SEESys: Online Pose Error Estimation System for Visual SLAM

Tianyi Hu
tianyi.hu@duke.edu
Duke University
Durham, NC, USA

Tim Scargill
ts352@duke.edu
Duke University
Durham, NC, USA

Fan Yang
fan.yang@duke.edu
Duke University
Durham, NC, USA

Ying Chen
ychen62@kennesaw.edu
Kennesaw State University
Kennesaw, GA, USA

Guohao Lan
g.lan@tudelft.nl
Delft University of Technology
Delft, the Netherlands

Maria Gorlatova
maria.gorlatova@duke.edu
Duke University
Durham, NC, USA

ABSTRACT

In this work, we introduce SEESys, *the first system to provide online pose error estimation for Simultaneous Localization and Mapping (SLAM)*. Unlike prior offline error estimation approaches, the SEESys framework efficiently collects real-time system features and delivers accurate pose error magnitude estimates with low latency. This enables *real-time quality-of-service information* for downstream applications. To achieve this goal, we develop a SLAM system run-time status monitor (RTS monitor) that performs feature collection with minimal overhead, along with a multi-modality attention-based Deep SLAM Error Estimator (DeepSEE) for error estimation. We train and evaluate SEESys using both public SLAM benchmarks and a diverse set of synthetic datasets, achieving an RMSE of 0.235 cm of pose error estimation, which is 15.8% lower than the baseline. Additionally, we conduct a case study showcasing SEESys in a real-world scenario, where it is applied to a real-time audio error advisory system for human operators of a SLAM-enabled device. The results demonstrate that SEESys provides error estimates with an average end-to-end latency of 37.3 ms, and the audio error advisory reduces pose tracking error by 25%.

CCS CONCEPTS

• **Human-centered computing** → *Ubiquitous and mobile computing systems and tools.*

KEYWORDS

SLAM, pose tracking, tracking error, error estimate, edge computing, deep learning

ACM Reference Format:
Tianyi Hu, Tim Scargill, Fan Yang, Ying Chen, Guohao Lan, and Maria Gorlatova. 2024. SEESys: Online Pose Error Estimation System for Visual SLAM. In *ACM Conference on Embedded Networked Sensor Systems (SenSys '24), November 4–7, 2024, Hangzhou, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3666025.3699341

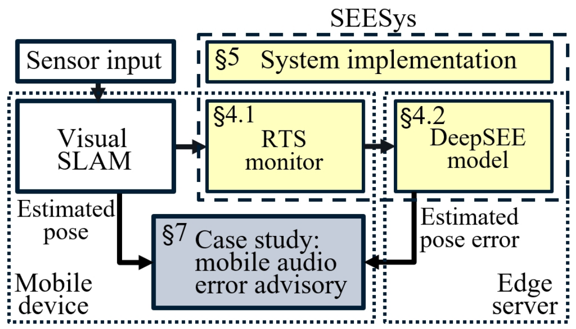

Figure 1: Overview of the SEESys system, including an RTS monitor for feature collection on a mobile device and the DeepSEE model for pose error estimation on an edge server. Additionally, we showcase a mobile audio error advisory as a potential application of our system.

1 INTRODUCTION

Simultaneous localization and mapping (SLAM) has become an essential component in various mobile systems, including augmented reality (AR) devices, mobile robots, and autonomous vehicles. These systems rely on the ability to quickly process data from onboard sensors to simultaneously map the environment and estimate their pose within it [8, 33]. However, even state-of-the-art solutions, such as ORBSLAM3 [10] and VINS-Mono [47], frequently encounter pose estimation errors, which represent the discrepancies between estimated poses and ground truth poses. These errors are often caused by challenging environmental conditions or rapid device motion [30, 45]. The magnitude of these errors can vary significantly, ranging from a few millimeters to over one meter within a short period [6, 10, 30, 47]. Consequently, the quality of downstream applications, such as virtual object positioning in AR [30, 55] and robot navigation [15], can degrade suddenly and unexpectedly.

Real-time awareness of pose error is therefore crucial for monitoring and maintaining the quality of service and has numerous potential applications. For example, it could alert applications or users when the current pose data or any outputs derived from it are unreliable. Additionally, knowing the error could guide SLAM systems or human operators to take corrective actions, such as reducing movement speed or repositioning to more visually favorable environments [30]. Ultimately, this information could enable adaptive SLAM systems, which adjust SLAM parameters based on the estimated error to reduce errors or optimize resource consumption, thereby enhancing the performance of downstream applications.

However, prior work relies on external systems to provide ground truth trajectories as references for error calculation [30, 58, 80]. These systems, such as motion capture systems, require significant time and effort for setup and calibration in new environments, making them impractical for out-of-the-box use.

While recent advances have made progress in estimating pose error [2, 54], they do not provide this information in real time, which limits their ability to support timely interventions required by many applications. Therefore, there is a need for an online method for real-time pose error estimation.

The primary challenges in online pose error estimation are two-fold. First, pose error arises from the complex interaction between visual environment properties and device motions, which together affect the magnitude of the error. These errors can stem from various sources and manifest at different stages within the SLAM pipeline. Second, extracting features from the SLAM pipeline and performing error estimation introduces computations overhead. Deploying such a system on resource-constrained mobile devices while achieving real-time estimation requires a customized pipeline to minimize the costs associated with feature collection, model inference, and data transmission. Additionally, a robust estimation model is needed to handle multi-modal inputs and be trained on a broader range of environments than those available in existing visual SLAM datasets.

To address these challenges, we present SEESys (**S**LAM **E**rror **E**stimation **Sys**tem), *the first system designed for online SLAM pose error estimation.* To achieve better error estimation accuracy over existing offline methods (e.g., [2]) while minimizing overhead on mobile devices, SEESys leverages a tailor-made edge computing architecture. This architecture incorporates several key strategies: a lightweight on-device feature collection method utilizing multi-threading and data compression, and offloading the multi-modal deep learning model for error estimation to the edge server. These choices optimize performance while reducing latency. This work focuses on developing SEESys for visual SLAM, but it can easily be extended to other variants, such as visual-inertial SLAM. The real-time error estimates provided by our solution offer an invaluable quality-of-service metric applicable to multiple application use cases without the need for external systems to obtain ground truth trajectories. We demonstrate one use case with a practical case study: implementing an audio error advisory via the *sonification* [26] of pose error estimates, which alerts human operators when a reduction in mobile device speed is needed to improve pose tracking. An annotated demo video showcasing this use case is available online[1].

Our main contributions are as follows:

- We design SEESys, the first system for estimating SLAM pose error *in real time* without access to ground truth pose; our system collects features from both sensor input and the internal SLAM pipeline at runtime, using a multi-modality attention-based model to estimate pose error.
- We implement SEESys on real mobile devices and evaluate its pose error estimation accuracy compared to the state-of-the-art offline method (which does not provide error estimates in real time); we achieve an RMSE of 0.235 cm, outperforming this baseline by 15.8%. Additionally, SEESys delivers pose error estimates with an average end-to-end latency of 37.3 ms in real-world scenarios.
- We present a case study demonstrating the use of SEESys for an audio error advisory; we are the first to propose and implement the sonification of SLAM pose error. Through an IRB-approved user study with 30 participants, we show that using SEESys to generate audio advisories helps guide human operators of SLAM-enabled devices, reducing pose tracking errors by 25%.

The remainder of the paper is organized as follows: Section 2 covers related work, while Section 3 presents relevant background on visual SLAM. Section 4 discusses system design of SEESys, and Section 5 details its implementation. In Section 6, we evaluate SEESys, followed by a case study in Section 7 demonstrating a practical use case. We discuss the limitations and future directions in Section 8, and conclude in Section 9. The code for SEESys, along with datasets we create for our implementation (Section 5.3) is publicly available on GitHub[2].

2 RELATED WORK

SLAM evaluation and benchmarks. Traditional SLAM evaluations (e.g., [6, 30, 58, 62]) rely on ground truth pose measurements obtained from motion capture systems (e.g., [46, 68]). However, the laborious setup and calibration required by these systems make this approach impractical for scaling to the diverse environments in which SLAM is deployed. To address this limitation, we develop a deep learning model to estimate pose error from input sensor data and the internal SLAM pipeline. Public SLAM benchmarks (e.g., [6, 13, 22, 30, 32, 58, 65, 81]) provide labeled data that could be used to train the estimation model; we select the SenseTime dataset [30] because it is representative of various motion types with handheld devices. Due to the limited range of visual conditions covered in real-world datasets like SenseTime, we augment our training data with synthetic data generated in diverse virtual environments. While conceptually similar to the TartanAir dataset [70], our synthetic datasets are generated using real handheld device trajectories rather than simulated aerial trajectories. This diversity in training data allows us to build a more robust deep learning model for pose error estimation.

