

# SiTAR: Situated Trajectory Analysis for In-the-Wild Pose Error Estimation

Tim Scargill\*

Ying Chen†

Tianyi Hu‡

Maria Gorlatova§

Department of Electrical and Computer Engineering, Duke University

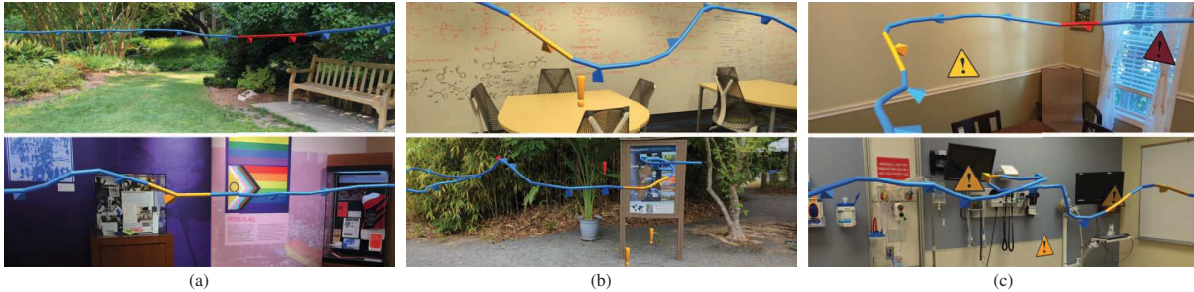


Figure 1: Our situated trajectory analysis system SiTAR in action, identifying environment regions associated with high estimated device pose error. Pictured are examples of environments from our in-the-wild study, in which we tested three visualization techniques: (a) trajectory-only, (b) trajectory + exclamation points, and (c) trajectory + warning signs. Best viewed in color.

## ABSTRACT

Virtual content instability caused by device pose tracking error remains a prevalent issue in markerless augmented reality (AR), especially on smartphones and tablets. However, when examining environments which will host AR experiences, it is challenging to determine where those instability artifacts will occur; we rarely have access to ground truth pose to measure pose error, and even if pose error is available, traditional visualizations do not connect that data with the real environment, limiting their usefulness. To address these issues we present **SiTAR** (Situated Trajectory Analysis for Augmented Reality), the first situated trajectory analysis system for AR that incorporates estimates of pose tracking error. We start by developing the first uncertainty-based pose error estimation method for visual-inertial simultaneous localization and mapping (VI-SLAM), which allows us to obtain pose error estimates without ground truth; we achieve an average accuracy of up to 96.1% and an average F1 score of up to 0.77 in our evaluations on four VI-SLAM datasets. Next, we present our SiTAR system, implemented for AR-Core devices, combining a backend that supplies uncertainty-based pose error estimates with a frontend that generates situated trajectory visualizations. Finally, we evaluate the efficacy of SiTAR in realistic conditions by testing three visualization techniques in an in-the-wild study with 15 users and 13 diverse environments; this study reveals the impact both environment scale and the properties of surfaces present can have on user experience and task performance.

**Index Terms:** Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Mixed / augmented reality;

## 1 INTRODUCTION

As techniques for persistent augmented reality (AR) mature and become more accessible, we are on the cusp of a long-anticipated future in which virtual content is integrated with the real world across

diverse environments. Immersive museum exhibits, navigation guidance in warehouses, factories with virtual displays of sensor data and virtual patient notes in hospitals are just some of the possibilities that can now be realized. However, within this promise still lurks the problem of virtual content instability, caused by AR device pose tracking error. Particularly on handheld devices, the limited field of view of tracking cameras means that challenging environment regions such as blank walls can dominate the visual input data available for the underlying visual-inertial simultaneous localization and mapping (VI-SLAM) algorithm; this results in device pose estimate errors, and virtual content that jitters or drifts out of position [55,60].

How then do AR developers decide which parts of an environment are conducive to accurate pose tracking? In other words, “*If I place virtual content here, is it likely to appear stable to users?*” While AR platforms offer general environment design guidelines (e.g., [27]), the complex interaction between lighting, texture, and the geometric and reflective properties of an environment makes estimating the magnitude of errors extremely challenging. Ideally one would quantify pose error by evaluating trajectories against a ground truth pose measurement, but this is infeasible in the vast majority of scenarios due to the cost and logistics associated with obtaining ground truth, e.g., through optical tracking systems [47,66]. Finally, some AR platforms provide real-time tracking status [4, 29], but they only indicate when tracking is severely compromised or lost completely, and the magnitude of errors is not detected. It is clear we require a pose error estimation method that provides more granular information, and which can be easily deployed in any environment.

Even if accurate pose error estimates are available, another issue remains – how do we best communicate that information? The pose errors calculated through trajectory evaluations are traditionally reported numerically or in 2D plots (e.g., [8,62]), which critically lack any connection with the real environment where a trajectory was created. One might ask, “*There is high error in this part of a trajectory, but which part of the environment does that correspond to? Where was the AR device camera facing at that point?*” This fundamentally limits our ability to understand which parts of an environment are the source of pose estimate errors, and the usefulness of trajectory evaluations for making AR environment design decisions. To fully empower developers to make positive changes that produce high-quality AR experiences, we require not only a method of obtaining accurate pose error estimates, but also a solution that connects those estimates with the environment where a trajectory was created.

To address both of these requirements, we present **SiTAR**

\*e-mail: ts352@duke.edu

†e-mail: ying.chen151@duke.edu

‡e-mail: tianyi.hu@duke.edu

§e-mail: maria.gorlatova@duke.edu

(Situating Trajectory Analysis for Augmented Reality), the first situated trajectory analysis system for AR that incorporates estimates of pose tracking error. We obtain pose error estimates through our development of an uncertainty-based estimation method. Then, to connect those estimates with real environment regions we employ *situated analytics* [19, 24, 35, 44, 63, 64] in the creation of trajectory visualizations. We test three trajectory visualization techniques for SiTAR in an in-the-wild study, to evaluate it in realistic environments, and the efficacy of our visualizations under diverse conditions. Our key contributions are summarized as follows:

- We develop the first uncertainty-based pose error estimation method for VI-SLAM, which achieves an average accuracy of up to 96.1% across four datasets, and provides estimates of pose error magnitude *without the need for ground truth pose data*.
- We present SiTAR, the first situated trajectory analysis system for AR that incorporates pose error estimates, and implement it for ARCore; we generate uncertainty-based estimates using an edge or cloud backend, and visualize them in the environments they are associated with, on real AR devices.
- We design three visualization techniques for highlighting problematic environment regions associated with high pose error, and test them in an in-the-wild study with 15 participants in 13 diverse environments. Participants indicated a preference for visualizations which attached virtual objects to problematic regions, and our results provide insights into how certain environment properties can impact both task performance and user experience.

The rest of the paper is organized as follows. In Section 2 we cover related work, then in Section 3 present our uncertainty-based pose error estimation method. We describe our system for situated trajectory analysis on AR devices in Section 4, develop and evaluate different types of visualization for situated trajectory analysis in Section 5, then summarize our conclusions and future work in Section 6. The code required for implementation of SiTAR is publicly available at <https://github.com/timscargill/SiTAR/>.

## 2 RELATED WORK

**VI-SLAM pose error estimation:** VI-SLAM evaluation tools (e.g., [31, 73]) are traditionally used to calculate pose error by comparing estimated and ground truth trajectories; this reliance on ground truth pose data is suited to the evaluations conducted with VI-SLAM benchmarks (e.g., [6, 16, 36, 37, 62, 76]), but is not scalable to the diverse environments which will host AR. On the other hand, AR platforms such as ARCore and ARKit detect when pose estimates are unavailable or questionable without ground truth [4, 29], but both only capture severe tracking issues, and do not provide estimates of error magnitude. Recent work has shown promise in the prediction of pose error through the characterization of raw sensor inputs and machine learning [3], but this only outputs the global consistency of a trajectory; to highlight specific problematic environment regions we require the local consistency, which our estimation method provides.

**Uncertainty quantification:** Existing works on uncertainty quantification across diverse fields primarily model data uncertainty [1, 10, 41], the ‘noise’ and ‘randomness’ inherent in data, and knowledge uncertainty [1, 70], uncertainty due to a system’s limited knowledge of input data. The Bayesian-based method through posteriori probability formulation is a typical uncertainty quantification method [51]. To reduce the computational complexity, Monte Carlo simulation-based *uncertainty propagation* has been applied to approximate Bayesian results [71] and quantifies uncertainty by repeated sampling. Drawing inspiration yet distinguishing our work from Monte Carlo propagation of data uncertainty induced by measurement errors [34], we are the first to leverage this uncertainty propagation methodology to quantify pose estimate uncertainty in

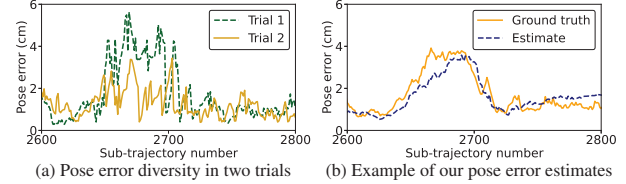


Figure 2: Our uncertainty propagation-based pose error estimation uses (a) the diversity of pose estimates in repeated trials to (b) estimate the average ground truth-based pose error for that input data. Samples shown are from the SenseTime A1 sequence [36].

