeping resource usage within limits.

Algorithm 1: Tracker

```
1  for each query image Q do
2  |  Compute the current pose estimate P_V using VIO;
3  |  Use P_V to render K image R_1, ..., R_k from aerial mesh;
4  |  Call localization pipeline L=Localizer (Q, [R_1,...,R_k], ∅);
5  |  if inlier ratio from L ≤ δ_L or relative error between P_L
   |   from L and P_V ≥ δ_P then
6  |  |  Accept P_V and return;
7  |  else
8  |  |  Accept P_L and return;
9  |  end
10 end
```

The Localizer. Line 5 of Alg. 1 invokes a localizer which returns a pose estimate based on the rendered images. UbiPose accepts this pose estimate based on its *quality*. One test for quality is the inlier ratio, the fraction of feature matches that correspond to the most likely pose as determined by RANSAC [30], as described below. Another is proximity to the VIO pose. When the inlier ratio is higher than a threshold δ_L and the pose estimate is within a distance threshold δ_P, UbiPose accepts the estimate, and uses this as an anchor for subsequent VIO estimates.

The localizer (Alg. 2) is novel: it extracts features online (bottom panel of Fig. 5). Line 8 invokes a feature extraction module Extract on each of the rendered images, as well as the query image. Our current implementation uses SuperPoint [24], a robust neural feature extractor. Line 9 matches features between the query image and each rendered image using a feature matching module Match. Our current implementation uses SuperGlue [74], another neural model for feature matching. UbiPose can be easily extended to use other neural network models for feature extraction and matching.

Finally, Line 12 estimates the pose using PnP [49] and RANSAC [30]. This latter algorithm returns an inlier ratio, which estimates what fraction of matches do *not* correspond to outliers (often resulting from noise). UbiPose uses this to estimate the quality of the pose estimate, as described above. Using inlier ratios as an estimate of localization quality is fairly standard in the localization literature [14, 22].

A hybrid design. VIO is fast, but drifts over time. The localizer produces good accuracy, but doesn't allow per-frame operation. UbiPose uses its localizer to periodically generate an accurate anchor, and uses VIO to calculate poses relative to the latest anchor in real-time.

Accuracy and Performance. As we show in §4, this approach results in accurate AR pose tracking. Relative to prior approaches, it is novel in rendering and extracting features

Algorithm 2: Localizer

```
1  Function Localizer (Query image Q, List of rendered images
     R, Initial correspondences C)
2      if C is not None then
3          Correspondences=C;
4      else
5          Correspondences=[];
6      end
7      Invoke Extract Q and each rendered image in R;
8      for each rendered image r in R do
9          Invoke Match between Q and r;
10         Add new 2D to 3D correspondences from matches
             based on OpenGL depth map;
11     end
12     Run PnP + RANSAC with Correspondences and return
         pose estimation with statistics;
13 end
```

Type	Approach	Render	Extract	Match	Localize
Full	Basic (Alg. 1)	3	4	3	1
Fast (Accept)	**Lift-and-Project**	1	2	1	1
Fast (Reject)	**Early-Exit**	1	2	1	1
Full (Optimized)	**Fused-Match**	3	4	2	2

Table 2: Summary of optimizations in the optimized tracker.

online using VIO pose estimates. UbiPose's approach is necessary because straightforward extensions[2] of prior approaches to aerial meshes do not produce accurate tracking (§4.3).

Unfortunately, the tracker and localizer (Algs. 1 and 2) are resource hungry. Specifically, on a modern mobile GPU (the NVIDIA Jetson Xavier NX [62]) neural feature matching and extraction using SuperPoint and SuperGlue require approximately 35 and 150 ms each per invocation, and Alg. 2 invokes the former 4 times, and the latter 3 times. The next section describes how UbiPose optimizes the localizer to reduce UbiPose's resource footprint, essential both to reduce the latency of pose tracking and to reduce power consumption.

3.4 The Optimized Tracker

Overview. Tbl. 2 summarizes these optimizations. Two of these, *Lift-and-Project* and *Early-Exit*, represent fast paths for acceptance and rejection of pose estimates, respectively. *Fused-Match* optimizes the basic tracker when the fast paths do not lead to a conclusive acceptance or rejection.

These optimizations work together as follows:

(1) UbiPose runs the query image Q on Lift-and-Project. If that produces a good pose, it accepts that and returns.
(2) If not, it runs Early-Exit, which reuses computations from Step 1, but determines if pose estimate is likely to be bad.
(3) If it does not Early-Exit, UbiPose uses Fused-Match to reduce Match invocations.

[2]We also tried constructing an SfM model with more robust learned features from SuperPoint. This gives low median but very high p95 errors (§4.3).

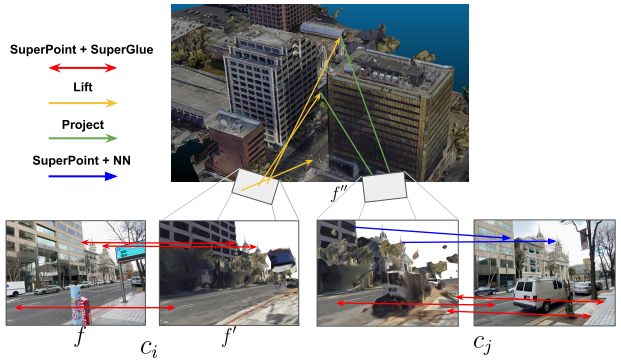

Figure 8: Illustration of the Lift-and-Project optimization.

Lift-and-Project. This optimization (Fig. 8) reuses matched features from previous camera frames.

Key Idea. Consider a camera image c_i. Suppose some feature f matched a feature f' in a rendered image (using Match). From this match, UbiPose can estimate the 3D position of f (it *lifts* f into 3D space). Now, consider a camera image c_j captured a short while later. UbiPose *projects* f onto a feature f'' in c_j using the VIO estimates when c_i and c_j we captured. This exploits the accuracy of VIO over short timescales. Our key insight is that f'' can be used to match features in c_j using very cheap feature-matching (*e.g.,* nearest neighbor matching [58]), since f'' is of the same modality (*i.e.,* from a camera image) as features in c_j. The following paragraphs describe this approach in detail.

The Lift Cache. At the core of Lift-and-Project is the Lift Cache, a cache of lifted features and their corresponding 3D positions from recent camera images. Initially, this cache is empty. To fill this cache, UbiPose follows a Full path (Tbl. 2) to obtain a pose estimate. As part of this path, it obtains all the inlier feature matches from RANSAC (*e.g.,* Line 5 in Alg. 1). It stores all inlier feature vectors and their corresponding 3D positions, as well as the current VIO pose in the lift cache.

Using the Lift Cache. Alg. 3 (and Fig. 8) describes how UbiPose uses the Lift cache once populated. Given a query image Q, it renders (Line 3) a *single* image (unlike 3 images in Alg. 1), extracts features from Q and the rendered image, and matches them (Line 4). The goal of this step is to lift the matched 2D features in Q to 3D (Line 5), leveraging the fact that every pixel in the rendered image has an associated 3D position obtained from the mesh. [3]

Next, the algorithm projects each cached feature to Q, using the VIO pose obtained when storing the feature (Line 6). These projected features, and the lifted features in Line 5 are all from camera images. Instead of using Match, UbiPose

[3]SLAM algorithms track map points in a similar way. However, they algorithms need to triangulate to find the 3D positions of map points, but UbiPose can get these directly from the aerial mesh.