SLAM pose error estimation. Several studies have proposed methods for estimating pose error without access to ground truth poses. For visual localization (rather than visual SLAM), Ferranti et al. [21] estimate pose confidence using a logistic regression model with features such as the number and distribution of inliers. Scargill et al. [54] estimate SLAM pose error through uncertainty in estimated poses, calculated via running repeated trials with the same input data on an edge server. While this replay-based method allows for pose error estimation without ground truth poses, the need for repeated trials prevents online error estimations. Although it is offline, it allows users to inspect which part of their trajectory has high pose error so that later they can adjust the environment settings accordingly for better QoS, for example, adding more light and visual features at the place where the estimated pose error is high. Other works [11, 12, 50] have explored SLAM uncertainty quantification of pose estimates (rather than pose tracking error magnitude) using

[1]https://tinyurl.com/SEESysVideo

[2]https://github.com/Duke-I3T-Lab/SEESys

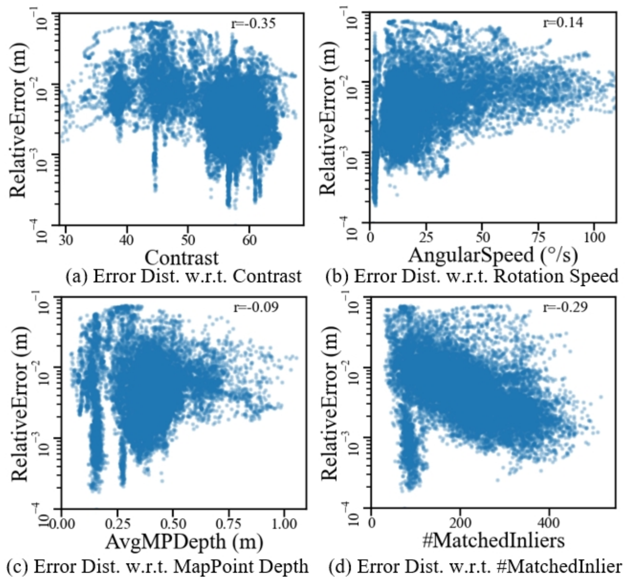

Figure 2: Pose error distribution with respect to factors affecting SLAM tracking performance and their Pearson correlation coefficients on the SenseTime [30] dataset.

the maximum likelihood estimator covariance matrix, extracting information from the SLAM pipeline's optimization of the environment map. The work closest to ours, Ali et al. [2], demonstrated that a random forest regression model based on input sensor data characteristics can be used to estimate the average pose error over an entire trajectory. However, this is an offline method that begins with a pre-collected dataset, lacking the ability to integrate with a SLAM system for real-time error estimation. Building on these studies and other literature that explores the factors influencing SLAM performance (e.g., [37, 42, 43, 55, 57, 62]), we develop an estimation model that accounts for *both sensor inputs and internal SLAM status*. Unlike the aforementioned offline methods, our system, SEESys, is the first solution for obtaining pose error estimates at run-time.

Auditory displays and sonification. Auditory displays [14] communicate information and provide feedback to users through sound, while sonification [26] refers to the process of rendering data as sound. This use of sound within a tightly closed human-computer interface has been termed *interactive sonification* [25]. Previous works have successfully employed interactive sonification to guide human behavior. For example, in [20], sonification helped runners to improve their running mechanics, and Matinfar et al. [40] demonstrated that sonification is a reliable alternative to conventional visual displays for surgical task guidance. In our case, we demonstrate the sonification of pose error magnitude as a use case for online pose error estimation. The auditory feedback provided to SLAM system operators helps them adjust their movements to reduce tracking error by 25%.

3 BACKGROUND

In this section, we provide an overview of how visual SLAM systems operate (Section 3.1), followed by a discussion of factors influencing pose tracking errors (Section 3.2).

3.1 A Primer on Visual SLAM

A visual SLAM system is deployed on a mobile device to concurrently map an environment and track the pose of the device within that environment. The system takes in one or more camera images from onboard sensors as inputs and outputs the latest map and pose estimate at a predefined frequency. Visual SLAM algorithms can broadly be divided into feature-based methods (e.g., ORB-SLAM3 [10], VINS-Mono [47]), which extract, match, and track recognizable features from input camera images, and direct methods (e.g., LSD-SLAM [19], DSO [69]), which track by minimizing photometric error between camera images. Feature-based methods are generally more popular due to better robustness to lighting changes and greater efficiency [34, 75]; therefore, in this work, we focus on feature-based visual SLAM. SLAM algorithms can also be divided into filter-based and optimization-based solutions; here we focus on optimization-based solutions, which have also gained popularity due to their ability to maintain better global consistency [8, 39, 66]. Feature-based, optimization-based SLAM typically consists of three main modules [10]: tracking, local mapping, and loop closing. We describe the functions of each module below.

Tracking: The tracking module extracts keypoints from the sensor input—grayscale camera frames—using a keypoint detector (e.g., [51, 52]). Each detected keypoint is associated with a descriptor (e.g., [9, 35]), which includes information about surrounding pixels so that the keypoint can be matched across frames. Keypoints are matched in subsequent frames to estimate the camera pose, and keypoints tracked in this process are then matched with those in previously identified keyframes (reference frames). A transformation is identified based on the number of good keypoint matches, inliers, and from these correspondences, new 3D map points can be established via triangulation. The tracking module also determines whether the current frame is added as a keyframe, based on how distinct the camera view is from previous frames.

Local mapping: The local mapping module inserts new keyframes and 3D map points identified by the tracking module into the map [8, 10, 66]. It then optimizes keyframe poses and map points in a local window around the current camera frame using *bundle adjustment* [7]. Bundle adjustment works by minimizing visual error, the discrepancy between keypoints originally detected in camera frames and their positions as determined by keyframe poses and map points. Greater visual error indicates greater noise in the camera images and errors in keypoint detection and matching.

Loop closure: The loop closure module is responsible for recognizing places previously visited in an environment and adjusting the estimated poses and map accordingly [8, 10, 66]. Potential loops are detected by matching the keypoints in the current frame with previous keyframes. If a potential loop is identified, it is then validated using the geometric properties of keypoints, the quality of keypoint matches, and the consistency of the loop with the overall trajectory [38]. Validated loops are corrected by optimizing the essential pose graph (keyframes and associated map points). Full bundle adjustment is then performed to refine all keyframes and map points subject to the loop closure constraints.

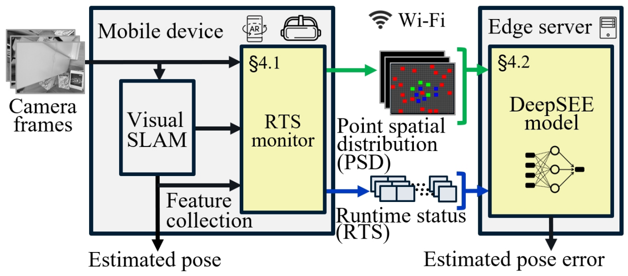

Figure 3: Overview of the SEESys architecture. Feature collection is accomplished on the mobile device, while model inference is performed on the edge server.

3.2 Factors Affecting Visual SLAM Pose Tracking Performance

The magnitude of pose tracking error in visual SLAM is influenced by the complex interaction between environmental characteristics and the device's movement within that environment. Here, we highlight three key factors that affect visual SLAM accuracy.

Visual texture: The detection of keypoints required for tracking relies on recognizable visual textures (variation in pixel intensities) in the camera image. As such, the surrounding environment must contain sufficient visual texture, and the texture in a camera image must be distinguishable by a corner detection algorithm at the current ambient light level. Textureless environment regions such as blank walls are frequently a problem for feature-based visual SLAM and result in greater pose error or even tracking losses [5, 8, 10, 30, 54–56]. Low ambient light levels introduce noise into camera images, which may corrupt the detection of visual features. Additionally, the quantity and spatial distribution of keypoints in a camera frame are essential to the robustness of pose estimation, where a large quantity of keypoint pairs can provide more redundancy while a uniform spatial distribution enables reliable pose estimation [57, 63].

Device motion: Rapid movement reduces similarity between subsequent camera frames, decreasing potential keypoint matches and increasing the likelihood of error [44]. Rapid rotation often causes a greater change in camera views than rapid translation. Additionally, rapid movement causes motion blur, hindering accurate feature detection and increasing error [5, 37, 43, 55]. Fine, low-contrast textures are particularly susceptible to blur [55]; environments with highly distinctive elements are relatively robust to rapid motion, but the same motion in an area with more subtle visual textures may make feature matching challenging or impossible.

Environment depth: The environment depth, the distance from the camera to surrounding surfaces, also plays an important role in determining how conducive a trajectory is to accurate pose tracking. On one hand, our experience with visual SLAM indicates that features detected on distant surfaces may be less reliable due to their decreased resolution, increased susceptibility to noise (including motion blur), and greater potential for occlusions. However, when a device camera is close to a surface, its field of view only covers a small environment region, which increases the potential for the camera view to contain limited visual texture [56].

3.3 SLAM Pose Error Reasoning

We conduct experiments to evaluate the abovementioned factors that potentially contribute to pose error, including the contrast in camera frames (representing the visual texture factor), the angular speed (derived from estimated device motion), the average map point depth (measuring the average depth of the surrounding environment), and the number of matched inliers (obtained from the SLAM tracking module). Figure 2 visualizes the distribution of pose errors across these factors using the SenseTime dataset [30], and we calculate the Pearson correlation coefficients between each factor and the pose error. The contrast in camera frames, average map point depth, and number of matched inliers (as shown in Figure 2.(a), 2.(c), 2.(d)) all exhibit a negative correlation with pose error, indicating that higher values of these factors tend to result in lower pose errors. On the contrary, angular speed (Figure 2.(b)) shows a positive correlation, suggesting that rapid rotations are associated with greater pose error. However, the correlations of these factors are weak (all ≤ 0.399), implying that pose error estimation is a complex problem that cannot be solved by a simple model. This highlights the need for more advanced machine learning models that can capture the underlying patterns among these factors.

Additionally, Figure 2.(d) demonstrates a clear correlation between the number of matched inliers, an internal factor of the SLAM system, and the pose error. Compared to previous work [2], which focused solely on sensor data characteristics, we argue that incorporating internal SLAM system status can lead to more accurate pose error estimation.

4 SYSTEM DESIGN

We present the architecture of our pose error estimation system, SEESys, in Figure 3. Since running visual SLAM alone would exhaust computational resources on a resource-constrained mobile device [17], SEESys leverages edge computing, where the computation for model inference is offloaded to a nearby server (e.g., [24, 36, 73, 77, 79]). This approach reduces the computational burden on the mobile device and ensures low-latency data transmission for real-time pose error estimation. SEESys consists of two primary modules: the RTS monitor (Section 4.1) for feature collection and preprocessing on the mobile device, and the DeepSEE model (Section 4.2) for pose error estimation on the edge server.