SLAM. Here we characterize both data uncertainty (sensor noise) and knowledge uncertainty (the non-deterministic nature of SLAM algorithms). Prior works have quantified the uncertainty of the optimization-based SLAM backend for entire trajectories using the maximum likelihood pose estimator covariance matrix, to guide decision-making in SLAM (e.g., [9, 12, 13, 38]); our work differs in that we capture local uncertainty by considering individual sub-trajectories, and that we use the pose estimate *output* to quantify uncertainty, such that our method can be applied to both open-source SLAM and the ‘black-box’ algorithms on commercial AR platforms.

**Situated analytics and semantic trajectories:** We propose and develop a new application for situated analytics [19, 24, 35, 44, 63, 64], data analysis supported by *situated visualizations* – for a recent review see [5]. Specifically, our system semantically enriches AR device trajectory visualizations with pose error estimates; this inclusion of data beyond pose in trajectory visualization is related to work on *semantic trajectories* [2, 49, 69]. One example is coloring 2D trajectory plots according to pose error magnitude in SLAM trajectory evaluations [31], which we build on to produce 3D, situated trajectory visualizations in AR. Previous works have demonstrated visualizing various types of trajectories in AR, including the planning or communicating of robot or drone movements [11, 53, 67, 75] and guidance for navigation [25, 48], medical procedures [15, 20, 74], and sports training [43]. Most closely related is [7], a recent toolkit that provides AR-based situated analytics for AR user movement and interactions. However, to the best of our knowledge our system is the first to enable situated analytics for device pose error.

## 3 UNCERTAINTY-BASED POSE ERROR ESTIMATION

### 3.1 Background and Motivation

Virtual content stability in markerless AR is determined by the accuracy of pose estimates achieved through a VI-SLAM algorithm. However, errors are often present in VI-SLAM-based pose estimates due to challenging visual or inertial input data [36], and as a result virtual content instability is a known issue in AR [55, 60]. The magnitude of errors is challenging to predict; not only can multiple different input data properties cause errors, but the inner VI-SLAM pipeline has multiple elements, and errors can occur at various points. Furthermore, relationships between input data characteristics and pose error obtained in one environment do not generalize to new environments, due to the complex interactions between lighting, textures, and the geometric and reflective properties of objects present.

The core motivation for our uncertainty-based estimation approach is the stochastic nature of pose estimates in state-of-the-art VI-SLAM (e.g., [8, 52]). We illustrate this in Figure 2a by plotting the translational pose error in each 10-frame sub-trajectory used to calculate relative error [73] when running the A1 sequence from the SenseTime benchmark [36] in ORB-SLAM3 [8]. Different trials result in varying error magnitudes, and this variance – pose estimate uncertainty – is generally greater when error is higher. Intuitively, this reflects our observation that *accurate trajectories are alike, whereas inaccurate trajectories are wrong in their own way*. This randomness arises due to the use of RANSAC [23] in SLAM to select a set of good feature matches, inliers, between neighboring camera

images; because RANSAC uses repeated random sub-sampling, and the number of iterations is limited due to real-time operation, it can produce sub-optimal results, particularly when the proportion of outliers is high (i.e., input data are challenging) [46, 54].

### 3.2 Pose Error Estimation Method

Our pose error estimation method is based on the propagation of uncertainty, the effect that the randomness or uncertainty of input variables has on the system output. Uncertainty propagation has been used to quantify output uncertainty in various domains, from transistor models [50] to visual analytics [57]. Specifically, we quantify how the randomness inherent in RANSAC-based inlier selection affects the uncertainty of VI-SLAM pose estimates by performing repeated trials with the same visual and inertial input data, similar to works that use Monte Carlo simulation for uncertainty propagation (e.g., [45]). To support repeated trials in our system for AR devices we employ an edge or cloud server and additional AR devices (see Section 4). Moreover, we show that this uncertainty, i.e., the diversity of pose estimates, can be used to estimate the average pose error that results from a set of visual and inertial input data, and thereby identify environment regions likely to result in high pose error.

Pose estimates generated in repeated SLAM trials are not necessarily in the same coordinate frame, which means that standard precision metrics cannot be applied to these unaligned estimates directly. Therefore, to calculate pose estimate diversity across a set of repeated trials, we perform pairwise trajectory evaluations (which include the alignment step) on the trajectory estimates obtained from those trials, for all possible trial pairs, such that for  $n$  trials we perform  $n(n-1)/2$  evaluations. We use  $B$  to denote this set of  $n$  trajectory estimates. We denote each trajectory estimate as  $\mathbf{b}_i = (b_1^i, b_2^i, \dots, b_S^i) \in B$ , where  $i$  ranges from 1 to  $n$ , and  $S$  represents the total number of sub-trajectories within a trajectory estimate. In the trajectory evaluations we calculate *relative pose error*, a measure of local trajectory consistency [39, 73], because this allows us to associate pose error with sub-trajectories, smaller portions of the visual and inertial input data, and hence localized areas of the environment. More formally, we run trajectory evaluation  $E$  on all possible pairs  $\{\mathbf{b}_i, \mathbf{b}_j\}$  of trajectory estimates in  $B$ , and obtain the set  $e_B$  of pose errors, i.e.,  $e_B = \{E(\mathbf{b}_i, \mathbf{b}_j) | \mathbf{b}_i, \mathbf{b}_j \in B, i \neq j\}$ . For each sub-trajectory we capture translational pose error, on the basis that errors in virtual object position are more noticeable than orientation errors, but this method could equally be applied to rotational pose error. Thus, for each sub-trajectory we have  $n(n-1)/2$  translational pose errors which represent pose output diversity. Rewriting  $E(\mathbf{b}_i, \mathbf{b}_j)$  as  $E(\mathbf{b}_i, \mathbf{b}_j) := (e_1^{i,j}, e_2^{i,j}, \dots, e_S^{i,j})$ , for the  $k$ -th sub-trajectory, these  $n(n-1)/2$  pose errors constitute the set  $\{e_k^{i,j} | \mathbf{b}_i, \mathbf{b}_j \in B, i \neq j\}$ .

To obtain our final pose error estimate for each sub-trajectory, we then apply a statistical measure  $M$  to characterize pose output diversity. When evaluating our method (Section 3.3) we test various statistical measures as methods of estimating pose error from diversity, which we term *uncertainty calculation settings*. These include the mean, median, trimmed means (the mean calculated after a predefined percentage of the smallest and largest values are removed from the data), and the mean after outliers are removed.

Formally, applying our uncertainty propagation-based method to trajectory estimates in  $B$ , we derive the pose error estimate  $U_{B,k}$  for the  $k$ -th sub-trajectory as  $U_{B,k} = M(\{e_k^{i,j} | \mathbf{b}_i, \mathbf{b}_j \in B, i \neq j\})$ . We then posit that for a set of trajectory estimates  $A$ , which is disjoint from  $B$  and obtained from the same inputs as  $B$ , the average pose error  $\bar{e}_A$  can be estimated as  $\bar{e}_A = (U_{B,1}, U_{B,2}, \dots, U_{B,S})$ . An example of the results we obtain using this method is shown in Figure 2b, which shows the average pose error from 100 trials of the SenseTime A1 sequence using ORB-SLAM3 (labeled ‘Ground truth’) along with our estimate, calculated using 100 separate trials and a 30% trimmed mean. To convert numerical pose error estimates to a form easily

interpretable by AR users, one can classify them by error magnitude, e.g., as low, medium or high error for each sub-trajectory – we apply this type of classifier in Section 3.3 to evaluate the performance of our pose error estimation method.