Algorithm 3: Optimized Algorithm

```
1  for each query image Q do
2      Compute the current pose estimate P_V using VIO;
3      Use P_V to render one image R_1 from aerial mesh;
4      Run Extract on Q and R_1 and Match on the pair;
5      Lift features in Q from the match;
6      Project matches M from the Lift Cache;
7      Run nearest-neighbor search to find matches between M
         and Q;
8      Get localization estimate L on matched features;
9      if inlier ratio from L ≥ δ_L and relative error between P_L
         from L and P_v ≤ δ_P then
10         Accept P_L and return;
11         Add inliers to the Lift Cache;
12         Run cache management operations;
13     else
14         Move to test for Early-Exit;
15     end
16 end
```

uses a much cheaper nearest neighbor match [58] (Line 7) and gets a localization estimate for these features using PnP and RANSAC (similar to Line 12 in Alg. 2).

If it decides to reject the pose estimate (using the same criteria as in Line 5 of Alg. 1), UbiPose proceeds to check if it should exit early, described below.

If, however, it accepts the pose estimate, this fast path terminates (Line 10), requiring only one localization, one render, two extract and one match operation (Tbl. 2). Before exiting, it adds the inlier matches to the Lift Cache (Line 11) and performs cache management operations (Line 12).

Managing the Lift Cache. UbiPose evicts points in the cache that have not recently contributed to matches. For each feature point in the cache, it keeps track of how many subsequent camera images the point was *visible* in, and in how many of those it *contributed* to an inlier match. It evicts points whose ratio of contribution count to visibility count is below a certain threshold.

Early-Exit. Visual feature matching works well in environments with sharply defined static structures in the environment (such as buildings). If however, at some locations, such structures are occluded (*e.g.,* due to a tree), it may be difficult to get good visual localization. With aerial meshes updated on the timescale of months [35], feature matching may also not work well if the environment has changed (*e.g.,* if trees have shed leaves, or been cut down). In these cases, UbiPose can avoid work by short-circuiting localization.

Our key insight is that UbiPose can reuse computations in Lift-and-Project to *exit localization early* if it is unlikely to be able to localize at that location. The inlier ratio, returned by RANSAC, is a signal for the quality of localization. In previous steps, a high inlier ratio signals a high quality localization. In Early-Exit, if the inlier ratio returned in Line 8 of Alg. 3 is below another threshold δ_E, UbiPose uses the VIO pose estimate and returns without proceeding further. Early-Exit is conservative, since Line 8 uses matches both from the Lift Cache and from Match on the single rendered frame.

Thus, when UbiPose decides to Early-Exit on a query image Q, it re-uses the one render, two Extract, one Match and one localization operation performed for Lift-and-Project (*i.e.,* it incurs no additional operations).

Fused-Match. If Lift-and-Project does not produce an acceptable pose estimate, and UbiPose does not Early-Exit, it can try to localize using the basic tracker in Alg. 1. To do so, it would invoke the following steps:

(1) Render images R_2 and R_3 and invoke Extract on them.
(2) Invoke Match pairwise on Q and R_2 and R_3.
(3) Feed these *and* the matches from Line 4 and Line 7 to the localizer, and decide whether to accept the resulting estimate or not (using the criteria in Line 5 of Alg. 1).

Because Match uses neural feature matcher like SuperGlue, it is compute-intensive, so reducing the number of invocations in Step 2 above can reduce latency and resource usage. Fused-Match is an optimization that reduces the two Match invocations in Step 2 to one. It leverages the fact that the exact 3D positions of all features can be obtained from the aerial mesh. So, Fused-Match projects all features obtained in Step 1 to a single image plane, then runs Match *once* on that image (Step 2). This saves a Match invocation, but the way we have structured our optimizations, Fused-Match requires an additional invocation to the localizer relative to Alg. 1. Localization is faster than neural feature matching, so this results in higher overall efficiency.

Naïvely fusing feature points can result in multiple features from different images being projected to the same or nearby pixels. This can adversely impact Match accuracy, since matching relies on feature uniqueness. Fused-Match filters out a feature if it has already projected a feature to a nearby pixel. This results not only in more accurate matching, but also faster matching, since matching takes time proportional to the number of features.

3.5 Other Details

Obtaining Initial Pose. To produce a pose estimate in world coordinates, VIO needs an anchor. When an app initiates AR pose tracking, UbiPose needs to generate an *initial anchor* pose. It could have used GPS position and heading, but GPS can be erroneous in obstructed environments. Instead, it uses visual localization on the aerial mesh to obtain the initial anchor. Through relatively simple coordinate transformations, whose details we omit, it is possible to convert aerial mesh coordinates to world coordinates.

To obtain the initial anchor, UbiPose uses the GPS and heading from the mobile device at the location at which the

user initiates AR pose tracking to render five images at angles of 15° from the mesh, each with a 30° field-of-view. It proceeds to obtain a pose estimate using steps similar to Alg. 1: invokes Extract on all images, and Match on every pair, then runs the localizer. If the localizer returns a low inlier ratio, UbiPose rejects the pose, and repeats with another camera image.[4] This approach is an order of magnitude more accurate than simply using the GPS position, and UbiPose finds a good initial pose within 2-3 frames on average(§4).

Model compression. The state-of-the-art neural models UbiPose uses for extraction and matching, SuperPoint and SuperGlue, are too resource-intensive for the mobile device. We converted both models to a TensorRT [63] compatible format (as currently available, they use PyTorch [68] and the ONNX Runtime [25]). Using TensorRT, we quantized them to FP16 precision and used kernel auto-tuning to optimize them for our target hardware, the Jetson NX board. These two optimizations enables UbiPose to run both models efficiently on the mobile platform without significant lose of accuracy.

Rendering. To render an image from a mesh, UbiPose uses OpenGL, whose renderer takes the size of the image desired. The renderer also uses the mobile device camera parameters (the extrinsic and intrinsic matrices) to render images with content, quality and perspective similar to the camera image. Rendered images use ambient light on the mesh, with shadows disabled, in order to properly capture texture and color. All of these help increase the efficacy of feature extraction and matching. To achieve fast rendering, we re-implemented a Python renderer [54] in C++ and exploited the mobile GPU.

4 EVALUATION

We quantify UbiPose's performance using real-world traces.

4.1 Methodology

Implementation. Our implementation of UbiPose uses OpenCV [15] for image processing, OpenGL [90] for rendering, TensorRT [63] to compress models and Colmap [76] to estimate poses from 2D-3D correspondences. Our total implementation is over 7500 lines of C++ code.

Mobile Platform. Our implementation runs entirely on an NVIDIA Jetson Xavier NX (6-core 64bit ARM CPU and 384-core Volta GPU) and we use this for all our experiments. Many mobile devices today have hardware comparable to, or better than, the Jetson NX. For example, Apple's Vision Pro [7] headset has an Apple M2 processor [6], which runs at a higher clock rate and contains more cores than the Jetson.