4.1 RTS Monitor

The lightweight RTS monitor operates concurrently with a SLAM system, on a mobile device. It collects and preprocesses features from various points in the SLAM pipeline, as illustrated in Figure 4. These features capture both the extent and distribution of visual information (i.e., recognizable texture) in the surrounding environment that facilitates pose tracking, as well as the current state of the SLAM algorithm. We tailor-design the RTS monitor to capture features that are commonly accessible in feature-based SLAM systems (see Section 3.1). We have carefully optimized the system to minimize resource consumption during runtime. Below we detail the two categories of our selected set of features: the Runtime Status (RTS) and the Point Spatial Distribution (PSD). Features are preprocessed, formulated as multivariate time series and 3D matrices, and subsequently transmitted to the edge server via a wireless network for model inference.

4.1.1 Runtime Status (RTS). It comprises features collected from the tracking, local mapping, and loop closure modules (see Section 3.1) during runtime to characterize SLAM performance. These

Table 1: Runtime status features captured by the RTS monitor for each camera frame. Sensor input characteristics are computed directly from camera images, while other features are extracted from the internal SLAM pipeline.

RTS feature category	Feature name	Description
Sensor input characteristics	Brightness	Mean normalized pixel intensity
	Contrast	RMS contrast (standard deviation of normalized pixel intensities)
	Entropy	Shannon entropy [61] of pixel intensities, image complexity
	Laplacian	Variance of the Laplacian, image edge strength (low values indicate blur)
SLAM tracking characteristics	MatchedInliers	Number of keypoints (features) matched in camera pose estimation
	Outliers	Number of keypoints (features) not matched in camera pose estimation
	RelativeTranslation	Relative camera translation (on x,y,z axes) in camera coordinates between consecutive camera frames
	RelativeRotation	Relative camera rotation (in roll, pitch, and yaw) between consecutive camera frames
SLAM mapping characteristics	AvgMPDepth	The mean of the depth of matched map points in a camera frame
	VarMPDepth	The variance of the depth of matched map points in a camera frame
	LocalMappingVisualError	Maps point reprojection error in local mapping optimization

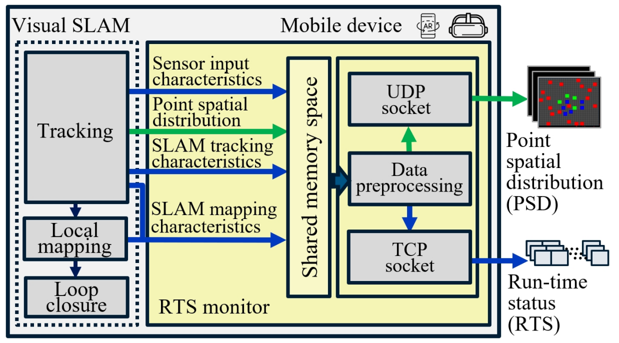

Figure 4: Workflow of the Runtime Status (RTS) monitor, which collects features from the SLAM pipeline, preprocesses the features as RTS time series and PSD matrices, and then sends them to the edge server.

features are detailed in Table 1. We categorize them into sensor input characteristics, SLAM tracking characteristics, and SLAM mapping characteristics.

- **Sensor input characteristics** capture properties of the visual input to the SLAM pipeline, grayscale camera frames. The *Brightness* of a frame indicates environment illumination, which determines how visible textures are as well as the likelihood of noise [18]. *Contrast* measures how distinguishable environment textures are; this is critical because the corner detection algorithms central to keypoint detection rely on sufficient pixel intensity differences [51]. *Entropy* measures image complexity and the amount of information available. *Laplacian* measures the edge strength of each camera frame, with lower values indicating blur, which can reduce tracking performance [37, 43].
- **SLAM tracking characteristics** are extracted from the SLAM tracking module. Motivated by the visual SLAM tracking optimization process (see Section 3), we collect *MatchedInliers* and *Outliers*. *MatchedInliers* captures the number of matched inlier points after pose estimation, while *Outliers* captures the number of outliers. Inspired by related works in visual odometry [43], image deblurring [37], and AR environment design [55] that point out the impacts of motion blur, we collect the *RelativeTranslation*, which represents the translation change along each axis to the previous camera pose (in camera coordinates), while *RelativeRotation* represents the estimated rotation change in the roll, pitch, and yaw.
- **SLAM mapping characteristics** are data related to the generated SLAM map. In Section 3.2, we note how environment depth impacts tracking performance. Consequently, we measure *AvgMPDepth* and *VarMPDepth*, the average and variance of map point depths within the camera frustum respectively. The *AvgMPDepth* and *VarMPDepth* features are extracted from the tracking module. Additionally, we capture *LocalMappingVisualError* from the local mapping module, which is the reprojection error of map points in the local map following local bundle adjustments by the SLAM system. As we covered in Section 3.1, this metric reflects the extent of noise in camera images and errors in feature matching.

4.1.2 Point Spatial Distribution (PSD). While the number of inlier points in each frame is a useful feature, it is known that the distribution of these inlier points significantly impacts the structure-from-motion and image localization tasks: a clustered distribution is generally detrimental while an even distribution is robust [21, 57]. Drawing inspiration from these findings and recognizing that an inlier point is a keypoint that can be associated with a 3D map point (see Section 3.1), we have designed and implemented a novel data structure: the PSD matrix, that records the spatial distribution for the keypoints, map points, and inlier points. This matrix further characterizes tracking states beyond the original RTS features. To our knowledge, no previous work has explicitly explored this feature for modeling SLAM tracking performance. As illustrated in Figure 5, the PSD matrix is concatenated from three 2D matrices, each shown in a different color, red for map points, green for inlier points, and blue for keypoints. These 2D matrices represent the spatial distribution and properties of their respective points, as detailed below.

- **Keypoints spatial distribution** is extracted in each camera frame by feature-based SLAM algorithms (e.g., ORB extraction

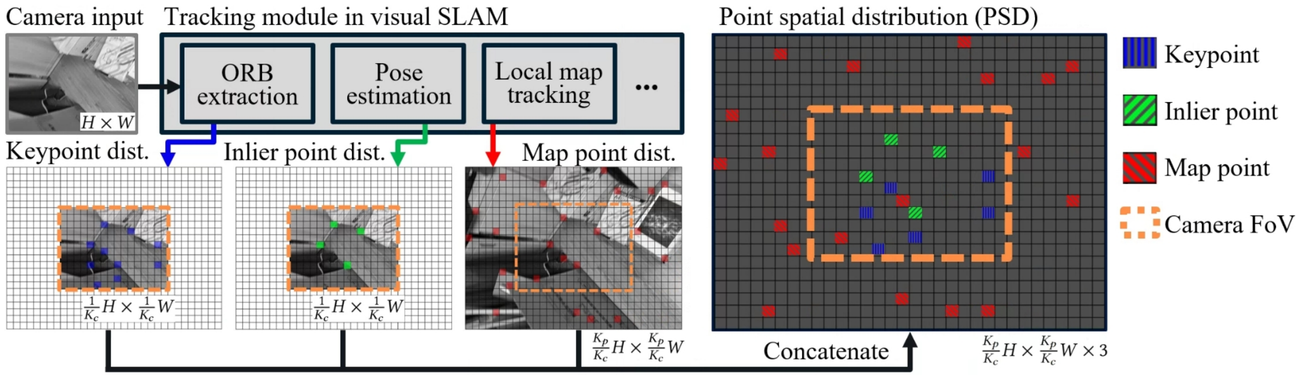
Camera input
Tracking module in visual SLAM
$H \times W$
ORB extraction
Pose estimation
Local map tracking
...
Keypoint dist.
Inlier point dist.
Map point dist.
$\frac{1}{K_c}H \times \frac{1}{K_c}W$
$\frac{1}{K_c}H \times \frac{1}{K_c}W$
$\frac{K_p}{K_c}H \times \frac{K_p}{K_c}W$
Point spatial distribution (PSD)
Keypoint
Inlier point
Map point
Camera FoV
Concatenate
$\frac{K_p}{K_c}H \times \frac{K_p}{K_c}W \times 3$

Figure 5: PSD matrix generation (best viewed in color). For each camera frame, we record the spatial distribution of keypoints, inlier points, and map points from the visual SLAM tracking module.

in the tracking module of ORB-SLAM3 [10]). To reduce memory usage, model dimensions, and offloading bandwidth, we reduce our PSD resolution by dividing the camera frame space, into small patches of size $K_c \times K_c$. This effectively reduces our PSD resolution to $\frac{1}{K_c}H \times \frac{1}{K_c}W$, where K_c represents the size of the patch. We first initialize the matrix as a zero matrix, where all the elements are zero. Then, for each small patch in a camera frame, we count the keypoints extracted in the SLAM tracking module that fall into the region of that small patch. If there are keypoints in the small patch region, we set the corresponding matrix element to the mean response value of those keypoints. The response is a keypoint property, where a higher value indicates greater distinctiveness and robustness to noise [3, 52]. Thus, the 2D matrix of keypoints captures the quality of keypoints through their response values, as well as their spatial distribution.

- **Inlier points spatial distribution** is captured in a similar way to the keypoints in a 2D matrix of size $\frac{1}{K_c}H \times \frac{1}{K_c}W$, where each matrix entry corresponds to a small patch with shape $K_c \times K_c$ in the camera frame. For each small patch, if there are no inlier points found in the pose estimation process in the SLAM tracking module, we set the corresponding matrix entry to 0, otherwise, we set it to 255.
- **Map points spatial distribution** considers map points that are located not only within the current camera field of view (FoV) but also surrounding the camera FoV in the SLAM local map. This design is motivated by our observation that tracking performance is more robust in environments with rich map points than in new environments with fewer map points. Including map points outside of the camera FoV captures surrounding information about the current camera pose, which helps in estimating the pose error when we jointly consider the surrounding map point distribution with the camera movement reflected in the relative pose. To achieve this, we create an expanded frustum that is larger than the camera frustum by a frustum expansion factor K_p and set the matrix shape to $\frac{K_p}{K_c}H \times \frac{K_p}{K_c}W$. Here we follow our compression design in the keypoint and matched inliers spatial distribution that divides camera frame space into small $K_c \times K_c$ patches to reduce resource consumption. We then extract map points from the tracking module of visual SLAM, reproject the 3D coordinates of map points on the local map onto the 2D camera plane, and convert their 3D world coordinates to 2D coordinates of the camera frame. If a map point is found within the expanded frustum, we record its depth value in the corresponding matrix entry, thereby providing the depth information of map points over a larger FoV than the original camera frame.