### 3.3 Pose Error Estimation Evaluation

#### 3.3.1 VI-SLAM Datasets

To the best of our knowledge there is no existing method for estimating relative pose error without ground truth pose measurements. Therefore, we evaluate our estimates against pose error calculated using ground truth pose, on 30 sequences from four VI-SLAM datasets. We select two widely-used datasets with motion patterns representative of handheld AR devices: from TUM VI [62] we use all six ‘room’ sequences (room1-6), and from SenseTime we use all eight ‘A’ sequences (A0-7). We then increase the diversity of visual environments using our recent method for combining inertial data from existing SLAM datasets with visual data from virtual environments [59]. We create two new VI-SLAM datasets using this method: ‘Hall’ and ‘LivingRoom’. For Hall we generate eight sequences which combine the SenseTime inertial data with visual data from a sample virtual environment in Unity’s High Definition Rendering Pipeline (HDRP) [65], a large 11m×7m×4m space with multiple different light sources, reflective surfaces and a curved wall. For LivingRoom we generate eight sequences using the SenseTime trajectories and a virtual environment we created in Unity’s HDRP, an 8m×6m×4m room containing both highly textured (e.g., a painting, soft furnishings) and featureless regions (a blank wall). Our Hall and LivingRoom datasets are publicly available to download at <https://github.com/timscargill/SiTAR/>.

#### 3.3.2 Evaluation Setup

For our main evaluation we run all 30 sequences with a current state-of-the-art VI-SLAM algorithm, ORB-SLAM3 [8] (monocular version, default settings). For an additional comparison we run the TUM VI sequences with another state-of-the-art VI-SLAM algorithm, VINS-Mono [52] (with a RANSAC threshold of 5px, which provides a good balance between accuracy and precision [56]). In our main evaluation we perform, for each sequence, 100 trials to calculate average ground truth-based pose error and 100 separate trials which we use to generate our pose error estimates. In our evaluation with VINS-Mono we perform 10 trials for ground truth-based error and 10 separate trials for estimation. We use a desktop computer (Intel i7-9700K CPU, Nvidia GeForce RTX 2060 GPU), but run SLAM on a virtual machine with 4 CPUs and 8GB RAM, representative of the computational resources of a mobile AR device. We perform trajectory evaluations using the evo Python package [31], with 10-frame sub-trajectories. Our testing indicated that 10 frames is a good balance between limiting visual input data to one environment region while maintaining sufficient error variance for informative results; the selection of optimal sub-trajectory lengths for different environments is an interesting topic for future work.

We test our estimation method with seven uncertainty calculation settings: the *mean*, *median*, trimmed means at 10%, 20%, 30%, 40% (*trim\_mean\_10*, *trim\_mean\_20*, *trim\_mean\_30*, *trim\_mean\_40*), and the mean after removing outliers defined by the interquartile range (*mean\_inliers*). For each setting we record results when the first 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 trials are used to calculate estimates. We evaluate our estimates with the performance of three 3-class error classifiers, each with different class boundaries: 5cm & 10cm; 2cm & 10cm; 2cm & 5cm. We choose these magnitudes because when applied to the ground truth data, they produce the imbalanced distribution we desire in our outputs; *to provide actionable information, our system should present a small number of sub-trajectories with especially high pose error, rather than an even distribution across all classes*. We quantify estimation performance using average accuracy and F1 score across all sequences.



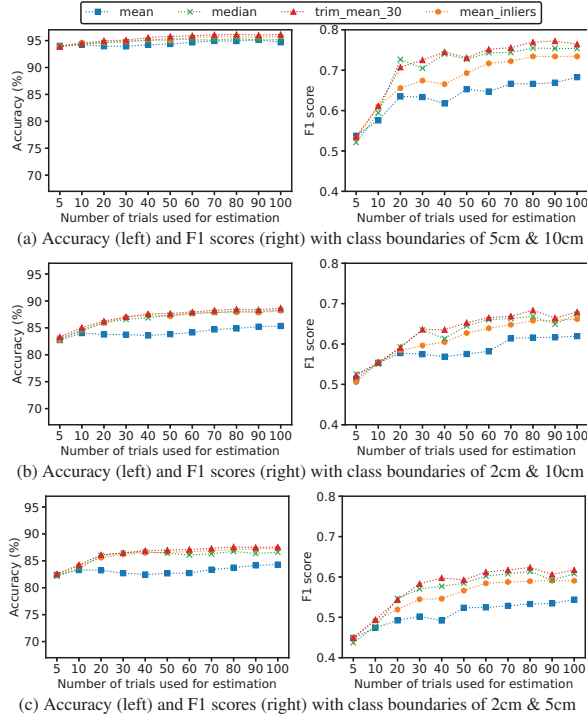


Figure 3: Performance of our uncertainty-based pose error estimation method, using different settings (see legend) and different numbers of trials. Applying outlier removal when calculating estimates (e.g., *trim\_mean\_30*) outperformed the standard *mean*.

### 3.3.3 Evaluation Results

Results for our main evaluation are shown in Figure 3; for each class boundary setting (Figure 3a: 5cm & 10cm, Figure 3b: 2cm & 10cm, Figure 3c: 2cm & 5cm) we plot accuracy (left) and F1 score (right) with different numbers of trials used for estimation. Each line plot represents an uncertainty calculation setting (see legend) from Section 3.2; for readability we only include the best-performing trimmed mean, omitting *trim\_mean\_10* (similar results to *mean\_inliers*), *trim\_mean\_20* (similar to *trim\_mean\_30*), and *trim\_mean\_40* (similar to *median*). Our estimation method achieves a peak accuracy of 96.1% (*trim\_mean\_30*, 80 trials), and a peak F1 score of 0.77 (*trim\_mean\_30*, 90 trials), both for the class boundary setting of 5cm & 10cm. Accuracy and F1 scores were lower for the other more challenging class boundary settings, but performance is still sufficient to provide a useful output. Higher accuracy and F1 scores were generally achieved as more trials were used for estimation, though there was a drop in F1 scores at 40 trials with the *mean*, due to large outliers between trials 31 and 40 causing over-prediction of error for three sequences. 30 trials resulted in comparable performance to 100 for all class boundary settings (with *trim\_mean\_30* the maximum accuracy drop was 1.6%, and the maximum drop in F1 score was 0.04), indicating this may provide a good balance between estimation performance and latency overhead (see Section 4.3).

When we examine the performance of different uncertainty calculation settings, *trim\_mean\_30* achieves the highest accuracy and F1 score for all class boundary settings. The *median* (essentially trimming at 50%) yields competitive performance, but we lose some valuable information. *Mean\_inliers* removes only a small percentage of outliers, and F1 scores suffer as a result. Notably the *mean*, which includes all error values, performs the worst of all settings with more than five trials. For example, when estimating using 40 trials, with class boundaries of 5cm & 10cm, *trim\_mean\_30* achieves an accuracy of 95.6% and an F1 score of 0.75, while the *mean* achieves 94.2% and 0.62 respectively. This highlights the benefit of outlier

Table 1: Mean specificity and sensitivity in our pose error estimation evaluation (100 trials), for each class boundary setting (5cm & 10cm, 2cm & 10cm, 2cm & 5cm) and uncertainty calculation setting.

| Setting             | Sensitivity |      |      | Specificity |      |      |
|---------------------|-------------|------|------|-------------|------|------|
|                     | 5-10        | 2-10 | 2-5  | 5-10        | 2-10 | 2-5  |
| <i>mean</i>         | 0.87        | 0.75 | 0.72 | 0.88        | 0.80 | 0.82 |
| <i>median</i>       | 0.86        | 0.70 | 0.66 | 0.81        | 0.72 | 0.75 |
| <i>trim_mean_30</i> | 0.87        | 0.70 | 0.67 | 0.83        | 0.73 | 0.76 |
| <i>mean_inliers</i> | 0.87        | 0.73 | 0.69 | 0.84        | 0.76 | 0.79 |

removal, especially when using larger numbers of trials.

Interestingly, this performance boost from outlier removal is not apparent in the sensitivity and specificity values for each uncertainty calculation and class boundary setting, shown in Table 1 (calculated individually for each class then averaged). In fact, sensitivity and specificity are greater for the *mean* because they do not take into account the prevalence of different classes [21]. For sequences with a small number of high error instances the *mean* results in several false positives for the high error class, but this does not lower specificity greatly, due to the large number of true negatives. However, if outlier-removing methods such as *trim\_mean\_30* fail to detect the small number of true positives, this results in low sensitivity for the high error class. We are more likely to detect all high error regions with the *mean*, but it is also more likely to result in warnings for ‘good’ regions, which would reduce the usefulness of our solution.

In contrast, in our evaluation with the VINS-Mono the best-performing uncertainty calculation setting was the *mean*; here our estimation method achieved an average accuracy of 99.4% and an average F1 score of 0.70 for class boundaries of 5cm and 10cm, 95.2% and 0.57 respectively for class boundaries of 2cm and 10cm, and 95.0% and 0.52 respectively for class boundaries of 2cm and 5cm. This difference in optimal uncertainty calculation setting for VINS-Mono is due to different pose error distributions across multiple trials compared to ORB-SLAM3, and suggests that our pose error estimation method may benefit from being configured for the specific VI-SLAM algorithm employed on an AR platform.