Trace Collection. We developed a simple data collection app on iOS, which allows us to collect traces of the camera images (at 1920×1440 resolution), the VIO poses of the camera, the camera intrinsics and extrinsics, as well as poses of the ARGeoAnchor (if the ARKit's GeoTracking service is available). To capture real-world image quality, we collected all images using off-the-shelf iPhones. We used SensorLog [77] to collect other sensor data including GPS and heading information. We physically visited each location, collected the data and generated ground-truth using SfM (described below) on the traces. Evaluating each trace took several hours to a day, depending on trace length.

[4]This runs during session initialization, which can take several seconds [8].

Trace	City	Type	UbiPose		ARKit	
			pos. (m)	orien. (°)	pos. (m)	orien. (°)
A	San Jose	Streetside	**0.5 (1.4)**	0.7 (1.2)	1.4 (2.0)	**0.5 (1.1)**
B	San Jose		**0.7 (0.9)**	1.0 (1.8)	0.7 (1.1)	**0.6 (1.4)**
C	Pittsburgh		**0.6 (0.9)**	**0.8 (1.8)**	2.0 (3.7)	3.6 (5.2)
D	Los Angeles	University	**0.5 (0.9)**	0.8 (1.5)	0.7 (1.4)	**0.5 (1.3)**
E	Los Angeles		**0.7 (1.7)**	1.0 (2.0)	0.7 (1.7)	**0.8 (2.3)**
F	Los Angeles		**0.4 (1.1)**	**0.8 (1.7)**	1.1 (4.6)	1.2 (3.0)
G	San Jose	Apartment	0.5 (0.9)	0.9 (1.9)	**0.5 (0.9)**	**0.9 (1.4)**

Table 3: Pose accuracy at locations where ARKit *is* available.

Trace Locations. We used this app to collect traces at about 17 different locations in three different metropolitan areas **Los Angeles, CA**, **San Jose, CA** and **Pittsburgh, PA** in the U.S. In these areas, we collected traces at university campuses, shopping centers, streets, corporate campuses, and apartment complexes by walking in these areas with the camera held forward-facing in landscape mode. Our evaluation focuses on diversity in geography (3 cities) and location types (5). In less than half of these locations (at least one of which was off-street), ARKit was available. All our evaluations use these traces, or a subset thereof. For the areas covered by our traces, the mesh sizes ranged from 30 to 160 MB, with an average of 65 MB. These are well within storage limits on modern mobile devices.

Aerial Meshes. We extracted 3D meshes from Google Earth [38] using Chrome [37], loaded them into Blender [12] using MapsModelImporter [55], and then exported them in a waveform format with texture mapping (.obj and .mtl format).

Metrics. We evaluate UbiPose using two primary metrics: the error in meters of the estimate camera *position* and the error in degrees of its *orientation*. For each trace, we compute the median and p95 values for each metric across all frames for which UbiPose invokes Alg. 3. Additionally, in some experiments, we also quantify UbiPose's median and p95 latency of estimating the pose of a frame, and the median and p95 power draw during the processing of a trace. For the latter, we leverage built-in power tracing in the NX.

Estimating Accuracy. To evaluate accuracy, we need pose ground-truth. Possible approaches to generating pose ground-truth include GNSS-RTK, OptiTrack and LiDAR-SLAM. In dense built environments targeted by AR applications, RTK can have high error due to reflections. Optitrack[65] is more suitable for indoor settings [66]. LiDAR-SLAM can drift by

tens of centimeters and requires calibrating the camera and the LiDAR, which can introduce more measurement error.

Instead, we use *pseudo ground-truth* generated using SfM. Often used in the localization literature [13, 72], this method may not perfectly estimate *absolute* error, but enables us to compare UbiPose with ARKit against a common reference. Pseudo ground-truth can be susceptible to local minima [13]. To overcome this, Brachmann *et al.* [13] suggest choosing evaluation thresholds large enough that the variation in the pseudo ground-truth is less likely to affect the measured performance. We choose p95 to account for such variations.

UbiPose Accuracy. To evaluate the accuracy of UbiPose, we generate an SfM model of the trace using Colmap. This produces a set of camera poses for each image and a point cloud. We align this point cloud to the aerial mesh using iterative closest point, ICP [11]. This enables us to map the SfM's image poses to the mesh's coordinate system, and hence to evaluate the accuracy of UbiPose's poses against the ground truth provided by the SfM model.

ARKit accuracy. We use the same SfM model to estimate ARKit accuracy as well, but need a way to transform the camera pose exposed by ARKit's pose tracker (called *GeoTracking*). The tracker does not expose camera pose in world coordinates, but instead provides the pose of the geo-anchor (§2) and the camera pose in the AR session coordinates. To address this, we first calculate the relative pose between the geo-anchor and the camera at each instant. Then, we align the first image's pose in the trace with the same image in the SfM model. This enables us to transform each ARKit pose estimate in SfM space, and we can estimate error.

4.2 ARKit Comparison

We first compare UbiPose against ARKit using 17 traces whose average duration is 259 s, ranging from 129 s to 346 s. ARKit is available[5] at some locations and not in others, so we discuss these separately.

Locations where ARKit is available. Tbl. 3 shows the results of seven traces, labeled **A**-**G**, at locations where ARKit was available. These locations span public streets in cities (**A**-**C**), universities (**D**-**E**), and a public street through an apartment complex (**G**). One of these, **F**, is in a pedestrian-only zone on a university campus, evidence of terrestrial imagery collection using pedestrians or bicycles.

UbiPose achieves about 0.5 m median and a little over 1 m p95 positioning error. Its median orientation error is, in most cases, 1° or less, and its p95 orientation error ranges from 1.2° to 2°. UbiPose's performance is not strongly correlated with which city it was collected in, and where. For example, across traces D-F, in a University in **Los Angeles** it has both the lowest and one of the highest median positioning errors.

[5]We use this as shorthand for "ARKit's GeoTracking capability is available".

Trace	City	Type	pos. (m)	orien. (°)
H	San Jose	Corporate	0.3 (0.6)	0.3 (0.7)
I	San Jose	Apartment	0.4 (1.1)	0.9 (1.9)
J	Los Angeles	Shopping	0.6 (1.2)	0.7 (1.2)
K	San Jose	Shopping	0.4 (1.0)	0.6 (1.6)
L	San Jose	Shopping	0.3 (1.1)	0.8 (1.1)
M	Los Angeles	University	0.6 (0.9)	0.8 (1.3)
N	Los Angeles	University	0.3 (0.5)	1.0 (1.8)
O	Los Angeles	Apartment	0.5 (0.9)	0.8 (1.3)
P	Pittsburgh	University	0.6 (0.9)	0.7 (0.9)
Q	Pittsburgh	University	0.7 (1.0)	0.4 (0.7)

Table 4: UbiPose accuracy where ARKit is *unavailable*.

In contrast, ARKit exhibits median positioning errors of 0.5 m to almost 2 m in some cases, and p95 positioning errors well over 1 m. Its orientation errors, however, are low: with a couple of exceptions, the median (p95) orientation error is 0.5-0.9° (less than 2°) in many cases.