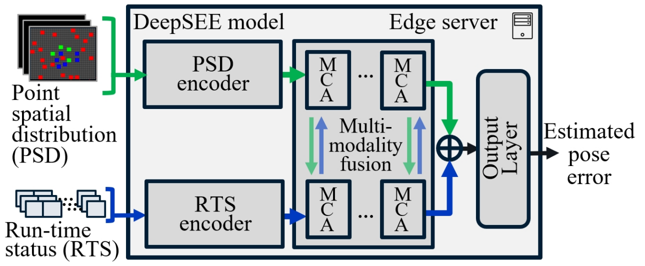

Figure 6: DeepSEE model architecture. The PSD and RTS encoders extract latent representations from their respective data, which are then fused using Multi-head Cross-attention (MCA) layers. The fused representations are processed by fully connected layers to estimate pose error.

We then concatenate the keypoint, inlier point, and map point spatial distribution together and form the PSD matrix, a 3D matrix with shape $\frac{K_p}{K_c}H \times \frac{K_p}{K_c}W \times 3$.

4.2 DeepSEE Model

The Deep SLAM Error Estimator (DeepSEE) model is an attention-based, multi-modality regression model we designed to take the multi-modality features collected by the RTS Monitor for pose error estimation. We design its architecture and training procedure to capture spatiotemporal patterns from both the PSD and RTS modalities.

4.2.1 Problem Statement. Given a SLAM benchmark with sensor data and ground-truth trajectories T, we first run a visual SLAM system on the sensor data to obtain its estimated trajectory $\hat{T}$ and runtime status $\mathbf{X}$ and point spatial distribution $\mathbf{P}$. We then derive the ground-truth pose error $\mathbf{y}_t$ by comparing the estimated trajectory $\hat{T}$ from the visual SLAM system with the ground-truth trajectory T using the Relative Pose Error (RPE) metric [80].

With ground truth pose error $\mathbf{y}_t$ and corresponding RTS modality input $\mathbf{X}_t$ and PSD modality input $\mathbf{P}_t$ at each camera frame's time stamp t, our goal is to train a neural network model, DeepSEE,

f_{DeepSEE}, to minimize the MSE Loss between the estimated pose error $\hat{y}_t$ and the ground truth error y_t, as denoted below:

$$\begin{aligned}\min_{\Theta} \mathcal{L}_{\text{MSE}}(\hat{\mathbf{y}}, \mathbf{y}) &:= \frac{1}{n}\sum_t (\hat{\mathbf{y}}_t - \mathbf{y}_t)^2 \\ \text{where } \hat{\mathbf{y}}_t &= f_{\text{DeepSEE}}(\Theta, \mathbf{X}_t, \mathbf{P}_t)\end{aligned} \tag{1}$$

Here, $\hat{\mathbf{y}}_t$ is the estimated pose error from our DeepSEE model, with Θ representing its parameters. The term n denotes the total number of samples in the dataset, $\mathbf{X}_t$ the RTS modality input, and $\mathbf{P}_t$ the PSD modality input. By optimizing this towards a lower loss value, the DeepSEE model can learn from the input modalities and estimate pose errors accurately.

The RTS modality input X_t is a sliding window of the runtime status features from timestamp $t - T + 1$ to timestamp t. It can be formulated as a multivariable time series as $X_t \in \mathbb{R}^{T \times C}$, where C represents the number of features as outlined in Table 1 and T specifies the length of the sliding window, thereby setting a fixed length for the RTS modality input.

Similarly, the PSD modality input P_t is a sliding window for the PSD matrices. Since adjacent camera frames have similar PSD, the sliding window only contains samples at timestamps $t-(K_L-1)\tau, t-(K_L-2)\tau, \cdots, t-\tau, t$ instead of taking all the PSD matrices within a time slot to reduce the redundancy in the PSD modality input and minimize the system resource overhead. The K_L denotes the number of PSD samples within the sliding window, and τ represents the interval between adjacent samples. The input P_t can then be presented as $P_t \in \mathbb{R}^{K_L \times \frac{K_p}{K_c}H \times \frac{K_p}{K_c}W \times 3}$, where H and W are the height and width of the camera frame, K_p is the frustum expansion factor, K_c is the compression ratio, and the 3 corresponds to the number of layers for keypoints, map points, and inlier points as we specified in Section 4.1.2.

4.2.2 *Model Architecture and Training Strategy.*

The DeepSEE model incorporates four main modules: a PSD encoder, an RTS encoder, a multi-modality cross-attention fusion module, and an output layer module. The model architecture is shown in Figure 6.

PSD Encoder f_{PSD}: We employ an attention-based video encoder, TimeSformer [4], to encode our PSD modality input to latent space. We choose an encoder originally designed for video inputs as we observed that the PSD modality input is a sliding window of PSD matrices sampled over a time period with a data structure $\{P_i\} \in \mathbb{R}^{K_L \times \frac{K_p}{K_c}H \times \frac{K_p}{K_c}W \times 3}$, which is similar to a video clip with K_L as the video length and $\frac{K_p}{K_c}H \times \frac{K_p}{K_c}W$ as the video resolution. Like other attention-based video encoders, the first layer of our PSD encoder is an embedded layer, which divides the input into non-overlapping patches with shape $H_p \times W_p \times 3$, where the H_p and W_p are the height and width of the small patches. It then extracts hidden representations across patches in both spatial and temporal dimensions, meaning for each patch in a PSD matrix, the model not only considers patches located at different spatial positions in the same PSD matrix but also patches at the same spatial position at different timestamp over the sliding window.

To ensure our PSD encoder can capture the representation of a sliding window of the PSD matrices, we design a self-supervised learning task to pre-train it. We attach two fully connected layers followed by the PSD encoder as the decoder, and ask the model to recover the original sliding window of a masked sliding window, where we randomly mask 50% of patches. The loss is the MSE between the original sliding window and the reconstructed sliding window. The design of the self-supervised learning task encourages the PSD encoder to learn latent space representations from each patch so they can be used to reconstruct the masked neighboring patches (in both time and space) in the decoder. We then store the learned weights of the PSD encoder to initialize the DeepSEE model.

RTS Encoder f_{RTS}: For encoding the RTS in the form of multivariate time series, we employ the TS2Vec encoder [76], which can learn latent space representations of time series data at multiple semantic levels. This is achieved through contrastive learning, applied hierarchically across augmented contextual views, enabling the encoder to capture complex, time-dependent patterns in the multivariate time series data.

Cross-attention Multi-modality Fusion Module: Upon obtaining the hidden representations of the SLAM runtime status and point distributions from their respective encoders, the integration of these multimodal insights is achieved through a bi-directional multi-modality cross-attention fusion module. This module facilitates a mutually informative interaction between the two modalities, allowing each to effectively utilize information from the other [16, 67, 71].

In the Multi-head Cross-attention (MCA) layers, attention computations occur bidirectionally between the RTS and PSD modalities. For the RTS modality, we denote the queries, keys, and values as Q_{RTS}, K_{RTS}, and V_{RTS} respectively; and for the PSD modality, these are denoted as Q_{PSD}, K_{PSD}, and V_{PSD}. The attention scores are calculated by setting the queries from one modality against the keys and values from the opposite modality:

$$\text{Att}_{RTS \to PT}(Q_{RTS}, K_{PSD}, V_{PSD}) = \text{softmax}\left(\frac{Q_{RTS}K_{PSD}^T}{\sqrt{d_k}}\right)V_{PSD}$$

And similarly, for the reverse direction:

$$\text{Att}_{PSD \to RTS}(Q_{PSD}, K_{RTS}, V_{RTS}) = \text{softmax}\left(\frac{Q_{PSD}K_{RTS}^T}{\sqrt{d_k}}\right)V_{RTS}$$

This bi-directional approach not only enhances the depth of feature extraction from each modality but also ensures that each modality effectively queries the most relevant features in the data stream of the other, thereby enhancing model performance.

Output Layer Module: In this module, we begin by concatenating the hidden states derived from the bidirectional cross-attention module. This operation combines the hidden representation extracted from both the RTS and PSD modalities. Following this, we employ a series of fully connected layers. These layers transform the concatenated hidden states into a continuous numerical output. The final output of this module is the pose error estimate.

5 SYSTEM IMPLEMENTATION

In this section, we detail the implementation of our online pose error estimation system SEESys. In Section 5.1, we provide an overview of our implementation. In Section 5.2, we describe how we obtain annotated labels for our error estimation model. In Section 5.3, we present the synthetic datasets we created to train our model, and in Section 5.4, we detail our model training procedure.

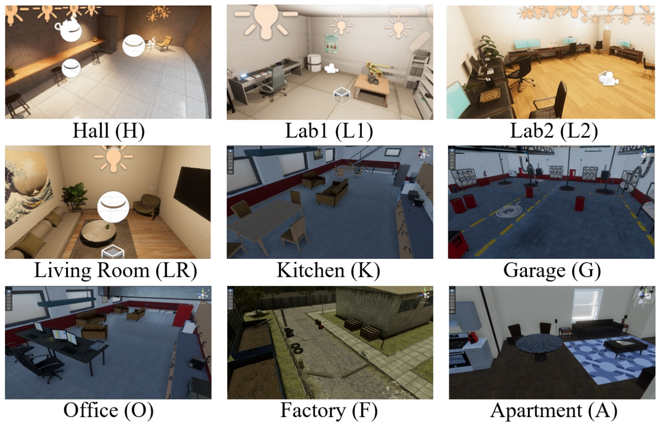

Figure 7: Unity virtual environments created to generate our synthetic visual SLAM datasets.