### 3.3.4 Evaluation Discussion

In a statistical context, research has demonstrated the disproportionate impact of outliers on analyses that rely on the sample mean and variance; for example, outliers can undermine the efficacy of parametric tests such as *t*-tests and *F*-tests [42]. The benefit of outlier removal increases when estimating pose error through uncertainty propagation because of the method used to calculate pose estimate uncertainty, pairwise trajectory evaluations. This is clear when we examine the relative contribution of each trial by calculating the ratio of error values that a single trial impacts to the total number of error values. For the ground truth-based data this is  $1/n$ , where  $n$  is the total number of trials. However, for pairwise comparisons this is the number of other trials which one trial can be compared with,  $n-1$ , over the total number of trials  $n(n-1)/2$ , giving us  $2(n-1)/[n(n-1)] = 2/n$ . This increased influence of outliers in pairwise comparisons explains the need to apply statistical measures such as the trimmed mean to remove extreme error values.

Our pose error estimation method is predicated on the observation that inaccurate trajectories are wrong in their own way. We analyzed our estimation performance for all 30 sequences individually and found that while this holds true for the majority, there are three for which it does not, resulting in noticeably worse performance. For example, despite SenseTime A0 being a challenging sequence that results in high pose error [36], trajectory estimates are similar; here inaccurate trajectories are wrong in a consistent way, such that our uncertainty-based estimates are lower than the actual pose error. Interestingly, this is also the case for Hall A0 and LivingRoom A0, suggesting the possible influence of the type of trajectory or inertial data. The underlying reason for these inaccurate yet consistent trajectory estimates is a topic we will examine in future work.

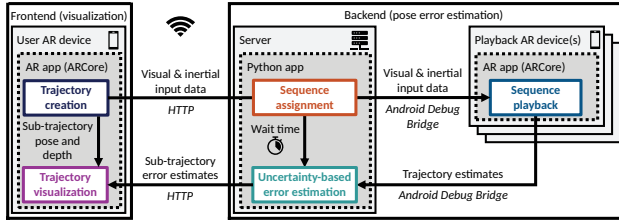


Figure 4: SiTAR system architecture for situated trajectory analysis on AR devices using uncertainty-based pose error estimation.

#### 4 SITUATED TRAJECTORY ANALYSIS SYSTEM: SiTAR

In this section we describe the motivation, design and architecture for SiTAR, our situated trajectory analysis system for AR that provides visualizations of uncertainty-based pose error estimates.

##### 4.1 Motivation

Traditionally, evaluations of device pose tracking accuracy are presented through numeric error values and 2D plots of ground truth and estimated trajectories (e.g., [8, 36, 62]). While essential for benchmarking tracking algorithms, neither communicates the environment regions visible through the device camera when errors occur. This is vital information for AR installation designers because pose error causes spatially registered virtual content in the current camera view to appear unstable to users. We require a solution that allows designers to identify the environment regions visible when pose tracking errors occur, so that they can either avoid placing virtual content there, or adjust environment properties to support better tracking.

To this end, we propose the use of *situated visualizations* for trajectory analysis, such that trajectory evaluation results are displayed relative to the real environment in which the trajectory was created. For example, a 3D trajectory can display where an AR device moved, while other types of visualization can be rendered at the locations visible in device camera views. These visualizations can then be semantically enriched with indicators of estimated pose error magnitude, such as color. Connecting trajectory evaluation results to the environment in this manner will allow AR designers and developers to quickly identify problematic environment regions, without relying on generic qualitative guidelines or resorting to laborious methods such as matching estimated error values with recorded camera images through timestamps. This empowers them to place virtual content with confidence, to rapidly implement and test environment optimization techniques, and opens up exciting possibilities for integrating predictive analytics into AR experience design.

##### 4.2 System Design

Our uncertainty propagation-based pose error estimation method (Section 3.2) requires the replaying of sequences (visual and inertial input data) multiple times to obtain multiple trajectory estimates; because ARCore provides native support for this through their Recording and Playback API [28] we describe the implementation of SiTAR for ARCore here, but this can be adapted for any AR platform which facilitates the recording and playback of sequences.

**System overview:** The SiTAR system architecture is shown in Figure 4. SiTAR consists of a frontend which handles trajectory creation and visualization on the user AR device, and a backend which provides pose error estimates to enrich trajectory visualizations. The system backend comprises a server and one or more playback AR devices; separate playback AR devices are employed because sequences must be replayed in real time, and using them frees up the user’s AR device in the meantime. Furthermore, the use of multiple playback AR devices enables the parallelization of sequence playback, so multiple trajectory estimates can be obtained in less time. Our system backend can be deployed using either an edge or cloud server, and with physical playback AR devices or

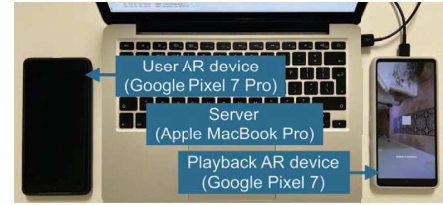


Figure 5: An example SiTAR hardware setup for ARCore, with an Apple MacBook Pro used as an edge server for the system backend.

Android emulators. Figure 5 shows an example hardware setup for SiTAR. Next, we describe the function of each system module in Figure 4, in the order corresponding to the main flow of data, i.e., Trajectory creation → Sequence assignment → Sequence playback → Uncertainty-based error estimation → Trajectory visualization.

**Trajectory creation:** The trajectory creation module allows users to create a trajectory (visual and inertial input data plus timestamped pose estimates) in an AR app on an AR device. We created this app in Unity 2021.3.14f1, using AR Foundation 4.2. Users start a trajectory with a UI button press, model AR user device movements, then end the trajectory with a button press. During the trajectory we capture visual and inertial input data using the ARCore Recording and Playback API, which saves the data in .mp4 format. The estimated position and orientation of the AR device are logged every five camera frames, such that the endpoints and midpoints of each 10-frame sub-trajectory are captured. To inform the rendering of certain visualizations (see Section 5.1) we also record depth estimates using the ARCore Depth API [26], at the sub-trajectory midpoints. Once a trajectory has been created, the visual and inertial input data are transmitted to the server via an HTTP POST request.

**Sequence assignment:** The sequence assignment module is implemented in a Python app on the server, built using the FastAPI [22] framework. This module handles an HTTP POST request containing the visual and inertial input data – a *sequence* – for a created trajectory, and saves these data to local storage. Based on the number and status of connected playback AR devices and the total number of trials to be performed, it assigns playbacks to each connected device. Visual and inertial input data are transferred to local storage on each of those AR devices via an Android Debug Bridge ‘push’ command. Finally, the module calculates a *wait time* based on when playback will complete (known because sequences must be run in real time).

**Sequence playback:** The sequence playback module replays the virtual and inertial input data on the playback AR device(s) the number of times specified by the sequence assignment module, and outputs a trajectory estimate for each iteration. We implemented this module in an AR app, created in Unity 2021.3.14f1, using AR Foundation 4.2. The app is installed on each playback AR device that will be used to replay sequences. Throughout sequence playback the estimated device pose is logged every frame, and the final trajectory estimates are saved to local storage after each iteration.

**Uncertainty-based error estimation:** The uncertainty-based error estimation module calculates error estimates for each sub-trajectory from the trajectory estimates generated by the sequence playback module(s). It is implemented on the server, within the same Python app as the sequence scheduling module. After using the *wait time* provided by the sequence assignment module to wait for sequence playback to complete, this module transfers trajectory estimates from the playback AR device(s) to the server via an Android Debug Bridge ‘pull’ command. It then calculates pose error estimates for each sub-trajectory using the method described in Section 3.2, with trajectory evaluations performed using the evo Python package [31]. Finally, the set of sub-trajectory error estimates (with associated timestamps) is transmitted to the user AR device via HTTP.