Tbl. 3 also shows in bold, for each trace, which approach has better position and orientation accuracy. Generally, UbiPose's positioning accuracy is better than ARKit's, and its orientation accuracy is slightly worse. ARKit's better orientation results may be a result of better fusion of its inertial sensors; UbiPose can be extended to exploit these sensors, and we have left this to future work.

There are some exceptions. In **G**, ARKit is better both in position and orientation. In **C** and **F**, UbiPose is better than ARKit in both. In these, ARKit indicated that it had low confidence in its pose estimates. For all three of these cases, the difference comes down to the quality of the imagery: in **G** the mesh quality is poor, and in the other two, it is better than the terrestrial imagery (we are not sure why, but we note that **F** is at an off-street location on campus, so imagery there was likely obtained using pedestrians). This suggests that a hybrid approach which matches both aerial and terrestrial imagery might give good uniform AR pose tracking performance.

Overall, we conclude that, at least for the traces we have studied, UbiPose's accuracy is comparable to that of ARKit.

Locations where ARKit is not available. Tbl. 4 shows UbiPose's accuracy at locations where ARKit is *not* available. These span all three of our cities, and are from a range of locations: corporate and university campuses, apartment complexes, and outdoor shopping areas.

In these locations, ARKit cannot track pose, but UbiPose is able to do so uniformly well. Its median positioning error ranges from 0.27 m to 0.61 m and its p95 positioning error is less than 1.2 m. Its median orientation error ranges from 0.27° to almost 1°, and its p95 orientation error is below 1.8°. These are qualitatively consistent with UbiPose performance in Tbl. 3, as one might expect: UbiPose accuracy shouldn't correlate with where terrestrial imagery is available.

In these locations again, accuracy does not seem to correlate with city or type of location, at least from our samples.

Trace	SIFT+NN		SP+SG	
	pos. (m)	orien. (°)	pos. (m)	orien. (°)
D	31.1 (91.7)	105.4 (172.7)	0.7 (52.3)	1.4 (130.8)
H	49.9 (179.0)	132.7 (176.8)	0.4 (66.3)	0.6 (101.5)

Table 5: Pose Accuracy by SfM from Aerial Mesh

Moreover, position accuracy is not a predictor for orientation accuracy: **N** has low positioning error, but high orientation error, **H** has low error along both dimensions. Accuracy depends entirely on the mesh quality, which is likely a function of when, and from what height it was collected. It is also a strong function of the degree to which the environment has enough visual features (obtained from street markings, buildings *etc.*) to enable matches. This is true for ARKit as well; in wide open spaces, both ARKit and UbiPose may not be able to track pose as accurately.

Overall, these results suggest that UbiPose can increase coverage of AR pose tracking to locations where ARKit is not available, such as those in Figs. 2 and 3. More important, it can do so without impacting accuracy of tracking in those locations, an important consideration for app developers whose users expect uniform quality of experience.

4.3 Comparison With Other Approaches

SfM from Aerial Mesh. This builds an SfM model from rendered aerial mesh images, and localizes a query image using the SfM model (§3.2). SfM usually uses SIFT features and nearest-neighbor matching (*SIFT+NN*). Because SuperPoint features and SuperGlue matching (*SP+SG*) perform better on meshes [67], we built an SfM model that uses these.

Tbl. 5 depicts results from this experiment. In this experiment, unlike the prior one, we present the results of image localization alone. In other words, we do not use VIO estimates to track pose. We do this to understand whether UbiPose could have used this approach instead of its localizer.

SIFT+NN exhibits unacceptable performance (Tbl. 5), with median positioning error of over 30 m, and median orientation error of 90°. SIFT features do not generalize to aerial mesh rendered images, since these have a very different visual appearance that camera images (§3.2). On the other hand, SP+SG has very low median position and orientation error, but its p95 errors are completely unacceptable (over 50 m and over 100° respectively). As discussed earlier, this results from distortions and low resolution of aerial meshes.

MeshLoc on Aerial Mesh. MeshLoc performs visual localization by extracting features offline from a terrestrial mesh, then matching the query image online against those features. In this section, we evaluate a MeshLoc-like approach, but use an aerial mesh to generate features.

Tbl. 6 shows the accuracy of localizing a query image purely using MeshLoc. The table shows results for traces in which MeshLoc performance deviates significantly from

Trace	City	UbiPose		MeshLoc	
		pos. (m)	orien. (°)	pos. (m)	orien. (°)
A	San Jose	**0.5** (**1.4**)	**0.7** (**1.2**)	0.8 (4.7)	0.8 (3.6)
D	Los Angeles	0.5 (0.9)	0.8 (**1.5**)	0.5 (0.9)	0.8 (2.2)
E	Los Angeles	0.7 (**1.7**)	1.1 (**2.0**)	**0.6** (3.6)	1.1 (4.6)
F	Los Angeles	0.4 (**1.1**)	**0.8** (**1.7**)	0.4 (1.2)	0.9 (4.8)
G	San Jose	**0.5** (**0.9**)	**0.9** (**1.9**)	0.7 (7.6)	1.9 (8.2)
H	San Jose	0.3 (0.6)	0.3 (**0.7**)	0.3 (0.6)	0.3 (0.8)
J	Los Angeles	**0.6** (**1.2**)	**0.7** (**1.2**)	0.7 (3.7)	1.1 (5.7)

Table 6: Accuracy comparison between UbiPose and MeshLoc. Best alternative in bold.

UbiPose. Thus, for traces *not* listed in Tbl. 6, MeshLoc has accuracy comparable to UbiPose's localizer.

MeshLoc compares well with UbiPose in median position and orientation error across all of these traces. However, its p95 errors are generally worse, and in some cases substantially so. For example, in **G**, it has a p95 position error of over 7 m and an orientation error of over 8 m. Other traces where both of these values are high include **J**, **A** and **E**. On a terrestrial mesh, by contrast, its p95 position error is under 0.5 m [67]. As discussed in §3.2, we attribute this to the lower resolution of, and distortions in, the aerial mesh. To be robust to these, UbiPose renders multiple mesh images for feature extraction and matching at the estimate pose.

4.4 Quantifying The Optimized Tracker

Fast Path Invocations. Tbl. 7 shows the statistics, across all our traces, of the percentage of frames which benefited from Lift-and-Project, and from Early-Exit. Both optimizations are crucial for UbiPose. Over 70% of frames across all traces, and over 90% of frames in one of them (**I**), use Lift-and-Project. A smaller percentage of frames on average (about 7%) use Early-Exit, but in one of our traces **B**, it is used 46% of the time. Thus, while Lift-and-Project is uniformly useful, Early-Exit is absolutely critical for at least one of our traces. As we show below, these fast path invocations result in lower latency and lower power consumption.

Impact of Optimizations. Tbl. 8 compares UbiPose to the basic tracker in Alg. 1.

Accuracy. In theory, our optimizations can potentially degrade accuracy. For example, Lift-and-Project projects matches from previous frames assuming short-term VIO pose stability, which can introduce error if that assumption is violated. However, at least for two of our traces (representing traces with ARKit and traces without), relative to the un-optimized tracker, UbiPose has comparable median and p95 position and orientation errors.