5.1 Implementation Overview

5.1.1 Hardware. Following recent works in visual SLAM on mobile devices [1, 74, 78], we selected the Nvidia Jetson Xavier NX as the mobile system platform for our prototype. The Nvidia Jetson Xavier NX features a 6-core ARMv8 CPU, operating at a maximum frequency of 1.9GHz, and includes 8GB of RAM. Its CPU performance is comparable to that of the Google Pixel 2, which is powered by a Snapdragon 835 CPU with 4 cores at 1.9 GHz and 4 cores at 2.45 GHz. Additionally, the Nvidia Jetson runs on an Ubuntu operating system, which simplifies the fulfillment of ORB-SLAM3 dependencies.

For the edge server that hosts our DeepSEE model, we use a laptop running on Ubuntu 22.04 with an Intel Core i7-12800H CPU, an Nvidia RTX3080 GPU, and 64GB RAM.

5.1.2 Software. We implemented the RTS monitor on the mobile device in C++. To address the overhead associated with data collection from the SLAM system, we integrated the RTS monitor as an additional thread within the existing visual SLAM system, ORB-SLAM3 [10]. This integration aligns with the conventional multi-threaded design of visual SLAM systems and offers advantages in shared memory access (allowing the RTS monitor to easily access necessary parameters from the SLAM pipeline) as well as concurrency and resource management (ensuring that interactions with shared variables are safely managed). We implemented the DeepSEE model on the edge server in Python with PyTorch.

5.1.3 Networking. The mobile device and the edge server are connected to the same Wi-Fi router operating on a 5.0 GHz frequency band. We use the *iftop* command on the edge server to track the uplink and downlink bandwidth filtered by the edge and mobile device's IP addresses. The average upstream bandwidth of the mobile device at runtime for sending the features is 663 Kb/s.

5.2 Label Transformation

After generating the ground truth labels, we found that the distribution of RPE is positively skewed with a long right tail. Due to this severely unbalanced distribution, the model is likely to ignore those rare but critical large pose errors. In practice, we care more about large pose errors than small pose errors. To address

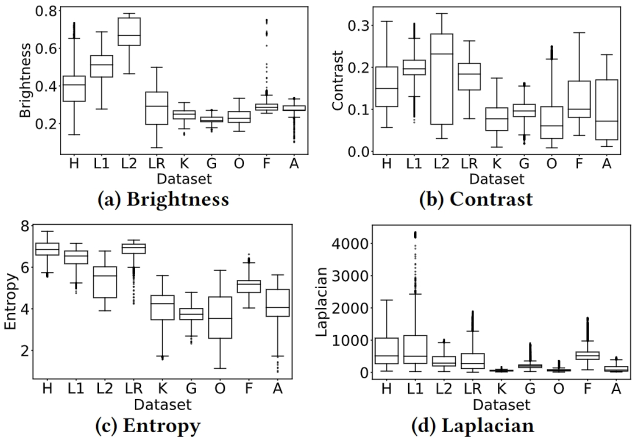

Figure 8: Sensor input characteristics for each of our synthetic visual SLAM datasets. We designed our set of virtual environments (Figure 7) to produce diverse levels of input image brightness, contrast, entropy (complexity) and Laplacian (edge strength).

this, we apply a log transformation that uses $\log(1 + K_L \times RPE)$ as the new label, where K_L is a shifting parameter. After applying the log transformation with $K_L = 10,000$, the label distribution is similar to a Gaussian distribution, thereby addressing the imbalance issue. After model inference, we apply the inverse transformation, $\frac{exp(RPE_{\text{transformed}})-1}{K_L}$, to recover the estimated pose error.

5.3 Training Data

There is a scarcity of large-scale visual SLAM datasets (with accurate ground truth pose data) that cover a diverse range of realistic, commonly encountered visual environments, as opposed to lab environments (e.g., [30, 58]) or environments designed specifically to be challenging (e.g., [70]). To address this, we use game engine-based synthetic data generation [23, 55, 59, 70], in which ground-truth trajectories from existing SLAM datasets are used to generate new visual input data (camera frames) in easily-modified virtual environments. As shown in Figure 7, we use this technique to create a diverse set of synthetic visual SLAM datasets that substantially augment our training data and improve the model generalization in new environments.

To produce realistic virtual environments for our synthetic datasets, we used the Unity game engine's High Definition Render Pipeline. We created nine diverse environments (Figure 7), that vary in multiple aspects, including illumination, scale and texture. To generate visual input data from each environment, we produced a new camera frame for each pose in the ground truth trajectories in the SenseTime [30] dataset. In total, we generated 221,195 camera frames in virtual environments (compared to 32,056 frames in the original SenseTime dataset) to use as visual SLAM inputs and collect labeled training data. To validate the diversity of these data, we calculated the sensor input characteristics (see Section 4.1) for each frame, and the results are shown in Figure 8. The median *Brightness* for each of our nine datasets ranges from 0.22 to 0.67, the median *Contrast* from 0.06 to 0.23, the median *Entropy* from 3.5 to 7.0, and the median *Laplacian* from 54 to 516. This diversity facilitates the training of

Table 2: Cross-validation evaluation rounds on the SenseTime dataset, with trajectories grouped by their motion types.

Group	Motion types	Trajectories
1	Patrol and inspection	A0, A1, A2, A4, A7
2	Aiming and inspection	A3, A6, B3, B4, B5, B6, B7
3	Rapid waving, shaking, rotation, and translation	A5, B0, B1, B2

an error estimation model that is robust to the different types of environments in which mobile devices run SLAM.

5.4 Training Procedure

Our training procedure consists of three stages: self-supervised pretraining, supervised pretraining, and supervised fine-tuning, as described below. This methodology enables our model to learn universal representations from input features and effectively address the synthetic-to-real problem arising from training on synthetic datasets.

Self-supervised pretraining: We use self-supervised pretraining methods to train the PSD and RTS encoders. For the PSD encoder, we follow the self-supervised learning procedure in Section 4.2. For the RTS encoder, we follow the procedures in TS2Vec [76]. This stage employs contrastive learning to encode universal representations of RTS features.

Supervised pretraining: In this stage, we freeze the PSD and RTS encoders with the weights obtained from the self-supervised pretraining, then train the rest of the model, including the multi-modal cross-attention module and output layers, using our synthetic data. The goal is to leverage our diverse synthetic dataset to develop a robust initial model.

Supervised fine-tuning: The final stage involves fine-tuning the model with real-world data. By doing so, we adapt the model, initially trained on synthetic data, to perform effectively in real-world scenarios.

6 EVALUATIONS

Next, we evaluate several aspects of our SEESys system. In Section 6.1, we evaluate the error estimation performance of our DeepSEE model. In Section 6.2, we measure the overhead of our RTS monitor on the mobile device, and in Section 6.3, we characterize the latency of the overall SEESys system.

6.1 Evaluation of Pose Error Estimation

We evaluate our DeepSEE on a real-world handheld SLAM benchmark, SenseTime [30]. We compare the performance of DeepSEE against a baseline that employs random forest regression for offline pose error estimation [2].

6.1.1 Evaluation Dataset. The SenseTime benchmark [30] comprises 16 trajectories, labeled A0–A7 and B0–B7, representing various indoor environments and user movement patterns in mobile AR scenarios. Some of these environments and movements are intentionally designed to be challenging for SLAM systems, making this benchmark ideal for investigating SLAM pose errors and evaluating our proposed online error estimation system. The SenseTime benchmark provides camera frames at a frame rate of $30Hz$ with

Table 3: Pose error estimation evaluation results using RMSE and MAPE metrics.

Category	Method	RMSE (cm)↓	MAPE↓
Baseline	Random Forest	0.279	0.549
	LSTM	0.273	0.708
Model Ablation Study	DeepSEE-w/o-PSD	0.261	0.510
	DeepSEE-w/o-SSL	0.236	0.526
	DeepSEE-w/o-CA	0.255	0.511
RTS Ablation Study	DeepSEE-w/o-Sensor	0.238	0.515
	DeepSEE-w/o-Tracking	0.244	0.569
	DeepSEE-w/o-Mapping	0.237	0.517
PSD Ablation Study	DeepSEE-w/o-Keypoint	0.238	0.514
	DeepSEE-w/o-Inlier	0.241	0.515
	DeepSEE-w/o-Mappoint	0.239	0.522
	DeepSEE	**0.235**	**0.507**

a resolution of 640 × 480, along with 6-DoF ground truth poses at $400Hz$ for pose error evaluation.

We employ cross-validation to maximize the use of available trajectories. In this process, trajectories alternately serve as training, validation, and test sets during the final fine-tuning stage. Our cross-validation data splitting is detailed in Table 2, where we categorize the SenseTime trajectories into three groups based on their motion types. In each round of cross-validation, we randomly sample two trajectories from each group. We put one of the sampled trajectories from each group into the validation set, the other into the test set, and the rest into the training set. No trajectories are shared among the training, validation or test set, ensuring that the evaluation is conducted on an unseen environment.

6.1.2 Evaluation Results. We assess the pose error estimation performance of DeepSEE and two baselines, random forest regression and LSTM, using two common metrics, Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE). The random forest regression model is an ensemble learning method used by the state-of-the-art offline pose error estimation method [2], while the LSTM model is a recurrent neural network designed to handle time-series data [27]. Our LSTM architecture for multivariate time series regression includes two LSTM layers, which capture the temporal dependencies and interactions between different input features in RTS, followed by fully connected layers to project the latent representations to the estimated pose error. Our evaluation results in Table 3 show that DeepSEE and its variants outperform both baselines. The full version of DeepSEE achieves an RMSE of 0.235 cm, an improvement of 15.8% over random forest regression and 13.9% over LSTM.

To evaluate the impact of the PSD modality, we compare system performance using only the RTS modality versus using both RTS and PSD modalities. Table 3 shows the results, where the setting without the PSD modality is denoted as DeepSEE-w/o-PSD. The results demonstrate that incorporating the PSD modality improves estimation accuracy, as the DeepSEE model reduces RMSE by 9.9% compared to DeepSEE-w/o-PSD.

We further assess the effect of self-supervised learning (SSL) on the PSD encoder by testing performance without initializing the PSD encoder weights with SSL, denoted as DeepSEE-w/o-SSL. While DeepSEE-w/o-SSL achieves a similar RMSE to DeepSEE, it exhibits a 3.8% increase in MAPE, indicating that self-supervised learning enhances performance.

