WAIS: Leveraging WiFi for Resource-Efficient SLAM

Aditya Arun
University of California, San Diego
San Diego, Califoria, USA
aarun@ucsd.edu

William Hunter
University of California, San Diego
San Diego, Califoria, USA
wshunter@ucsd.edu

Roshan Ayyalasomayajula
University of Buffalo
Amherst, New York, USA
roshana@buffalo.edu

Dinesh Bharadia
University of California, San Diego
San Diego, Califoria, USA
dineshb@ucsd.edu

ABSTRACT

Interest in autonomous navigation and exploration for indoor applications has spurred research into indoor Simultaneous Localization and Mapping (SLAM) robot systems. While most of these SLAM systems use camera and LiDAR sensors in tandem with an odometry sensor, these odometry sensors drift over time. Visual (LiDAR/camera-based) SLAM systems deploy compute and memory-intensive search algorithms to detect 'Loop Closures' to combat this drift, making the trajectory estimate globally consistent. Instead, WAIS (WiFi Assisted Indoor SLAM) demonstrates using WiFi-based sensing can reduce this resource intensiveness drastically. By covering over 1500 m in realistic indoor environments and WiFi deployments, we showcase 4.3× and 4× reduction in compute and memory consumption compared to state-of-the-art Visual and Lidar SLAM systems. Incorporating WiFi into the sensor stack improves the resiliency of the Visual-SLAM system. We find the 90th percentile translation errors improve by ~ 40% and orientation errors by ~ 60% compared with purely camera-based systems. Additionally, we open-source a toolbox, WiROS, to furnish online and compute efficient WiFi measurements.

Codebase: https://github.com/ucsdwcsng/WAIS.git
Dataset: https://forms.gle/XWLLBnWsMct1BRnR8

CCS CONCEPTS

• **Networks** → **Location based services**; • **Computer systems organization** → **Robotics**; **Sensors and actuators**.

KEYWORDS

Wireless Sensing, Compute and Memory Efficiency, SLAM, Localization

ACM Reference Format:
Aditya Arun, William Hunter, Roshan Ayyalasomayajula, and Dinesh Bharadia. 2024. WAIS: Leveraging WiFi for Resource-Efficient SLAM . In *The 22nd Annual International Conference on Mobile Systems, Applications and Services (MOBISYS '24), June 3–7, 2024, Minato-ku, Tokyo, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3643832.3661895

MOBISYS '24, June 3–7, 2024, Minato-ku, Tokyo, Japan

ACM ISBN 979-8-4007-0581-6/24/06
https://doi.org/10.1145/3643832.3661895

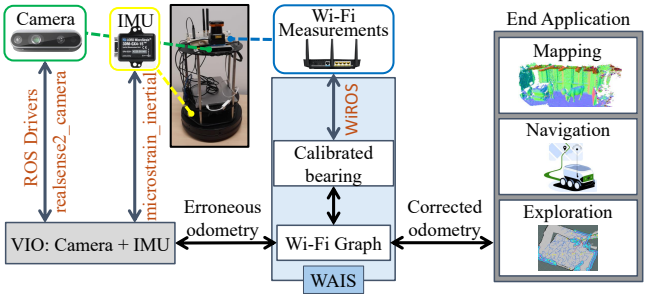

Figure 1: Visual inertial odometry measurements, corrupted by motion drift, are input into WAIS's module. Combining these measurements with Wi-Fi measurements from WiROS provides online corrected odometry measurements [2]. Lidar Inertial odometry or wheel odometry can also be used.

1 INTRODUCTION

Robots have gained traction in diverse indoor applications like inventory management in warehouses, hospitals, and construction sites [35], last-mile delivery [43], and disaster relief [22]. As these robots become more prevalent, it is crucial to ensure their scalability and efficiency. Typically, robots utilize Simultaneous Localization and Mapping (SLAM) frameworks with visual (monocular or 3D cameras) and LiDAR sensors [18, 28, 37, 44] for real-time navigation and mapping their environment[1]. However, these Visual-SLAM systems suffer from two fundamental problems: (a) lack of robustness to visual occlusions and dynamic lighting [29], and (b) an inability to be deployed in resource-constrained scenarios, like VR headsets, drones, or smaller household robots [49].

A few of the recent works [4, 9, 36, 53, 59] solve for robustness using wireless (radio-frequency) sensors for SLAM, as these wireless sensors can see beyond the limited visual field of view. But, these systems do not integrate the wireless signal measurements with the visual measurements while considering resource efficiency. Specifically, a vital piece of a SLAM system called the 'loop closure' module, which is used to correct long-term localization errors, precludes these systems' operation in an 'online' and resource-efficient fashion.

[1] mapping is essential for autonomy making visual sensors indispensable in SLAM systems
[2] "Mapping" image borrowed from [42].