Latency. To measure the latency, we instrumented the C++ implementation of UbiPose to record the time required for pose tracking for each image. UbiPose's median latency per frame is 2× smaller that of the un-optimized version. UbiPose currently localizes camera frames every second, so our

	Lift-and-Project	Early-Exit
Median	73.2 %	2.7 %
Average	73.9 %	7.2 %
Min	31.8 %	0 %
Max	91.5 %	46.9 %

Table 7: Percentage of frames that benefit from optimizations.

optimizations free up resources for other tasks on the mobile device. In contrast, without these optimizations, the device would be busy most of the time running UbiPose. UbiPose's p95 latency is higher than the un-optimized tracker because of its additional localizer invocation (§3.4). Individually, the median latenciy for Lift-and-Project is 220ms, Early-exit is 90ms, and Fused-Match is 420ms. However, the latency speed-ups of UbiPose's optimization cannot be attributed to individual modules since they rely on each other (Early-Exit and Fused-Match rely on Lift-and-Project).

Power consumption. Because we target mobile deployments, power usage is an important factor to consider. UbiPose consumes 5.7W (median) and 6.9W (p95) both of which are lower than the basic algorithm's power consumption of 7W (median) and 7.8W (p95). For context, the iPhone 12 Pro Max's GPU has comparable average power consumption [32]. UbiPose's improvements come from the Lift-and-Project optimization that reuses feature points, thereby reducing the number of neural network model inferences required.

4.5 Other Results

Memory footprint UbiPose needs 700MB for the mesh, 1GB for TensorRT-generated NN models, and 20MB working memory. These are well within the Jetson's 8GB RAM.

Initial Pose. Tbl. 9 displays the accuracy of initial pose estimation (§3.5). For context, it also shows the GPS error at that location. UbiPose's initial pose is 5× more accurate than GPS location, and it finds an acceptable initial pose within 2-3 frames on average, with a p95 position error of 2.3 m and p95 orientation error of 2.5°.

One-shot Neural Matchers. Recent fast one-shot neural matchers [20, 81] perform Extract and Match in one step, and are plausible candidates for UbiPose's localizer. However, LoFTR [81], when used for visual localization using an aerial mesh, incurs a position error of 10 m (70 m) and orientation error of 22.9° (145°) on trace **F** (and comparable error on trace **H**, omitted for brevity), too high to be useful for UbiPose.

Generalizing to other Cameras. Our experiments use iPhone cameras. AR headsets have qualitatively different cameras; for instance, the Hololens has a grayscale camera. To test whether UbiPose generalizes to these, we collected a trace using a grayscale stereo camera with lower resolution and different calibration and auto-exposure algorithms. Unmodified UbiPose achieves, on this trace, a position error of 0.2 m (0.5 m) and orientation error of 0.4° (1.7°). This result suggests that UbiPose may be able to generalize to AR headsets.

5 LIMITATIONS AND FUTURE WORK

Mesh Availability. The degree to which UbiPose enables ubiquitous pose estimation depends on mesh availability, which in turn depends on depends on mesh providers. In UbiPose's experiments, we used the aerial mesh from Google Earth [38]. Prior work [10] shows that Google Earth covers 97% of U.S.'s and 86% of Canada's metro areas [18, 39]. In some suburban or rural areas, Google Earth only provides satellite images (2D) but not meshes (3D). In these, UbiPose will not be able to provide accurate pose estimates, but neither can terrestrial imagery [59].

Mesh Freshness. The freshness of the aerial mesh can impact the accuracy of pose estimation. We have observed in some of our experiments that stale meshes containing trees with or without leaves, or new construction, can result in fewer matched features between camera images and the mesh rendered image, leading to degraded accuracy or even localization failure (§3.4). Localization using terrestrial imagery can be similarly impacted by stale images. One might assume that terrestrial imagery would always have higher freshness, but anecdotal evidence suggests otherwise. In one of our evaluation locations, the aerial mesh was from 2022, the street-level imagery from 2016. In another, street-level imagery was from Nov 2022, aerial from May 2022.

Quality of aerial-mesh. While our experiments use the highest quality mesh available from Google Earth, we find that aerial meshes have poorer quality than terrestrial meshes. Our techniques in §3 essentially compensate for this quality difference (§3.3). If higher quality aerial meshes are available in the future, UbiPose might be able to achieve accurate, and cheaper, pose estimation.

To illustrate the quality differences between aerial and terrestrial meshes, we took a terrestrial mesh from Aachen, Germany (used by MeshLoc [67]) and obtained an aerial mesh for the same location. By analyzing these, we found three factors that impact UbiPose's localization accuracy. Aerial images have a different perspective than a mobile device camera image; this negatively impacts feature correspondence. Because aerial images are captured from a distance, aerial meshes often lack texture, resulting in poorer feature matching. For a similar reason, the 3D positions of points in the aerial mesh can be inaccurate, relative to a terrestrial mesh. This reduces the accuracy of PnP [49] (§3.3).

Other Designs and Extensions. Future work can explore offloading computation from lower-end mobile devices to edge and/or cloud [41, 89] to maintain performance SLOs. We have left it to future work to integrate our implementation into iOS and Android. Apple doesn't provide guidance on how to run

Trace	UbiPose				Basic Algorithm (Alg. 1)			
	pos. (m)	orien. (°)	power (mW)	latency (ms)	pos. (m)	orien. (°)	power (mW)	latency (ms)
A	0.3 (0.6)	0.3 (0.7)	5670 (6604)	364 (755)	0.3 (0.6)	0.3 (0.9)	7097 (7845)	672 (723)
H	0.5 (0.8)	0.8 (1.4)	5711 (6951)	366 (783)	0.5 (1.2)	0.8 (1.2)	6892 (7577)	681 (745)

Table 8: Latency of UbiPose in pose tracking with optimized and basic algorithm

Trace	GPS Error (m)	Initial Pose Error	
		pos. (m)	orien. (°)
D	3.3 (8.9)	0.6 (1.3)	0.9 (1.7)
H	12.1 (20.1)	0.6 (2.3)	0.5 (2.5)

Table 9: Initial pose accuracy of UbiPose

and optimize third-party ML models on Apple's neural engine [40]. Android's TensorFlow Lite quantization tools [84] don't improve SuperGlue performance in our experiments, and TensorRT, which we rely upon for model compression, doesn't support Apple hardware as a back-end. Finally, both UbiPose and ARKit estimate pedestrian-carried camera poses; localization of aerial (*e.g.,* drone) camera images [21] is an interesting direction for future work. Future work can also train feature extractors and matchers for cross-modality matching, detect VIO drift to optimize visual localization invocations, and explore extensibility to AR headsets.

6 RELATED WORK

Sparse feature-based visual localization. Existing visual localization approaches [42, 50, 69, 72, 73, 76, 82] represent scenes using sparse features [21], often in the form of Structure-from-Motion (SfM) point clouds containing local features extracted from database images. These approaches use different feature extraction (*e.g.,* SIFT [51], ORB [71] and SuperPoint [24]) and matching techniques (*e.g.,* nearest neighbor [58], SuperGlue [74] and D2-Net [29]). As we show in §4 they are unsuitable for use with aerial meshes. UbiPose uses multi-view matching techniques differently than the existing approaches [19, 53]; it matches feature points across two modalities and does not need to triangulate to find the 3D positions of the feature points.