**Trajectory visualization:** Once a trajectory is created, the trajectory visualization module renders the initial trajectory estimate. The user AR device positions recorded at the endpoints and mid-

Table 2: Mean latency of the uncertainty-based estimation module.

| Sequence (duration)        | # of trials used for estimation |       |       |        |         |
|----------------------------|---------------------------------|-------|-------|--------|---------|
|                            | 5                               | 10    | 20    | 50     | 100     |
| <b>TUM VI room1 (141s)</b> | 4.5s                            | 18.9s | 78.9s | 539.2s | 2809.3s |
| <b>SenseTime A3 (45s)</b>  | 1.9s                            | 8.0s  | 32.7s | 229.4s | 1275.2s |
| <b>Ours (15s)</b>          | 0.5s                            | 1.9s  | 8.1s  | 67.7s  | 525.4s  |

points of each sub-trajectory define the endpoints of cylinders and the position of connecting spheres (diameters of 0.3m) showing the trajectory path. The AR device orientations recorded at each midpoint are used to render camera frustums at the sub-trajectory midpoints immediately following 0.5m increments on the trajectory – this indicates which direction the camera was facing at this point in the trajectory (a possible future enhancement would be to take into account rotation over the entire sub-trajectory when rendering frustums). Once sub-trajectory error estimates are obtained from the uncertainty-based error estimation module, they are loaded into a dictionary. The trajectory visualization module queries this dictionary to fetch estimated error for each timestamped sub-trajectory, and coloring or annotations can be applied to each rendered sub-trajectory based on these error values. Depth estimates from the trajectory creation module can also be used to inform the rendering of other types of visualization, as described in Section 5.1.

### 4.3 System Latency Characterization

We identify three potential latency bottlenecks for SiTAR, depending on the system configuration. First, the most obvious latency source is the *sequence playback module*, due to the need to replay sequences in real time. Sequence playback latency  $L_p$  can be approximated by  $L_p = d \times \lceil T/D \rceil$ , where  $d$  is the duration of the trajectory (sequence) created by the user,  $T$  is the number of trials, and  $D$  is the number of playback AR devices. Thus we can greatly reduce the playback latency by increasing the number of playback AR devices, but at the minimum it will be the trajectory duration.

Second, the data transmission step that incurs the greatest latency is the *wireless transfer of visual and inertial input data from the user AR device to the server*. In our testing the average file size of a sequence recorded with the ARCore Recording and Playback API is approximately 1–1.2MB per second of recorded data; assuming typical transmission speeds, transmission latency to either an edge or cloud server will usually be negligible compared to playback latency. For example, in our in-the-wild study (Section 5) the mean latency of transferring sequences to an edge server was 2.9s.

Third, we consider the latency of the *uncertainty-based error estimation module* on the server, largely determined by the number of trials used for estimation and the length of estimated trajectories. Recalling from Section 3.2 that for  $n$  trials we perform  $n(n-1)/2$  trajectory evaluations, latency grows substantially with both the number of trials and trajectory length, because error must be calculated for each sub-trajectory in each evaluation. We measure mean latency (10 trials) for a long sequence, TUM VI room1, a medium-length sequence, SenseTime A3, and our own short sequence recorded using ARCore, when running trajectory evaluations on a desktop computer (Intel i7-9700K CPU, Nvidia GeForce RTX 2060 GPU); our results are shown in Table 2. We note the modest estimation performance gains achieved with e.g., 50 rather than 20 trials (Figure 3) may not justify the latency overhead in many scenarios.

We also measure the *end-to-end latency of SiTAR* when the back-end is deployed using an edge server, with the hardware shown in Figure 5. These measurements are from 45 trials we conducted as part of an in-the-wild study (Section 5), across various residential and organizational local area networks; trajectory duration was approximately 15s, five trials were used for pose error estimation, the uncertainty calculation setting was a trimmed mean at 20%, and we used one playback AR device. Mean end-to-end latency was 89.1s, with a standard deviation of 12.9s and a range of 60.3–132.7s.

## 5 VISUALIZATIONS FOR SITUATED TRAJECTORY ANALYSIS

In this section we present three possible visualization techniques for situated trajectory analysis, and evaluate them through an in-the-wild user study. Rather than comparing user preferences in a fully controlled lab study, our goal here is to investigate the challenges presented by the diverse environments in which situated trajectory analysis will be conducted, and thereby inform future techniques.

### 5.1 Visualization Techniques

First, we detail the three visualization techniques we created and implemented. These visualizations apply different types of semantic enrichment to device trajectories to convey estimated pose error magnitude and problematic environment regions to the user.

**Trajectory-only:** Here we apply pose error-based semantic enrichment through coloring of the originally rendered trajectory. Once the pose error values are obtained for each sub-trajectory, we apply different colors to the cylinders and spheres used to render that sub-trajectory. For example, for the three-class classifier we developed in Section 3.3, which categorizes sub-trajectories into high, medium and low estimated error, we render the sub-trajectories associated with high error in red, medium error in yellow, and low error in the original blue. Any camera frustums rendered within a sub-trajectory are also rendered in the appropriate color. Examples of our trajectory-only visualization are shown in Figure 1a.

**Trajectory + exclamation points:** This technique extends our trajectory-only technique by rendering 3D exclamation points, a symbol widely associated with a warning [18], in problematic environment regions. To this end, we calculate the point on an environment surface that the user AR device camera is facing in the middle of each sub-trajectory. We experimented with ray casting against the environment map generated by the VI-SLAM algorithm; however, on devices without a depth sensor there are often large gaps in this map, particularly in the problematic regions (e.g., blank walls) we wish to detect [58]. Instead, we leverage the depth map from the ARCore Depth API to obtain the distance from the camera to the environment. Combined with the device pose, this gives us the 3D position on the surface where we render a virtual object [17]. We check if an exclamation point was previously rendered within a Euclidean distance of 0.5m (to avoid clusters of exclamation points), and if not, render a 3D exclamation point (width=5cm, height=30cm, depth=5cm), coloring it to match the sub-trajectory. Examples of our trajectory + exclamation points technique are shown in Figure 1b.

**Trajectory + warning signs:** This technique also extends our trajectory-only technique, but instead of rendering exclamation points on problematic environment surfaces, we render a triangular 2D warning sign containing an exclamation point, a widely known symbol which when employed in heads-up displays for driver assistance was associated with low driver response times [14]. Not only do we render the warning signs at the position defined by the environment surface, but we also match the orientation of the sign to that surface, so it appears, for example, pasted onto a wall or table. We again leverage the depth map generated by the ARCore Depth API, but this time we obtain both the depth magnitude and normal estimate, which combined with the camera pose provides the required 6D pose. As with exclamation points, we check that a warning sign was not previously rendered within a Euclidean distance of 0.5m, then render a 2D warning sign (width=30cm, height=30cm), and color it to match the sub-trajectory. Examples of our trajectory + warning signs technique are shown in Figure 1c.

### 5.2 Preliminary Testing

After integrating our visualization techniques into the SiTAR trajectory visualization module (Section 4.2), we performed approximately 50 trials in five environments, including lab, office and resi-



dential spaces. We qualitatively assessed performance by comparing the problematic regions SiTAR detected with expert judgments of visual input data likely to result in high pose error (based on both qualitative statements in previous works, e.g., [8, 36], and our experience). For example, we expected the system to identify featureless regions such as blank walls and powered-off TVs, poorly-lit areas, and reflective surfaces such as glass; as shown in Figure 1, these are the types of regions SiTAR highlighted. This consistency of performance with our earlier testing (Section 3.3) is logical, as to the best of our knowledge the VI-SLAM algorithms employed by commercial AR platforms share many similarities with the open-source algorithm on which we developed our estimation method.

Importantly, this testing indicated that in the vast majority of cases pose error magnitude is not so great as to render visualizations away from the intended problematic regions, such that the guidance they provide is still reliable. This was supported by the data from our in-the-wild study (Section 5.3); the average sub-trajectory error for the 45 trajectories ranged from 0.5cm to 24.9cm, with a mean of 3.7cm. These results emphasize the presence of noticeable errors, but the magnitudes observed mean that visualizations should still be rendered in the same environment region. To warn users when error is high enough to potentially affect the reliability of situated visualizations, we added additional UI guidance on the SiTAR user AR device when average estimated pose error exceeds 20cm.

Based on our testing, we also made two small adjustments to the SiTAR output to support our user study. First, to control task length we limit the number of problematic environment regions identified to three. Second, to maximize the probability that at least three regions are found (and avoid having to repeat trials), we search sub-trajectory pose error estimates for magnitudes greater than or equal to 10cm, 5cm, 2cm, then 1cm, until three sub-trajectories are found.

### 5.3 Visualization Experiments Design

To evaluate our three visualizations, we conducted an IRB-approved user study. As our goal was to investigate challenges associated with deploying different visualization techniques for situated trajectory analysis in diverse environments, we opted for an in-the-wild study, in which participants used SiTAR in a variety of different locations. We chose a within-subject design in which each participant experienced all three visualizations, presented in randomized order.