To verify the effectiveness of the Cross-attention Multi-modality Fusion Module, we evaluate a variant, DeepSEE-w/o-CA, where the cross-attention fusion module is replaced with fully connected layers that directly process RTS and PSD modality representations. The results show an 8.5% increase in RMSE without the cross-attention module, highlighting its importance.

We conduct an ablation study to investigate the contribution of features obtained from the SLAM pipeline focusing on both RTS and PSD features. For RTS, we remove each category individually and find that sensor input and SLAM mapping characteristics contribute less than 2% improvement in RMSE and MAPE. However, SLAM tracking characteristics yield a 3.8% improvement in RMSE and a 12.2% improvement in MAPE, indicating that all features are valuable for pose error estimation, with tracking characteristics having the most significant impact. For PSD features, we evaluate model performance by masking the keypoint, inlier point, and map point spatial distribution matrices individually. The results show that map point and inlier point features contribute the most, improving MAPE by 3.0% and RMSE by 2.6%, respectively. Keypoints contribute less, with improvements of less than 1.4% in both metrics. We hypothesize that the extended frustum in the map point spatial distribution design provides a broader environmental perception beyond the camera's field of view, thereby enhancing the model's performance. Inliers contribute more than keypoints because they represent a subset of keypoints directly involved in pose prediction, making them more relevant to tracking accuracy.

6.2 RTS Monitor Overhead

To evaluate the overhead introduced by the RTS monitor, we measured resource usage and tracking latency on the mobile device when running ORB-SLAM3 with and without our RTS monitor, for the A0-A7 trajectories in the SenseTime dataset, with a Nvidia Jetson Xavier NX as the mobile device. For resource usage, we measured the CPU usage and memory usage using psutil [49], a Python library for system utilization. For tracking latency, we measured the time elapsed for the mobile device to process each camera frame.

The results of our RTS monitor overhead evaluation are shown in Table 4. The addition of the RTS monitor increases average CPU usage on the mobile device from 40.7% to 45.3%, a relative increase of 11.3%. The respective values for average memory usage are 466.3MB and 506.9MB, a relative increase of 8.7%, and the respective values for tracking latency are 34.8ms and 37.3ms, reflecting a 7.2% relative increase. Overall, this relative increase of less than 12% across multiple metrics illustrates the lightweight nature of our RTS monitor, showing that it can be run on a mobile device with a small overhead.

6.3 System Latency Characterization

Next, we evaluate the end-to-end latency of SEESys to demonstrate its suitability for real-time applications. We recorded system latency during the user study for our case study (Section 7), involving 30 participants, with each participant's trajectory lasting approximately two minutes. The average round-trip latency for data collection

Table 4: Mobile device resource consumption and tracking latency when running an ORB-SLAM3 alone (SLAM) and with the RTS monitor (SLAM+RTS).

Metric	SLAM	SLAM+RTS
Average CPU usage (%)	40.7	45.3
Median CPU usage (%)	45.0	51.7
Average memory usage (MB)	466.3	506.9
Median memory usage (MB)	515.0	560.9
Average tracking latency (ms)	34.8	37.3
Median tracking latency (ms)	35.6	37.6

and preprocessing on the mobile device was 1.9 ms. The average latency between transmitting the collected features and receiving the corresponding estimated pose error from the edge server was 35.4 ms. These results together demonstrate that SEESys provides pose error estimates in real time with an end-to-end latency of 37.3ms, which is negligible in our following case study.

7 CASE STUDY: AUDIO ERROR ADVISORY

In this section, we conduct a case study on one possible use case for online pose error estimation, an audio error advisory. In Section 7.1, we cover the motivation for this advisory, and in Section 7.2 we detail its implementation. In Section 7.3, we present the design of a user study to demonstrate our advisory, and in Section 7.4, we provide our user study results.

7.1 Motivation

As the use of mobile devices running SLAM becomes more widespread, these devices are more frequently controlled by human operators who are often unaware to the effects of motion on pose error. For example, someone scanning a room using a handheld device may change direction rapidly, not realizing that this may result in greater pose tracking error and in turn lower the accuracy of the environment map they create. Thus, a solution is needed to advise users when their motions cause high pose errors, allowing them to slow down and improve tracking quality.

One option would be to generate this error advisory based on the estimated device motion alone. However, this ignores the effect of the visual environment; as we noted in Section 3, SLAM is more robust to rapid motion in certain visual environments compared to others. As such, an advisory based on motion alone would likely produce unnecessary warnings in some environments or too few warnings in others. Instead, our SEESys system provides a timely pose error indicator that takes into account both visual and motion characteristics.

A SLAM pose error advisory could take various forms, such as visual, auditory, or haptic feedback. In this work, we focus on an audio error advisory, because it can be easily integrated with a wide range of mobile devices (i.e., minimal hardware requirements), and does not interfere with a human operator's vision-based navigation of an environment. The effectiveness and usability of different forms of error advisory is an important topic for future work.

7.2 Audio Advisory Implementation

We propose the *sonification* [26] of pose error magnitude, such that an audio signal conveys how pressing the need is for a human operator to reduce the speed of device motion. While we experimented

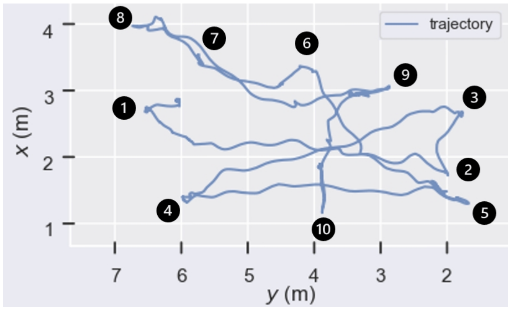

Figure 9: Top view of target object placement and a mobile device trajectory in our user study, where participants are instructed to scan the objects from 1 to 10.

with modulating the amplitude or frequency of a tone based on error magnitude, we ultimately chose to modulate the frequency of audio pulses, similar to a Geiger counter [53], as the most familiar and intuitive type of audio feedback. The Geiger counter metaphor has previously been used to encode distance in auditory displays for navigation [28, 41] and peripersonal reaching [72], but we are the first to propose it for encoding SLAM pose error.

Given the latency associated with generating audio signals from a pose error estimate (e.g., using a Python library such as sounddevice [64]), especially on mobile devices with fewer resources, we opt to generate audio feedback at 3 Hz. To this end, we calculate the moving average of the pose error estimates e using overlapping sliding windows of 0.33s, and set the maximum duration of the generated audio signal to 0.33s. The number of audio pulses N in the audio signal is calculated according to the estimated pose error e as follows:

$$N = \begin{cases} 0 & \text{if } e < \alpha \\ \lfloor 250e \rfloor & \text{if } \alpha \le e < \beta \\ \lfloor 0.33/(p+i) \rfloor & \text{if } e \ge \beta \end{cases} \tag{2}$$

where α is a pose error threshold below which audio is not generated, β is a pose error threshold above which the audio pulses N would not increase. p and i are the audio pulse length and inter-pulse interval, respectively. In our user study, we set $\alpha = 0.4cm$, $\beta = 1.2cm$, $p = 0.001s$ and $i = 0.099s$. We implement our audio advisory on the same mobile device as in Section 5.1, the Nvidia Jetson Xavier NX. Once the edge server completes model inference, it transmits the estimated pose error to the mobile device using a UDP socket via a wireless network. There are two concurrent processes in our audio advisory system: one receives the UDP packets and generates the audio signal based on the estimated pose error using Equation 2, while the other plays the latest audio signal on a speaker. This multi-processing setup allows low-latency communication and real-time audio feedback, enhancing system responsiveness and reliability.

7.3 User Study Design

We conducted an IRB-approved user study, in which we compared three possible conditions for an audio error advisory: (1) no audio advisory, (2) an audio advisory based on estimated device rotation speed, and (3) an audio advisory based on our SEESys error estimate. We estimated device rotation speed as the moving average of the RMS of relative rotation in yaw, pitch, and roll (see Table 1), over the same sliding window as the SEESys-based advisory. We then used this rotation value θ (in degrees) to generate N audio pulses, where $N = \max\{\lfloor \frac{\theta}{50^\circ} \rfloor - 1, 0\}$. Our hypotheses were that the presence of an audio advisory would result in lower pose tracking error and fewer tracking losses and that the SEESys-based advisory would be a more effective method than the device rotation-based advisory.

To test our hypotheses, we designed a task simulating a room scanning or inspection scenario. In this scenario, a user moves around a room while holding a mobile device running SLAM (e.g., a smartphone). This device is either used to map the space and objects within it (e.g., to create a digital twin) or to inspect virtual content attached to physical objects (e.g., a persistent AR experience).

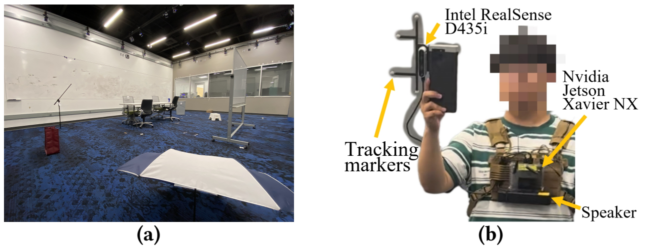

Figure 10: Setup of the user study used to test the effectiveness of our audio error advisory: (a) the study environment, including objects used for the task; (b) the study apparatus used by participants.