Mesh and Dense 3D model based localization. LandscapeAR [17] estimates pose to within 100 m using digital elevation models (DEMs). Zhang et al. [93] use learned features and view synthesis to generate reference poses for a given query image, but this takes 10-20 s per frame. MeshLoc [67] uses terrestrial meshes for neural feature extraction and matching. UbiPose obtains high pose accuracy in areas where terrestrial meshes may not be available and can run fast on mobile devices.

SLAM based visual localization. SLAM [28] simultaneously estimates pose and builds a 3D map of an environment. Visual SLAM [19], like SfM, builds sparse 3D feature maps of the environment. If these maps were widely available, they could potentially enable ubiquitous pose tracking, but would have the same drawback as terrestrial imagery: at scale, they could only be collected using vehicles [1]. UbiPose, using aerial meshes, enables wider coverage for AR pose tracking.

Other camera pose estimation and tracking approaches. PoseNet [46] is a CNN-based 6-DoF pose estimator. A line of work has improved upon PoseNet using various techniques [16, 27, 57, 78, 79, 86–88]. Even though these techniques can estimate absolute pose estimation quickly, their accuracy is worse than approaches, like UbiPose, that exploit imagery or dense structural models of the environment. Other work explores estimating pose leveraging moir'e patterns' high sensitivity to the camera's pose changes [60], and improving pose tracking using inertial sensors [2, 75, 91].

7 CONCLUSIONS

UbiPose extends coverage of AR pose tracking on mobile devices to areas where terrestrial imagery is not available. It uses aerial meshes, a novel tracker and localizer that uses VIO estimates to render images from the aerial mesh on the mobile device, then performs feature extraction and matching. It uses several optimization to reduce the high cost of neural extraction and matching, and achieves accuracy comparable to ARKit in areas where that is available. In off-street locations without terrestrial imagery, UbiPose is able to accurately track pose while ARKit cannot.

REFERENCES

[1] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. 2020. CarMap: Fast 3D Feature Map Updates for Automobiles. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Santa Clara, CA, 1063–1081. https://www.usenix.org/conference/nsdi20/presentation/ahmad

[2] Mary Alatise and Gerhard P Hancke. 2017. Pose estimation of a mobile robot using monocular vision and inertial sensors data. In *IEEE AFRICON*. 1552–1557. https://doi.org/10.1109/AFRCON.2017.8095713

[3] Jesús Omar Álvarez Márquez and Jürgen Ziegler. 2020. In-Store Augmented Reality-Enabled Product Comparison and Recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys '20)* (Virtual Event, Brazil). ACM, New York, NY, USA, 180–189. https://doi.org/10.1145/3383313.3412266

[4] Apple. 2020. ARKit 4 GeoTracking. https://developer.apple.com/videos/play/wwdc2020/10611/?time=246.

[5] Apple. 2021. Apple Maps introduces new ways to explore major cities in 3D. https://www.apple.com/newsroom/2021/09/apple-maps-introduces-new-ways-to-explore-major-cities-in-3d/.

[6] Apple. 2023. Apple unveils M2, taking the breakthrough performance and capabilities of M1 even further. https://www.apple.com/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities/

[7] Apple. 2023. Apple Vision Pro. https://www.apple.com/apple-vision-pro/

[8] Apple. 2023. Tracking geographic locations in AR. https://developer.apple.com/documentation/arkit/content_anchors/tracking_geographic_locations_in_ar.

[9] Apple. 2023. Understanding World Tracking. https://developer.apple.com/documentation/arkit/configuration_objects/understanding_world_tracking

[10] Justino Beirne. 2021. https://www.justinobeirne.com/google-maps-apple-maps-3d-imagery-coverage

[11] P. J. Besl and N. D. McKay. 1992. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14, 2 (1992), 239–256.

[12] Blender. 2023. Blender. https://www.blender.org/.

[13] Eric Brachmann, Martin Humenberger, Carsten Rother, and Torsten Sattler. 2021. On the Limits of Pseudo Ground Truth in Visual Camera Re-localisation. https://doi.org/10.48550/ARXIV.2109.00524

[14] Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and Carsten Rother. 2016. DSAC - Differentiable RANSAC for Camera Localization. https://doi.org/10.48550/ARXIV.1611.05705

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

[16] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. 2018. Geometry-aware learning of maps for camera localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2616–2625.

[17] Jan Brejcha, Michal Lukáč, Yannick Hold-Geoffroy, Oliver Wang, and Martin Čadík. 2020. LandscapeAR: Large Scale Outdoor Augmented Reality by Matching Photographs with Terrain Models Using Learned Descriptors. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 295–312.

[18] US Census Bureau. 2023. Metropolitan and Micropolitan Statistical Areas Population Totals: 2020-2022. https://www.census.gov/data/tables/time-series/demo/popest/2020s-total-metro-and-micro-statistical-areas.html#v2022

[19] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. 2021. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *IEEE Transactions on Robotics* 37, 6 (2021), 1874–1890.

[20] Hongkai Chen, Zixin Luo, Lei Zhou, Yurun Tian, Mingmin Zhen, Tian Fang, David McKinnon, Yanghai Tsin, and Long Quan. 2022. ASpanFormer: Detector-Free Image Matching With Adaptive Span Transformer. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII* (Tel Aviv, Israel). Springer-Verlag, Berlin, Heidelberg, 20–36. https://doi.org/10.1007/978-3-031-19824-3_2

[21] Shuxiao Chen, Xiangyu Wu, Mark W. Mueller, and Koushil Sreenath. 2021. Real-Time Geo-Localization Using Satellite Imagery and Topography for Unmanned Aerial Vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Prague, Czech Republic). IEEE Press, 2275–2281. https://doi.org/10.1109/IROS51168.2021.9636705

[22] Siddharth Choudhary and P. J. Narayanan. 2012. Visibility Probability Structure from SfM Datasets and Applications. In *Computer Vision – ECCV 2012*, Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 130–143.

[23] Edmanuel Cruz, Sergio Orts-Escolano, Francisco Gomez-Donoso, Carlos Rizo, Jose Carlos Rangel, Higinio Mora, and Miguel Cazorla. 2019. An Augmented Reality Application for Improving Shopping Experience in Large Retail Stores. *Virtual Reality* 23, 3 (Sep. 2019), 281–291. https://doi.org/10.1007/s10055-018-0338-3

[24] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. 2018. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) workshops*. 224–236.

[25] ONNX Runtime developers. 2021. ONNX Runtime. https://onnxruntime.ai/. Version: x.y.z.

[26] Daniel Dilger. 2017. Inside Apple's ARKit and Visual Inertial Odometry, new in iOS 11. https://appleinsider.com/articles/17/10/12/inside-apples-arkit-and-visual-inertial-odometry-new-in-ios-11.

[27] Mingyu Ding, Zhe Wang, Jiankai Sun, Jianping Shi, and Ping Luo. 2019. CamNet: Coarse-to-fine retrieval for camera re-localization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2871–2880.