**Participants and locations:** We recruited 15 participants (9 female, 6 male; aged 18 to 44; normal or corrected-to-normal eyesight) from our personal and professional networks. Our study covers 13 diverse environments, including a wildlife center, a medical simulation space, a library, a botanical garden, a museum, a classroom, plus multiple residential and office spaces. This included outdoor spaces and indoor spaces ranging from small offices ( $\approx 3\text{m} \times 4\text{m} \times 3\text{m}$ ) to large open lobbies ( $\approx 20\text{m} \times 8\text{m} \times 6\text{m}$ ). Examples of these environments are shown in Figure 1. One environment was used per participant, so that each participant experienced the three visualizations under similar conditions, but different areas and movement paths were used for each visualization to avoid learning effects.

**Task:** Given that our targeted use case for situated trajectory analysis is the identification of problematic environment regions, we define a task in which participants have to locate and capture images of three problematic environment regions conveyed by each visualization. This task is similar to the search task used in studies on locating out-of-view objects in AR (e.g., [30, 61, 68]). Prior to the task, users created a 15-second trajectory (timing guided by the study administrator) that included a view of multiple environment regions, then found a position from which they could view the whole trajectory. As soon as the error-based visualization was rendered, participants were notified with UI text and an audio effect that the task had started. During the task the participants were free to locate the three problematic environment regions identified in any order, and were

required to press a UI button to capture an image of each region. Once they had captured the three images participants were notified with UI text and an audio effect that the task had been completed.

**Dependent variables and operationalization:** We measured task performance using task completion time, the total amount of time taken to capture an image of each of the three problematic environment regions. We measured user experience using the User Experience Questionnaire (UEQ) [40], along with subjective rankings obtained from a post-experiment questionnaire. In this post-experiment questionnaire we also captured participants reasoning for these subjective rankings by asking them to justify their choices with advantages and disadvantages of the visualizations. We measured the workload associated with each visualization using the raw, unweighted NASA Task Load Index (NASA-TLX) [33].

**Apparatus:** For the user AR device in SiTAR we used a Google Pixel 7 Pro running Android 13 and ARCore v1.37, with the display set at full brightness; participants used this device to create trajectories, perform tasks, and complete questionnaires. We deployed our SiTAR system backend (the uncertainty-based pose error estimation) on an edge server, an Apple MacBook Pro laptop with a 2.6GHz dual-core Intel Core i5 processor and 8GB RAM, running macOS BigSur 11.7 and Python 3.8. We used one playback AR device, a Google Pixel 7 running Android 13 and ARCore v1.37.

**Procedure:** After introducing participants to the study we asked them to sign a consent form. They then completed a pre-experiment questionnaire, used to obtain data on demographics and AR experience. Next, we conducted a training phase, in which we introduced participants to the app and the three visualizations, provided a demonstration of trajectory creation and task completion, and allowed the participant to practice this until they were comfortable completing it independently. We then started the main study, in which participants conducted one trial with each visualization. After each trial, participants completed the UEQ and NASA-TLX for the visualization they had just experienced. At the end of the study the participants completed the post-experiment questionnaire.

### 5.4 Visualization Experiments Results

The results for our three visualization techniques are shown in Figures 6 and 7. Below we compare them for each of our dependent variables – task performance, user experience, and workload. For readability, we denote the trajectory-only visualization by  $V_T$ , trajectory + exclamation points by  $V_E$ , and trajectory + warning signs by  $V_W$ . To test for statistical significance we use Friedman’s ANOVA (suitable for our repeated measures design with three conditions, without assuming data is normally distributed), followed by post-hoc pairwise comparisons using the Nemenyi test. For statistical significance we assume  $\alpha = 0.05$ . Our comparisons should be interpreted in the context that this is an exploratory rather than a fully controlled study; the different environments in which participants experienced visualizations could affect them differently, and while we found no evidence of order effects, they are possible in this randomized rather than counterbalanced study. However, the following results provide insights into the effectiveness of our visualizations across diverse scenarios, informing the design of future visualizations and studies.

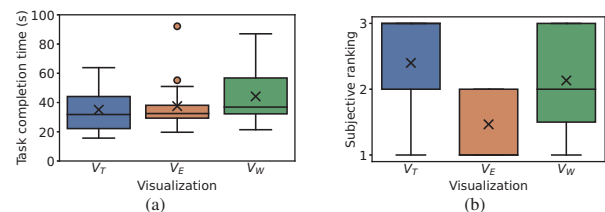


Figure 6: Results from our in-the-wild study of three visualizations for situated trajectory analysis: (a) performance, measured using task completion time; (b) subjective ranking (lower is better).

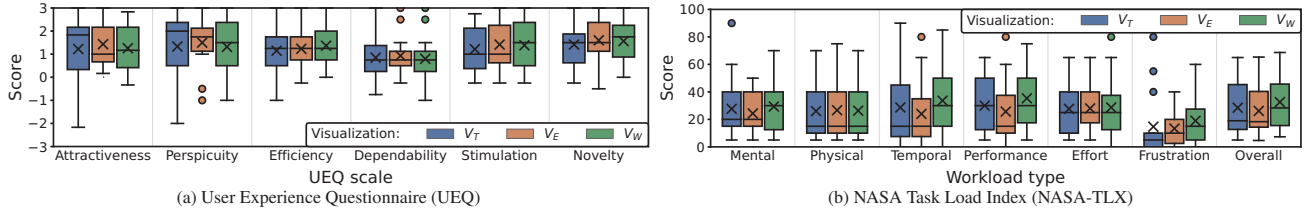


Figure 7: Results from our in-the-wild study of three visualizations for situated trajectory analysis: (a) user experience, measured using the user experience questionnaire [40]; (b) workload, measured using the NASA Task Load Index [33].

#### 5.4.1 Task Performance

For task completion time (Figure 6a), Friedman’s ANOVA revealed significant differences between conditions ( $\chi^2(2) = 6.533, p = 0.038$ ), and a post-hoc Nemenyi test showed task completion time was significantly greater for  $V_W$  than  $V_T$  ( $p = 0.046$ ,  $V_W$  mean = 44.2s,  $V_T$  mean = 35.1s). Our post-experiment survey revealed one possible reason, in that some participants indicated they found the warning signs hard to locate, one commenting that “the warning signs could be harder to determine where they were.” This may be due to the warning signs being 2D, and hence not visible from certain directions. Other participants noted that even when visible, warning signs did not convey the desired information effectively: “the orientation/location of the warning signs were confusing and it was not clear which signs aligned with which parts of the trajectory.”

Upon examining instances of high task completion times for  $V_T$  and  $V_W$ , we found they occurred either in large environments or regions with glass surfaces (e.g., windows, museum display cases). For example, in the case of the two outliers for  $V_E$ , at least one exclamation point was rendered  $> 6m$  away. These types of environments pose significant challenges for depth estimate-based rendering of virtual objects, because estimates are less accurate at greater distances and on reflective surfaces [72]. This in turn results in the inaccurate position or orientation of virtual objects; for example, an exclamation point may be rendered ‘outside’ of a glass window, or a warning sign at  $90^\circ$  to the actual surface of a museum display case. This degradation of visualizations motivates the development of techniques more robust to large environments and reflective surfaces, especially given their prevalence in modern architecture.

#### 5.4.2 User Experience

For subjective rankings (Figure 6b), Friedman’s ANOVA revealed significant differences between conditions ( $\chi^2(2) = 6.933, p = 0.031$ ), and a post-hoc Nemenyi test showed users ranked  $V_E$  significantly higher (lower subjective ranking score) than  $V_T$  ( $p = 0.029$ ,  $V_E$  mean = 1.5,  $V_T$  mean = 2.4). Participant justifications included “I liked the additional signal that indicated a problem area” and “Really hard to gauge without...symbols if you were pointing the right direction”, indicating that they liked the extra information that the exclamation points and warning signs provided. Participants who ranked warning signs lower than exclamation points often noted the problem of locating the signs or determining their orientation.

Friedman’s ANOVA did not reveal significant differences between conditions for any of the UEQ scales (Figure 7a), perhaps due to all visualizations having a prominent element, the trajectory, in common. We note that while all three visualizations obtained mean scores  $> 1$  on the attractiveness, perspicuity, efficiency, stimulation and novelty scales, participants rated them lower for dependability, with mean scores for all visualizations  $< 1$ . This indicates that there is work to be done on improving the perceived reliability of situated trajectory analysis, and ensuring users feel in control of interactions.