Participants: We recruited 30 participants (9 female, 21 male; aged 18 to 28 years) from our personal and professional networks. All participants had normal or corrected to normal eyesight and hearing. Participants were split into three groups of 10, corresponding to each of the audio advisory conditions.
Experiment setup and apparatus: The experiment setup is shown in Figure 10. The study was performed under controlled conditions in an 8.8 $m \times 6.2\ m \times 3.8\ m$ (length × width × height) lab space with no exterior windows and fixed overhead lighting. We used a Vicon motion capture system [68] to obtain ground truth pose. We placed 10 objects in random positions throughout the space and labeled them with the numbers 1–10 as shown in Figure 9. Participants held a smartphone to simulate the use of a mobile device. Optical tracking markers were attached to the smartphone to facilitate ground truth pose measurements, and the Intel RealSense D435i was used to capture SLAM sensor inputs (camera frames). To implement SEESys we used the setup detailed in Section 5.1, and implemented our audio advisory as described in Section 7.2. Participants carried the NVIDIA Jetson Xavier NX mobile device, with an attached speaker that played the audio output.
Task: Participants started at a fixed position, with the smartphone facing object 1. Participants were instructed to move the smartphone slowly while remaining in this position until tracking initialization was verbally confirmed by the experiment administrator. Participants then navigated from objects 1 to 10 in order; at each object, they scanned all visible parts of the object from a distance of approximately 0.5m.
Performance metrics: For each participant's device trajectory we measured *tracking error* using RPE, and *tracking completeness* using the percentage of camera frames tracked.

Table 5: Results of our user study for each dependent variable. The SEESys-based audio advisory resulted in lower tracking error and greater tracking completeness than both the no audio condition and the device rotation-based audio advisory.

Performance metric	No audio	Audio (Rotation)	Audio (SEESys)
Tracking Error (cm)	1.2	1.0	0.9
Tracking Completeness (%)	70.6%	87.3%	88.7%

Procedure: After participants signed a consent form, they answered a pre-experiment questionnaire via Qualtrics [48]. Participants assigned to an audio advisory condition then completed system training, in which they were instructed how to use the audio advisory, and practiced responding to it in a separate environment to the study area. All participants then completed task training, in which the experiment administrator explained the task, modeled it, and watched the participant do it. Once it was clear they understood, participants completed the main task. Finally, participants completed a post-experiment questionnaire via Qualtrics.

7.4 User Study Results

Our study results are shown in Table 5 (averages across all participants). Without any advisory (no audio), user motion led to frequent and prolonged tracking losses, and as a result a tracking completeness of just 70.6%. During these losses, the areas and objects a mobile device faces are not scanned successfully. This no audio condition also resulted in the greatest tracking error, 1.2 *cm*. The addition of a device rotation-based audio advisory improved tracking performance, with a tracking error of 1.0 *cm* and tracking completeness of 87.3%. The SEESys-based audio advisory resulted in the best tracking performance, with a tracking error of 0.9cm and tracking completeness of 88.7%. The benefit of using SEESys was particularly marked for tracking error; the SEESys-based advisory achieved 10% lower error than using device rotation for the audio advisory, and 25% lower error compared to no audio advisory.

The tracking performance improvements achieved via our SEESys-based audio error advisory are just one example of how SEESys can enhance SLAM performance. In some scenarios, an audio advisory may be undesirable because it impinges on user experience; in others, adjusting mobile device motion may not be appropriate for the intended application. In addition to user-centered adjustments, our system could also support automated optimizations. The pose error estimates provided by SEESys could enable automatic adjustment of internal SLAM parameters and resource allocation to optimize both tracking performance and resource consumption. Specifically, we envision the use of reinforcement learning algorithms that use the SEESys pose error estimates as inputs, to intelligently control various aspects of the SLAM pipeline.

8 LIMITATION AND FUTURE WORK

8.1 Limitations

While we enhance the model's robustness by augmenting the training set with synthetic data from diverse environments, several limitations still persist. Notably, the current model does not account for varying camera specifications, such as resolution, aspect ratio, and field of view. These differences in camera specifications can impact SLAM tracking performance, yet the current DeepSEE model design does not incorporate this variation. Consequently, a model trained on one camera type may underperform on another when significant differences in camera specifications exist. Additionally, although 30 individuals participated in our user study, it was conducted in an indoor facility (as shown in Figure 10a), where we rely on a motion capture system to provide ground truth trajectories for evaluation. This constraint limits the applicability of our user study to outdoor scenarios. However, we can explore more varied indoor experiments in future work.

8.2 Future Work

While this work focuses on pose error estimation for visual SLAM, SEESys can be extended to other SLAM variants, such as visual-inertial SLAM, by adapting the model's input features. For instance, the RTS could incorporate IMU readings and inertial residuals as part of its sensor input and SLAM mapping characteristics. Expanding to other SLAM variants will introduce a broader range of downstream applications. Furthermore, we plan to investigate other factors in SLAM error reasoning and perform a more comprehensive evaluation of the SEESys in various scenarios. To begin with, we can follow established methodologies in mobile SLAM tracking evaluation [29, 30] to vary experimental conditions, such as motion types, lighting conditions, and environmental feature levels. We also aim to deploy SEESys on a wider range of mobile devices and conduct comparative studies across different hardware platforms to assess the system's adaptability. Additionally, leveraging an edge server enables further optimization of mobile-edge orchestration, enabling collaboration between mobile devices and the edge server through an adaptive offloading strategy [31, 60]. Overall, we believe that the current design of SEESys provides a solid foundation for developing a more comprehensive system for online SLAM pose error estimation.

9 CONCLUSIONS

This paper presented SEESys, the first system for online, deep learning-based SLAM pose error estimation. SEESys consists of two primary modules: the lightweight RTS monitor for feature collection, and the multi-modal DeepSEE model for deep learning-based SLAM error estimation. Our evaluations demonstrate the improved error estimation accuracy of DeepSEE over the existing offline baseline, that the RTS monitor can be implemented on mobile devices with only a small overhead, and that SEESys delivers error estimates in real time in a practical scenario. Additionally, our case study on one use case for online error estimation, an audio error advisory, shows that this guidance can help human operators of mobile devices improve tracking performance.

ACKNOWLEDGMENTS

We thank Prof. Boyuan Chen and the Duke Robotics Lab for sharing their mocap facility. This work was supported in part by NSF grants CSR-2312760, CNS-2112562, and IIS-2231975, NSF CAREER Award IIS-2046072, NSF NAIAD Award 2332744, a CISCO Research Award, and a Meta Research Award.

REFERENCES

[1] Ali J Ali, Zakieh Sadat Hashemifar, and Karthik Dantu. 2020. Edge-SLAM: Edge-assisted visual simultaneous localization and mapping. In *Proceedings of ACM MobiSys*.

[2] Islam Ali, Bingqing Wan, and Hong Zhang. 2023. Prediction of SLAM ATE using an ensemble learning regression model and 1-D global pooling of data characterization. *arXiv preprint arXiv:2303.00616* (2023).

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-up robust features (SURF). *Computer Vision and Image Understanding* 110, 3 (2008), 346–359.

[4] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. 2021. Is space-time attention all you need for video understanding?. In *Proceedings of ICML*.

[5] Mihai Bujanca, Xuesong Shi, Matthew Spear, Pengpeng Zhao, Barry Lennox, and Mikel Luján. 2021. Robust SLAM systems: Are we there yet?. In *Proceedings of IEEE/RSJ IROS*.

[6] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. 2016. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research* 35, 10 (2016), 1157–1163.

[7] Alvaro Parra Bustos, Tat-Jun Chin, Anders Eriksson, and Ian Reid. 2019. Visual SLAM: Why bundle adjust?. In *Proceedings of IEEE ICRA*.

[8] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. 2016. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics* 32, 6 (2016), 1309–1332.

[9] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. 2010. BRIEF: Binary robust independent elementary features. In *Proceedings of ECCV*.

[10] 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 multimap SLAM. *IEEE Transactions on Robotics* 37, 6 (2021), 1874–1890.

[11] Yongbo Chen, Shoudong Huang, Liang Zhao, and Gamini Dissanayake. 2021. Cramér–Rao bounds and optimal design metrics for pose-graph SLAM. *IEEE Transactions on Robotics* 37, 2 (2021), 627–641.

[12] Ying Chen, Hazer Inaltekin, and Maria Gorlatova. 2023. AdaptSLAM: Edge-assisted adaptive SLAM with resource constraints via uncertainty minimization. In *Proceedings of IEEE INFOCOM*.

[13] Santiago Cortés, Arno Solin, Esa Rahtu, and Juho Kannala. 2018. ADVIO: An authentic dataset for visual-inertial odometry. In *Proceedings of ECCV*.

[14] Ádám Csapó and György Wersényi. 2013. Overview of auditory representations in human-machine interfaces. *ACM Computing Surveys (CSUR)* 46, 2 (2013), 1–23.

[15] Xinke Deng, Zixu Zhang, Avishai Sintov, Jing Huang, and Timothy Bretl. 2018. Feature-constrained active visual SLAM for mobile robot navigation. In *Proceedings of IEEE ICRA*.

[16] Karan Desai and Justin Johnson. 2021. Virtex: Learning visual representations from textual annotations. In *Proceedings of IEEE/CVF CVPR*.

[17] Aditya Dhakal, Xukan Ran, Yunshu Wang, Jiasi Chen, and KK Ramakrishnan. 2022. SLAM-share: visual simultaneous localization and mapping for real-time multi-user augmented reality. In *Proceedings of ACM CoNEXT*. 293–306.

[18] Samuel F Dodge and Lina J Karam. 2018. Quality robust mixtures of deep neural networks. *IEEE Transactions on Image Processing* 27, 11 (2018), 5553–5562.

[19] Jakob Engel, Thomas Schöps, and Daniel Cremers. 2014. LSD-SLAM: Large-scale direct monocular SLAM. In *Proceedings of ECCV*.

[20] Martin Eriksson and Roberto Bresin. 2010. Improving running mechanics by use of interactive sonification. In *Proceedings of ISon*.

[21] Luca Ferranti, Xiaotian Li, Jani Boutellier, and Juho Kannala. 2021. Can you trust your pose? Confidence estimation in visual localization. In *Proceedings of IEEE ICPR*.

[22] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Proceedings of IEEE CVPR*.

[23] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. 2019. FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. In *Proceedings of IEEE/RSJ IROS*.

[24] Mengxi Hanyao, Yibo Jin, Zhuzhong Qian, Sheng Zhang, and Sanglu Lu. 2021. Edge-assisted online on-device object detection for real-time video analytics. In *Proceedings of IEEE InfoCom*.