[28] H. Durrant-Whyte and T. Bailey. 2006. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine* 13, 2 (2006), 99–110. https://doi.org/10.1109/MRA.2006.1638022

[29] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. 2019. D2-Net: A Trainable CNN for Joint Detection and Description of Local Features. In *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[30] Martin A. Fischler and Robert C. Bolles. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM* 24, 6 (June 1981), 381–395. https://doi.org/10.1145/358669.358692

[31] U.S. Space Force. 2022. GPS performance. https://www.gps.gov/systems/gps/performance/

[32] Andrei Frumusanu. 2020. The iphone 12 and 12 pro review: New design and diminishing returns. https://www.anandtech.com/show/16192/the-iphone-12-review/4

[33] Yasutaka Furukawa and Carlos Hernández. 2015. Multi-View Stereo: A Tutorial. *Found. Trends. Comput. Graph. Vis.* 9, 1–2 (jun 2015), 1–148. https://doi.org/10.1561/0600000052

[34] Fotis Giariskanis, Yannis Kritikos, Eftychia Protopapadaki, Anthi Papanastasiou, Eleni Papadopoulou, and Katerina Mania. 2022. The Augmented Museum: A Multimodal, Game-Based, Augmented Reality Narrative for Cultural Heritage. In *ACM International Conference on Interactive Media Experiences (IMX '22)* (Aveiro, JB, Portugal). ACM, New York, NY, USA, 281–286. https://doi.org/10.1145/3505284.3532967

[35] Google. 2022. Google Earth updates. https://gadgetsbeat.com/how-often-does-google-earth-update-photos-maps/

[36] Google. 2022. Overview of arcore and supported development environments | google developers. https://developers.google.com/ar/develop#how_does_arcore_work.

[37] Google. 2023. Google Chrome. https://www.google.com/chrome/.

[38] Google. 2023. Google Earth. https://earth.google.com/.

[39] Statistics Canada Government of Canada. 2016. Census metropolitan areas (Canada). https://www23.statcan.gc.ca/imdb/p3VD.pl?Function=getVD&TVD=314312&CVD=314313&CPV=A&CST=01012016&CLV=1&MLV=3

[40] Matthijs Hollemans. 2023. The Neural Engine — what do we know about it? https://github.com/hollance/neural-engine.

[41] Yitao Hu, Weiwu Pang, Xiaochen Liu, Rajrup Ghosh, Bongjun Ko, Wei-Han Lee, and Ramesh Govindan. 2021. Rim: Offloading Inference to the Edge. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation* (Charlottesvle, VA, USA) *(IoTDI '21)*. Association for Computing Machinery, New York, NY, USA, 80–92. https://doi.org/10.1145/3450268.3453521

[42] Martin Humenberger, Yohann Cabon, Nicolas Guerin, Julien Morat, Jérôme Revaud, Philippe Rerole, Noé Pion, Cesar de Souza, Vincent Leroy, and Gabriela Csurka. 2020. Robust image retrieval-based visual localization using kapture. *arXiv preprint arXiv:2007.13867* (2020).

[43] Google Inc. 2022. Street View: How it works. https://www.google.com/streetview/how-it-works/.

[44] Hamraz Javaheri, Maryam Mirzaei, and Paul Lukowicz. 2021. How Far Can Wearable Augmented Reality Influence Customer Shopping Behavior. In *17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '20)* (Darmstadt, Germany). ACM, New York, NY, USA, 464–469. https://doi.org/10.1145/3448891.3448930

[45] Nan Jiang, Debin Huang, Jing Chen, Jie Wen, Heng Zhang, and Honglong Chen. 2021. Semi-Direct Monocular Visual-Inertial Odometry Using Point and Line Features for IoV. *ACM Transactions on Internet Technology* 22, 1, Article 5 (February 2021), 23 pages. https://doi.org/10.1145/3432248

[46] Alex Kendall, Matthew Grimes, and Roberto Cipolla. 2015. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*. 2938–2946.

[47] Takumi Kiriu, Mohit Mittal, Panote Siriaraya, Yukiko Kawai, and Shinsuke Nakajima. 2019. Development of an Acoustic AR Gamification System to Support Physical Exercise. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19)* (Nice, France). ACM, New York, NY, USA, 1056–1058. https://doi.org/10.1145/3343031.3350589

[48] Amanda Krause. 2021. Google Maps has now photographed 10 million miles in Street View. https://www.cnet.com/tech/tech-industry/google-maps-has-now-photographed-10-million-miles-in-street-view/

[49] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. 2009. EP n P: An accurate O (n) solution to the P n P problem. *International journal of computer vision* 81 (2009), 155–166.

[50] Yunpeng Li, Noah Snavely, Daniel P Huttenlocher, and Pascal Fua. 2016. Worldwide pose estimation using 3d point clouds. *Large-Scale Visual Geo-Localization* (2016), 147–163.

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

[52] Simon Lynen, Bernhard Zeisl, Dror Aiger, Michael Bosse, Joel A. Hesch, Marc Pollefeys, Roland Siegwart, and Torsten Sattler. 2019. Large-scale, real-time visual-inertial localization revisited. *CoRR* abs/1907.00338 (2019). arXiv:1907.00338 http://arxiv.org/abs/1907.00338

[53] Eleonora Maset, Federica Arrigoni, and Andrea Fusiello. 2017. Practical and Efficient Multi-view Matching. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 4578–4586. https://doi.org/10.1109/ICCV.2017.489

[54] Matthew Matl. 2022. Pyrender Documentation. https://pyrender.readthedocs.io/en/latest/examples/index.html.

[55] Elie Michel. 2023. Map Model Importers. https://github.com/eliemichel/MapsModelsImporter.

[56] Microsoft. 2023. Streetside view. https://www.microsoft.com/en-us/maps/streetside.

[57] Arthur Moreau, Nathan Piasco, Dzmitry Tsishkou, Bogdan Stanciulescu, and Arnaud de La Fortelle. 2022. LENS: Localization enhanced by NeRF synthesis. In *Conference on Robot Learning*. PMLR, 1347–1356.

[58] Marius Muja and David G Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)* 2, 331-340 (2009), 2.

[59] Stephen Ninan and Sivakumar Rathinam. 2022. LIDAR data based Segmentation and Localization using Open Street Maps for Rural Roads. arXiv:2202.07049 [cs.RO]

[60] Jingyi Ning, Lei Xie, Yi Li, Yingying Chen, Yanling Bu, Baoliu Ye, and Sanglu Lu. 2022. MoiréPose: Ultra High Precision Camera-to-Screen Pose Estimation Based on Moiré Pattern. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking (MobiCom '22)* (Sydney, NSW, Australia). ACM, New York, NY, USA, 106–119. https://doi.org/10.1145/3495243.3560537

[61] Marius Noreikis, Yu Xiao, and Antti Ylä-Jääski. 2017. SeeNav: Seamless and Energy-Efficient Indoor Navigation Using Augmented Reality. In *Proceedings of the on Thematic Workshops of ACM Multimedia* (Mountain View, California, USA). ACM, New York, NY, USA, 186–193. https://doi.org/10.1145/3126686.3126733

[62] NVIDIA. 2022. The NVIDIA Jetson Xavier NX. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/.