#### 5.4.3 Workload

For the NASA TLX results (Figure 7b), Friedman’s ANOVA revealed significant differences between conditions for overall workload ( $\chi^2(2) = 6.241, p = 0.044$ ), and a post-hoc Nemenyi test

showed overall workload was significantly higher for  $V_W$  than  $V_E$  ( $p = 0.046$ ,  $V_W$  mean = 32.6,  $V_E$  mean = 26.2). This is consistent with participant feedback on difficulty finding warning signs or that their orientation was confusing. While three participants scored  $V_T$  highly for frustration – comments included “Just the trajectory did not provide much information” and “Trajectory has insufficient data to help me understand my task” – all other participants scored  $V_T \leq 10$  for frustration, with the median frustration score for  $V_T$  lower than for  $V_E$  or  $V_W$ . This indicates that despite participants preferring visualizations which attached virtual objects to problematic regions, searching for those objects may be a source of frustration, and prompts the inclusion of techniques that make them more noticeable (e.g., the addition of motion in kineticons [32]), and to help users navigate to them when they are out of view (e.g., [30, 61, 68]).

## 6 CONCLUSION AND FUTURE WORK

In this paper we presented SiTAR, the first system that incorporates pose error estimates into situated trajectory analysis. To enable this, we developed the first VI-SLAM pose error estimation method based on uncertainty propagation, and demonstrated its efficacy on four VI-SLAM datasets. We integrated our pose error estimation method into our SiTAR system, which can be deployed on real AR devices with an edge or cloud server backend. We tested SiTAR in an in-the-wild study with 15 users in 13 diverse environments; this study revealed an overall preference for pose error visualizations which placed virtual objects in problematic environment regions, as well as how environment properties can impact these techniques.

SiTAR opens up a number of opportunities for future work. First, our visualization experiments in diverse environments prompt investigation into how techniques can be best adapted to suit different types of environments; this includes the visualizations themselves, sub-trajectory length, and the distance between virtual objects attached to environment regions. Second, there are a number of possible improvements to situated pose error visualizations; these include techniques to aid users further in determining the source of pose error (e.g., visualizations that show the entire spatial regions covered by input camera images, the addition of input data properties to trajectories), visualizations more robust to user pose, and methods of indicating which environment regions have been scanned. Backup methods of communicating problematic regions (when tracking error is so great as to make situated visualizations unreliable) should also be considered, e.g., displaying the input camera frames for a sub-trajectory. Finally, there is exciting scope for extending SiTAR to analyze trajectories from multiple users. For example, we envision recording user trajectories, evaluating them with the SiTAR backend offline, then generating persistent situated visualizations that combine error estimates from multiple trajectories. Administrators can then analyze these data at their convenience, to identify environment or content adjustments that will improve virtual object stability.

## ACKNOWLEDGMENTS

We thank our study participants for their assistance in this research. This work was supported in part by NSF grants CSR-1903136, CNS-1908051 and CNS-2112562, NSF CAREER Award IIS-2046072, a Meta Research Award and a CISCO Research Award.



## REFERENCES

- [1] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarekovic, and S. Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.
- [2] B. H. Albanna, I. F. Moawad, S. M. Moussa, and M. A. Sakr. Semantic trajectories: A survey from modeling to application. *Information Fusion and Geographic Information Systems (IF&GIS'2015) Deep Virtualization for Mobile GIS*, pp. 59–76, 2015.
- [3] I. Ali, B. Wan, and H. Zhang. Prediction of SLAM ATE using an ensemble learning regression model and 1-D global pooling of data characterization. *arXiv preprint arXiv:2303.00616*, 2023.
- [4] Apple. ARKit ARTrackingStateReason. <https://developer.apple.com/documentation/arkit/artrackingstate>, 2023.
- [5] N. Bressa, H. Korsgaard, A. Tabard, S. Houben, and J. Vermeulen. What's the situation with situated visualization? A survey and perspectives on situatedness. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):107–117, 2021.
- [6] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.
- [7] W. Büschel, A. Lehmann, and R. Dachselt. MIRIA: A mixed reality toolkit for the in-situ visualization and analysis of spatio-temporal interaction data. In *Proceedings of CHI*, 2021.
- [8] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial, and multi-map SLAM. *IEEE Transactions on Robotics*, pp. 1–17, 2021.
- [9] L. Carlone and S. Karaman. Attention and anticipation in fast visual-inertial navigation. *IEEE Transactions on Robotics*, 35(1):1–20, 2018.
- [10] J. Chang, Z. Lan, C. Cheng, and Y. Wei. Data uncertainty learning in face recognition. In *Proceedings of IEEE/CVF CVPR*, 2020.
- [11] L. Chen, K. Takashima, K. Fujita, and Y. Kitamura. PinpointFly: An egocentric position-control drone interface using mobile AR. In *Proceedings of CHI*, 2021.
- [12] Y. Chen, H. Inaltekin, and M. Gorlatova. AdaptSLAM: Edge-assisted adaptive SLAM with resource constraints via uncertainty minimization. In *Proceedings of IEEE INFOCOM*, 2023.
- [13] Y. Chen, L. Zhao, Y. Zhang, S. Huang, and G. Dissanayake. Anchor selection for SLAM based on graph topology and submodular optimization. *IEEE Transactions on Robotics*, 38(1):329–350, 2021.
- [14] S. Y. Cheng, A. Doshi, and M. M. Trivedi. Active heads-up display based speed compliance aid for driver assistance: A novel interface and comparative experimental studies. In *Proceedings of IEEE IV*, 2007.
- [15] S. Condino, B. Fida, M. Carbone, L. Cencenelli, G. Badiali, V. Ferrari, and F. Cutolo. Wearable augmented reality platform for aiding complex 3D trajectory tracing. *Sensors*, 20(6):1612, 2020.
- [16] S. Cortés, A. Solin, E. Rahtu, and J. Kannala. ADVIO: An authentic dataset for visual-inertial odometry. In *Proceedings of ECCV*, 2018.
- [17] R. Du, E. Turner, M. Dzitsiuk, L. Prasso, I. Duarte, J. Dourgarian, J. Afonso, J. Pascoal, J. Gladstone, N. Cruces, et al. DepthLab: Real-time 3D interaction with depth maps for mobile augmented reality. In *Proceedings of ACM UIST*, 2020.
- [18] J. Edworthy. *Warning Design: A Research Prospective*. CRC Press, 1996.
- [19] N. A. ElSayed, B. H. Thomas, K. Marriott, J. Piantadosi, and R. T. Smith. Situated analytics: Demonstrating immersive analytical tools with augmented reality. *Journal of Visual Languages & Computing*, 36:13–23, 2016.
- [20] S. Eom, D. Sykes, S. Rahimpour, and M. Gorlatova. NeuroLens: Augmented reality-based contextual guidance through surgical tool tracking in neurosurgery. In *Proceedings of IEEE ISMAR*, 2022.
- [21] B. J. Erickson and F. Kitamura. Magician's corner: 9. performance metrics for machine learning models. *Radiology: Artificial Intelligence*, 3(3):e200126, 2021.
- [22] FastAPI. FastAPI. <https://fastapi.tiangolo.com/>, 2023.
- [23] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [24] P. Fleck, A. S. Calepso, S. Hubenschmid, M. Sedlmair, and D. Schmalstieg. RagRug: A toolkit for situated analytics. *IEEE Transactions on Visualization and Computer Graphics*, 2022.
- [25] V. Gay-Bellile, P. Lothe, S. Bourgeois, E. Royer, and S. N. Collette. Augmented reality in large environments: Application to aided navigation in urban context. In *Proceedings of IEEE ISMAR*, 2010.
- [26] Google. ARCore - Depth. <https://developers.google.com/ar/develop/depth>, 2023.
- [27] Google. ARCore - Environment. <https://developers.google.com/ar/design/environment/definition>, 2023.
- [28] Google. ARCore - Recording and playback. <https://developers.google.com/ar/develop/recording-and-playback>, 2023.
- [29] Google. ARCore TrackingFailureReason. <https://developers.google.com/ar/reference/java/com/google/ar/core/TrackingFailureReason>, 2023.
- [30] U. Gruenfeld, A. E. Ali, S. Boll, and W. Heuten. Beyond halo and wedge: Visualizing out-of-view objects on head-mounted virtual and augmented reality devices. In *Proceedings of ACM Mobile HCI*, 2018.
- [31] M. Grupp. evo: Python package for the evaluation of odometry and SLAM. <https://github.com/MichaelGrupp/evo>, 2017.
- [32] C. Harrison, G. Hsieh, K. D. Willis, J. Forlizzi, and S. E. Hudson. Kineticons: Using iconographic motion in graphical user interface design. In *Proceedings of CHI*, 2011.
- [33] S. G. Hart and L. E. Staveland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in Psychology*, vol. 52, pp. 139–183. Elsevier, 1988.
- [34] D. Heißelmann, M. Franke, K. Rost, K. Wendt, T. Kistner, and C. Schwehn. Determination of measurement uncertainty by Monte Carlo simulation. In *Advanced Mathematical and Computational Tools in Metrology and Testing XI*, pp. 192–202. World Scientific, 2019.
- [35] S. Hubenschmid, J. Wieland, D. I. Fink, A. Batch, J. Zagermann, N. Elmquist, and H. Reiterer. ReLive: Bridging in-situ and ex-situ visual analytics for analyzing mixed reality user studies. In *Proceedings of CHI*, 2022.
- [36] L. Jinyu, Y. Bangbang, C. Danpeng, W. Nan, Z. Guofeng, and B. Hujun. Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality. *Virtual Reality & Intelligent Hardware*, 1(4):386–410, 2019.
- [37] M. Kasper, S. McGuire, and C. Heckman. A benchmark for visual-inertial odometry systems employing onboard illumination. In *Proceedings of IEEE/RSJ IROS*, 2019.
- [38] K. Khosoussi, M. Giamou, G. S. Sukhatme, S. Huang, G. Dissanayake, and J. P. How. Reliable graphs for SLAM. *The International Journal of Robotics Research*, 38(2-3):260–298, 2019.
- [39] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner. On measuring the accuracy of SLAM algorithms. *Autonomous Robots*, 27:387–407, 2009.
- [40] B. Laugwitz, T. Held, and M. Schrepp. Construction and evaluation of a user experience questionnaire. In *HCI and Usability for Education and Work*, pp. 63–76. Springer, 2008.
- [41] X. Li, Y. Dai, Y. Ge, J. Liu, Y. Shan, and L. Duan. Uncertainty modeling for out-of-distribution generalization. In *Proceedings of ICLR*, 2022.
- [42] H. Liao, Y. Li, and G. Brooks. Outlier impact and accommodation methods: Multiple comparisons of type I error rates. *Journal of Modern Applied Statistical Methods*, 15(1):23, 2016.
- [43] T. Lin, R. Singh, Y. Yang, C. Nobre, J. Beyer, M. A. Smith, and H. Pfister. Towards an understanding of situated AR visualization for basketball free-throw training. In *Proceedings of CHI*, 2021.
- [44] W. Luo, Z. Yu, R. Rzayev, M. Satkowski, S. Gumhold, M. McGinity, and R. Dachselt. PEARL: Physical environment based augmented reality lenses for in-situ human movement analysis. In *Proceedings of CHI*, 2023.
- [45] H. O. Madsen, S. Krenk, and N. C. Lind. *Methods of Structural Safety*. Courier Corporation, 2006.