[25] Thomas Hermann and Andy Hunt. 2005. An introduction to interactive sonification. *IEEE Multimedia* 12, 2 (2005), 20–24.

[26] Thomas Hermann, Andy Hunt, and John G Neuhoff. 2011. *The sonification handbook*. Vol. 1. Logos Verlag Berlin.

[27] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[28] Simon Holland, David R Morse, and Henrik Gedenryd. 2002. AudioGPS: Spatial audio navigation with a minimal attention interface. *Personal and Ubiquitous Computing* 6 (2002), 253–259.

[29] Tianyi Hu, Fan Yang, Tim Scargill, and Maria Gorlatova. 2024. Apple vs. Meta: A Comparative Study on Spatial Tracking in SOTA XR Headsets. In *Proceedings of ACM ImmerCom (co-located with ACM MobiCom)*.

[30] Li Jinyu, Yang Bangbang, Chen Danpeng, Wang Nan, Zhang Guofeng, and Bao Hujun. 2019. Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality. *Virtual Reality & Intelligent Hardware* 1, 4 (2019), 386–410.

[31] Caihong Kai, Hao Zhou, Yibo Yi, and Wei Huang. 2020. Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability. *IEEE Transactions on Cognitive Communications and Networking* 7, 2 (2020), 624–634.

[32] Mike Kasper, Steve McGuire, and Christoffer Heckman. 2019. A Benchmark for Visual-Inertial Odometry Systems Employing Onboard Illumination. In *Proceedings of IEEE/RSJ IROS*.

[33] Georg Klein and David Murray. 2007. Parallel tracking and mapping for small AR workspaces. In *Proceedings of IEEE/ACM ISMAR*.

[34] Nicola Krombach, David Droeschel, and Sven Behnke. 2017. Combining feature-based and direct methods for semi-dense real-time stereo visual odometry. In *Proceedings of IAS*.

[35] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. 2011. BRISK: Binary robust invariant scalable keypoints. In *Proceedings of IEEE/CVF ICCV*.

[36] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *Proceedings of ACM MobiCom*.

[37] Peidong Liu, Xingxing Zuo, Viktor Larsson, and Marc Pollefeys. 2021. MBA-VO: Motion blur aware visual odometry. In *Proceedings of IEEE/CVF ICCV*.

[38] Wen Lik Dennis Lui and Ray Jarvis. 2010. A pure vision-based approach to topological SLAM. In *Proceedings of IEEE/RSJ IROS*.

[39] Andréa Macario Barros, Maugan Michel, Yoann Moline, Gwenolé Corre, and Frédérick Carrel. 2022. A comprehensive survey of visual SLAM algorithms. *Robotics* 11, 1 (2022), 24.

[40] Sasan Matinfar, Mehrdad Salehi, Daniel Suter, Matthias Seibold, Shervin Dehghani, Navid Navab, Florian Wanivenhaus, Philipp Fürnstahl, Mazda Farshad, and Nassir Navab. 2023. Sonification as a reliable alternative to conventional visual surgical navigation. *Scientific Reports* 13, 1 (2023), 5930.

[41] David McGookin, Stephen Brewster, and Pablo Priego. 2009. Audio bubbles: Employing non-speech audio to support tourist wayfinding. In *Proceedings of HAID*.

[42] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics* 31, 5 (2015), 1147–1163.

[43] Janne Mustaniemi, Juho Kannala, Simo Särkkä, Jiri Matas, and Janne Heikkilä. 2018. Fast motion deblurring for feature detection and matching using inertial measurements. In *Proceedinngs of IEEE ICPR*.

[44] Luigi Nardi, Bruno Bodin, M Zeeshan Zia, John Mawer, Andy Nisbet, Paul HJ Kelly, Andrew J Davison, Mikel Luján, Michael FP O'Boyle, Graham Riley, et al. 2015. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *Proceedings of IEEE ICRA*.

[45] Linus Nwankwo and Elmar Rueckert. 2023. Understanding Why SLAM Algorithms Fail in Modern Indoor Environments. In *Proceedings of the International Conference on Robotics in Alpe-Adria Danube Region*.

[46] OptiTrack. 2024. OptiTrack. https://optitrack.com/.

[47] Tong Qin, Peiliang Li, and Shaojie Shen. 2018. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics* 34, 4 (2018), 1004–1020.

[48] Qualtrics. 2024. Qualtrics. https://www.qualtrics.com.

[49] Giampaolo Rodola. 2009. psutil. https://github.com/giampaolo/psutil.

[50] Antoni Rosinol, John J Leonard, and Luca Carlone. 2023. Probabilistic volumetric fusion for dense monocular slam. In *Proceedings of the IEEE/CVF WACV*.

[51] Edward Rosten and Tom Drummond. 2006. Machine learning for high-speed corner detection. In *Proceedings of ECCV*.

[52] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of IEEE/CVF ICCV*.

[53] Ernest Rutherford and Hans Geiger. 1908. An electrical method of counting the number of α-particles from radio-active substances. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 81, 546 (1908), 141–161.

[54] Tim Scargill, Ying Chen, Tianyi Hu, and Maria Gorlatova. 2023. SiTAR: Situated trajectory analysis for in-the-wild pose error estimation. In *Proceedings of IEEE ISMAR*.

[55] Tim Scargill, Ying Chen, Nathan Marzen, and Maria Gorlatova. 2022. Integrated design of augmented reality spaces using virtual environments. In *Proceedings of IEEE ISMAR*.

[56] Tim Scargill, Majda Hadziahmetovic, and Maria Gorlatova. 2023. Invisible textures: Comparing machine and human perception of environment texture for AR. In *Proceedings of ACM ImmerCom (co-located with ACM MobiCom)*.

[57] Johannes L Schonberger and Jan-Michael Frahm. 2016. Structure-from-motion revisited. In *Proceedings of IEEE/CVF CVPR*.

[58] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jörg Stückler, and Daniel Cremers. 2018. The TUM VI benchmark for evaluating visual-inertial odometry. In *Proceedings of IEEE/RSJ IROS*.

[59] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*. Springer, 621–635.

[60] Sina Shahhosseini, Tianyi Hu, Dongjoo Seo, Anil Kanduri, Bryan Donyanavard, Amir M Rahmani, and Nikil Dutt. 2022. Hybrid learning for orchestrating deep learning inference in multi-user edge-cloud networks. In *Proceedings of IEEE ISQED*. IEEE, 1–6.

[61] C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27, 3 (1948), 379–423.

[62] Xuesong Shi, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, et al. 2020. Are we ready for service robots? the OpenLORIS-Scene datasets for lifelong SLAM. In *Proceedings of IEEE ICRA*.

[63] Chester C Slama, Charles Theurer, and Soren W Henriksen. 1980. *Manual of photogrammetry*. Number Ed. 4.

[64] Python sounddevice. 2024. Python sounddevice. https://python-sounddevice.readthedocs.io/.

[65] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. 2012. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of IEEE/RSJ IROS*.

[66] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. 2017. Visual SLAM algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications* 9 (2017), 1–11.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017).

[68] Vicon. 2024. Vicon. https://www.vicon.com/.

[69] Rui Wang, Martin Schworer, and Daniel Cremers. 2017. Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras. In *IEEE/CVF ICCV*.

[70] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. 2020. TartanAir: A dataset to push the limits of visual SLAM. In *Proceedings of IEEE/RSJ IROS*.

[71] Xi Wei, Tianzhu Zhang, Yan Li, Yongdong Zhang, and Feng Wu. 2020. Multi-modality cross attention network for image and sentence matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10941–10950.

[72] Graham Wilson and Stephen A Brewster. 2015. Using dynamic audio feedback to support peripersonal reaching in visually impaired people. In *Proceedings of ACM SIGACCESS*.

[73] Jingao Xu, Hao Cao, Danyang Li, Kehong Huang, Chen Qian, Longfei Shangguan, and Zheng Yang. 2020. Edge assisted mobile semantic visual SLAM. In *Proceedings of IEEE InfoCom*.

[74] Jingao Xu, Hao Cao, Zheng Yang, Longfei Shangguan, Jialin Zhang, Xiaowu He, and Yunhao Liu. 2022. SwarmMap: Scaling up real-time collaborative visual SLAM at the edge. In *Proceedings of USENIX NSDI*.

[75] Nan Yang, Rui Wang, and Daniel Cremers. 2017. Feature-based or direct: An evaluation of monocular visual odometry. *arXiv preprint arXiv:1705.04300* (2017), 1–12.

[76] Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. 2022. TS2Vec: Towards universal representation of time series. In *Proceedings of AAAI*.

[77] Jinrui Zhang, Huan Yang, Ju Ren, Deyu Zhang, Bangwen He, Ting Cao, Yuanchun Li, Yaoxue Zhang, and Yunxin Liu. 2022. MobiDepth: real-time depth estimation using on-device dual cameras. In *Proceedings of ACM MobiCom*.

[78] Xinran Zhang, Hanqi Zhu, Yifan Duan, Wuyang Zhang, Longfei Shangguan, Yu Zhang, Jianmin Ji, and Yanyong Zhang. 2024. Map++: Towards user-participatory visual SLAM systems with efficient map expansion and sharing. In *Proceedings of ACM MobiCom*.

[79] Yunfan Zhang, Tim Scargill, Ashutosh Vaishnav, Gopika Premsankar, Mario Di Francesco, and Maria Gorlatova. 2022. Indepth: Real-time depth inpainting for mobile augmented reality. In *Proceedings of ACM IMWUT*.

[80] Zichao Zhang and Davide Scaramuzza. 2018. A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In *Proceedings of IEEE/RSJ IROS*.

[81] David Zuñiga-Noël, Alberto Jaenal, Ruben Gomez-Ojeda, and Javier Gonzalez-Jimenez. 2020. The UMA-VI dataset: Visual–inertial odometry in low-textured and dynamic illumination environments. *The International Journal of Robotics Research* 39, 9 (2020), 1052–1060.