[63] NVIDIA. 2023. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt.

[64] Ifeanyi Paul Odenigbo, Jaisheen Kour Reen, Chimamaka Eneze, Aniefiok Friday, and Rita Orji. 2022. The Journey: An AR Gamified Mobile Application for Promoting Physical Activity in Young Adults. In *Adjunct Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization (UMAP '22 Adjunct)* (Barcelona, Spain). ACM, New York, NY, USA, 342–353. https://doi.org/10.1145/3511047.3537652

[65] OptiTrack. 2023. OptiTrack Documentation. https://docs.optitrack.com

[66] OptiTrack. 2023. Quick Start Guide: Getting Started - OptiTrack Documentation. https://docs.optitrack.com/quick-start-guides/quick-start-guide-getting-started

[67] Vojtech Panek, Zuzana Kukelova, and Torsten Sattler. 2022. MeshLoc: Mesh-Based Visual Localization. In *Proceedings of the 17th European Conference on Computer Vision (ECCV) Part XXII* (Tel Aviv, Israel, October 23-27). Springer-Verlag, Berlin, Heidelberg, 589–609. https://doi.org/10.1007/978-3-031-20047-2_34

[68] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary

DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[69] Lukas Platinsky, Michal Szabados, Filip Hlasek, Ross Hemsley, Luca Del Pero, Andrej Pancik, Bryan Baum, Hugo Grimmett, and Peter Ondruska. 2020. Collaborative augmented reality on smartphones via life-long city-scale maps. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 533–541.

[70] Sebastião Rocha and Arminda Lopes. 2020. Navigation Based Application with Augmented Reality and Accessibility. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems (CHI EA '20)* (Honolulu, HI, USA). ACM, New York, NY, USA, 1–9. https://doi.org/10.1145/3334480.3383004

[71] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision (ICCV)*. 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544

[72] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. 2019. From Coarse to Fine: Robust Hierarchical Localization at Large Scale. In *CVPR*.

[73] Paul-Edouard Sarlin, Frederic Debraine, Marcin Dymczyk, Roland Siegwart, and Cesar Cadena. 2018. Leveraging Deep Visual Descriptors for Hierarchical Efficient Localization. In *Proceedings of The 2nd Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 87)*, Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto (Eds.). PMLR, 456–465. https://proceedings.mlr.press/v87/sarlin18a.html

[74] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. 2020. SuperGlue: Learning Feature Matching with Graph Neural Networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4937–4946. https://doi.org/10.1109/CVPR42600.2020.00499

[75] Gerhard Schall, Daniel Wagner, Gerhard Reitmayr, Elise Taichmann, Manfred Wieser, Dieter Schmalstieg, and Bernhard Hofmann-Wellenhof. 2009. Global pose estimation using multi-sensor fusion for outdoor Augmented Reality. In *The 8th IEEE International Symposium on Mixed and Augmented Reality*. 153–162. https://doi.org/10.1109/ISMAR.2009.5336489

[76] Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[77] SensorLog. 2023. SensorLog. http://sensorlog.berndthomas.net/.

[78] Yoli Shavit, Ron Ferens, and Yosi Keller. 2021. Learning multi-scene absolute pose regression with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2733–2742.

[79] Yoli Shavit and Yosi Keller. 2022. Camera pose auto-encoders for improving pose regression. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part X*. Springer, 140–157.

[80] Avinash Kumar Singh, Jia Liu, Carlos A. Tirado Cortes, and Chin-Teng Lin. 2021. Virtual Global Landmark: An Augmented Reality Technique to Improve Spatial Navigation Learning. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (CHI EA '21)* (Yokohama, Japan). ACM, New York, NY, USA, Article 276. https://doi.org/10.1145/3411763.3451634

[81] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. 2021. LoFTR: Detector-Free Local Feature Matching with Transformers. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 8918–8927. https://doi.org/10.1109/CVPR46437.2021.00881

[82] Hajime Taira, Masatoshi Okutomi, Torsten Sattler, Mircea Cimpoi, Marc Pollefeys, Josef Sivic, Tomas Pajdla, and Akihiko Torii. 2018. InLoc: Indoor visual localization with dense matching and view synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7199–7209.

[83] Natalia Teixeira, Bianca Lahm, Fabiana Frata Furlan Peres, Claudio Roberto Marquetto Mauricio, and Joao Marcelo Xavier Natario Teixeira. 2022. Augmented Reality on Museums: The Ecomuseu Virtual Guide. In *Symposium on Virtual and Augmented Reality (SVR'21)* (Virtual Event, Brazil). ACM, New York, NY, USA, 147–156. https://doi.org/10.1145/3488162.3488219

[84] TensorFlow. 2023. Post-training quantization. https://www.tensorflow.org/model_optimization/guide/quantization/post_training

[85] Tram Thi Minh Tran and Callum Parker. 2020. Designing Exocentric Pedestrian Navigation for AR Head Mounted Displays. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems (CHI EA '20)* (Honolulu, HI, USA). ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/3334480.3382868

[86] Mehmet Ozgur Turkoglu, Eric Brachmann, Konrad Schindler, Gabriel J Brostow, and Aron Monszpart. 2021. Visual camera re-localization using graph neural networks and relative pose supervision. In *2021 International Conference on 3D Vision (3DV)*. IEEE, 145–155.

[87] Florian Walch, Caner Hazirbas, Laura Leal-Taixe, Torsten Sattler, Sebastian Hilsenbeck, and Daniel Cremers. 2017. Image-based localization using lstms for structured feature correlation. In *Proceedings of the IEEE International Conference on Computer Vision*. 627–637.

[88] Bing Wang, Changhao Chen, Chris Xiaoxuan Lu, Peijun Zhao, Niki Trigoni, and Andrew Markham. 2020. Atloc: Attention guided camera localization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 10393–10401.

[89] Jianfeng Wang, Tamás Lévai, Zhuojin Li, Marcos A. M. Vieira, Ramesh Govindan, and Barath Raghavan. 2022. Quadrant: A Cloud-Deployable NF Virtualization Platform. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 493–509. https://doi.org/10.1145/3542929.3563471

[90] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. 1999. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc.

[91] Xinyu Yi, Yuxiao Zhou, and Feng Xu. 2021. TransPose: Real-Time 3D Human Translation and Pose Estimation with Six Inertial Sensors. *ACM Transactions on Graphics* 40, 4, Article 86 (August 2021). https://doi.org/10.1145/3450626.3459786

[92] Yiyi Zhang and Tatsuo Nakajima. 2022. Exploring 3D Landmark-Based Map Interface in AR Navigation System for City Exploration. In *Proceedings of the 20th International Conference on Mobile and Ubiquitous Multimedia (MUM '21)* (Leuven, Belgium). ACM, New York, NY, USA, 220–222. https://doi.org/10.1145/3490632.3497858

[93] Zichao Zhang, Torsten Sattler, and Davide Scaramuzza. 2021. Reference pose generation for long-term visual localization via learned features and view synthesis. *International Journal of Computer Vision* 129 (2021), 821–844.