- [46] T. Nöll, A. Pagani, and D. Stricker. Real-time camera pose estimation using correspondences with high outlier ratios. In *Proceedings of VISAPP*, 2010.
- [47] OptiTrack. OptiTrack. <https://optitrack.com/>, 2023.
- [48] F. Pankratz, A. Dippon, T. Coskun, and G. Klinker. User awareness of tracking uncertainties in AR navigation scenarios. In *Proceedings of IEEE ISMAR Adjunct*, 2013.
- [49] C. Parent, S. Spaccapietra, C. Renso, G. Andrienko, N. Andrienko, V. Bogorny, M. L. Damiani, A. Gkoulalas-Divanis, J. Macedo, N. Pelekis, et al. Semantic trajectories modeling and analysis. *ACM Computing Surveys (CSUR)*, 45(4):1–32, 2013.
- [50] A. Petrocchi, A. Kaintura, G. Avolio, D. Spina, T. Dhaene, A. Raffo, and D. M.-P. Schreurs. Measurement uncertainty propagation in transistor model parameters via polynomial chaos expansion. *IEEE Microwave and Wireless Components Letters*, 27(6):572–574, 2017.
- [51] M. Ping, X. Jia, C. Papadimitriou, X. Han, and C. Jiang. Statistics-based Bayesian modeling framework for uncertainty quantification and propagation. *Mechanical Systems and Signal Processing*, 174:109102, 2022.
- [52] T. Qin, P. Li, and S. Shen. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.
- [53] C. P. Quintero, S. Li, M. K. Pan, W. P. Chan, H. M. Van der Loos, and E. Croft. Robot programming through augmented trajectories in augmented reality. In *Proceedings of IEEE/RSJ IROS*, 2018.
- [54] R. Raguram, J.-M. Frahm, and M. Pollefeys. A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus. In *Proceedings of ECCV*, 2008.
- [55] X. Ran, C. Slocum, M. Gorlatova, and J. Chen. ShareAR: Communication-efficient multi-user mobile augmented reality. In *Proceedings of ACM HotNets*, 2019.
- [56] C. Riu, V. Nozick, P. Monasse, and J. Dehais. Classification performance of RANSAC algorithms with automatic threshold estimation. In *Proceedings of VISIGRAPP*, 2022.
- [57] D. Sacha, H. Senaratne, B. C. Kwon, G. Ellis, and D. A. Keim. The role of uncertainty, awareness, and trust in visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):240–249, 2015.
- [58] K. Sartipi, T. Do, T. Ke, K. Vuong, and S. I. Roumeliotis. Deep depth estimation from visual-inertial SLAM. In *Proceedings of IEEE/RSJ IROS*, 2020.
- [59] T. Scargill, Y. Chen, N. Marzen, and M. Gorlatova. Integrated design of augmented reality spaces using virtual environments. In *Proceedings of IEEE ISMAR*, 2022.
- [60] T. Scargill, G. Premsankar, J. Chen, and M. Gorlatova. Here to stay: A quantitative comparison of virtual object stability in markerless mobile AR. In *Proceedings of IEEE/ACM CPHS Workshop (co-located with CPS-IoT Week)*, 2022.
- [61] T. Schinke, N. Henze, and S. Boll. Visualization of off-screen objects in mobile augmented reality. In *Proceedings of ACM Mobile HCI*, 2010.
- [62] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stückler, and D. Cremers. The TUM VI benchmark for evaluating visual-inertial odometry. In *Proceedings of IEEE/RSJ IROS*, 2018.
- [63] S. Shin, A. Batch, P. W. Butcher, P. D. Ritsos, and N. Elmqvist. The reality of the situation: A survey of situated analytics. *IEEE Transactions on Visualization and Computer Graphics*, 2023.
- [64] B. H. Thomas, G. F. Welch, P. Dragicevic, N. Elmqvist, P. Irani, Y. Jansen, D. Schmalstieg, A. Tabard, N. A. ElSayed, R. T. Smith, et al. Situated analytics. *Immersive Analytics*, 11190:185–220, 2018.
- [65] Unity. High definition render pipeline. <https://unity.com/srp/High-Definition-Render-Pipeline>, 2023.
- [66] Vicon. Vicon. <https://www.vicon.com/>, 2023.
- [67] M. Walker, H. Hedayati, J. Lee, and D. Szafir. Communicating robot motion intent with augmented reality. In *Proceedings of ACM/IEEE HRI*, 2018.
- [68] J. Wieland, R. C. H. Garcia, H. Reiterer, and T. Feuchtnner. Arrow, bezier curve, or halos? – Comparing 3D out-of-view object visualization techniques for handheld augmented reality. In *Proceedings of IEEE ISMAR*, 2022.
- [69] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer. Semantic trajectories: Mobility data computation and annotation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 4(3):1–38, 2013.
- [70] A. Zakaria, F. B. Ismail, M. H. Lipu, and M. Hannan. Uncertainty models for stochastic optimization in renewable energy applications. *Renewable Energy*, 145:1543–1571, 2020.
- [71] J. Zhang, J. Yin, and R. Wang. Basic framework and main methods of uncertainty quantification. *Mathematical Problems in Engineering*, 2020:1–18, 2020.
- [72] Y. Zhang, T. Scargill, A. Vaishnav, G. Premsankar, M. Di Francesco, and M. Gorlatova. InDepth: Real-time depth inpainting for mobile augmented reality. In *Proceedings of ACM IMWUT*, 2022.
- [73] Z. Zhang and D. Scaramuzza. A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In *Proceedings of IEEE/RSJ IROS*, 2018.
- [74] S. Zhao, X. Xiao, Q. Wang, X. Zhang, W. Li, L. Soghier, and J. Hahn. An intelligent augmented reality training framework for neonatal endotracheal intubation. In *Proceedings of IEEE ISMAR*, 2020.
- [75] S. Zollmann, C. Hoppe, T. Langlotz, and G. Reitmayr. FlyAR: Augmented reality supported micro aerial vehicle navigation. *IEEE Transactions on Visualization and Computer Graphics*, 20(4):560–568, 2014.
- [76] D. Zuñiga-Noël, A. Jaenal, R. Gomez-Ojeda, and J. Gonzalez-Jimenez. The UMA-VI dataset: Visual-inertial odometry in low-textured and dynamic illumination environments. *The International Journal of Robotics Research*, 39(9):1052–1060, 2020.