Let us quickly dive into how the current Visual and LiDAR SLAM systems operate, why they need these 'Loop-Closure' modules, and why the 'Loop-Closure' modules limit the efficiency of these systems. The state-of-the-art SLAM systems combine Camera/LiDAR and odometry measurements from either IMUs (inertial measurement units) or wheel encoders to locate themselves and map the environment. Despite this fusion, error in the predicted trajectory increases over time due to the accumulation of sensor errors. Loop closure [3] modules exist within these Visual/LiDAR SLAM algorithms to correct this global motion drift. They ensure a robot revisiting a space predicts a location similar to the previous visit. That is, the robots correlate the current visual observations with past observations, ensure self-consistency of the trajectory, and close the loop by correcting any inconsistencies to correct the trajectory on a global scale. However, these much-needed motion drift corrections are also the weakest links in Visual/LiDAR SLAM systems as they increase the memory requirements, are compute-intensive, and reduce the SLAM systems' reliability. This has been extensively studied previously [3, 49] and Section 3 quantifies this problem further.

In WAIS, we overcome these challenges by designing a wireless sensing-aided SLAM system that is robust and deployable in resource-constrained scenarios. WAIS achieves this using the Dual-Layered design for the Wi-Fi-based global corrections and Visual Sensor-based Local correction modules. WAIS is also designed to be readily integrable into the existing Visual SLAM systems through our sensor front-end WiROS as shown in Fig. 1. Essentially, WAIS + WiROS provides a drop-in replacement for loop-closure systems in third-party SLAM systems. To achieve this, we surmount three challenges in integrating Wi-Fi sensors into SLAM, allowing ready integration of WAIS into existing SLAM systems to correct trajectory drifts without relying on loop closure modules.

1. Compute-efficient integration of Wi-Fi and Visual SLAM: WAIS's first contribution lies in solving for a resource-efficient alternative to loop-closure modules. A loop-closure module achieves these drift-corrections via two independent operations: (a) *identification* of the loop-closure and (b) applying *corrections* to the mapped environment, including the robot locations and any visual landmarks that have accrued errors over time. WAIS proposes using deployed Wi-Fi access points (AP) as landmarks to track and anchor the robot to the environment and correct for its localization drifts. Intuitively, the unique MAC address of APs provides a readily accessible feature identifier, removing the need for feature correlation as needed for visual landmarks, simplifying the *identification* operation.

Once the loop closures are identified, applying the necessary *corrections* is imperative. These drift corrections are needed for (a) '*local drifts*,' that accumulate due to integration errors in wheel odometry, accelerometers, and gyroscopes create trajectory drifts in a span of a few meters, and (b) '*global drifts*'[3], that accumulate over many tens of meters of travel.

Following this distinction, WAIS designs a two-layered system to tackle each error independently. The first layer corrects the local drifts in navigation and mapping using the Visual SLAM algorithms. Here, any open-source SLAM system (with their loop-closure modules turned off) can be used; we evaluate WAIS with Kimera [44] and Cartographer [18]. Additionally, WAIS can work purely with wheel odometry without visual sensors for scenarios where mapping is unnecessary. WAIS then designs the second layer – the Wi-Fi SLAM layer corrects for the global drifts of the navigation path, which automatically corrects for the visually mapped features within the environment. This dual-layered design has three merits: (a) enables real-time corrections of trajectories by decoupling potentially noisy wireless measurements from visual measurements and simplifying the optimization, (b) makes WAIS extensible with existing SLAM systems, and (c) showcases a viable method to integrate other RF and acoustic sensors easily within the SLAM stack. Specifically, we design WAIS to operate with commercially available Wi-Fi radios and enterprise Wi-Fi network deployments to ensure wide use of our system.

2. Real-time Wi-Fi parameter estimation: Designing this Wi-Fi layer has additional challenges. The existing Wi-Fi-based SLAM algorithms only perform end-to-end optimization [4, 19] and do not furnish real-time estimates. There are two key issues with making this system real-time. Firstly, to integrate the Wi-Fi measurements and track the Wi-Fi APs, we compute the angle of arrival (or bearing) at the robot of a received signal from a specific AP. These bearings can be fused with odometer/IMU measurements to correct for the robot drift in the Wi-Fi layer. However, using existing bearing extraction algorithms [4, 27] are compute-intensive, creating a bottleneck for our system. Secondly, before optimization, we need to roughly initialize the location of our Wi-Fi APs (used as landmarks) in our scenario. A poor initialization can lead to a longer convergence time for the optimizer, further reducing the timeliness of our system.

Here, WAIS introduces the second contribution. First, we design a PCA-based Wi-Fi Bearing estimation algorithm. Via a simple observation – multipath reflections, which corrupt the bearing estimations, are random across small motions and can be effectively averaged out over a few packets – we reduce the compute of existing bearing estimations methods by over 200×. Second, we design a smart initialization algorithm that uses the RSSI measurements to predict the approximate AP location, which is further optimized within the Wi-Fi SLAM algorithm.

3. Readily deploying WAIS: Before integration within the Wi-Fi layer, the Wi-Fi sensor measurements must be calibrated and time-synchronized to the measurements from the other sensing modalities. To improve deployability, we provide our third and final contribution. We develop and open source WiROS[4] to provide out-of-box integration with Robot OS (ROS) to furnish time-synchronized measurements with other sensing modalities. Additionally, WiROS's core contribution lies in its wireless calibration framework to estimate and apply real-time phase offsets accrued within the Wi-Fi radio. This calibration is critical to ensure bias-free Wi-Fi-bearing estimation during SLAM operation.

Evaluation highlights: To verify our claims, we have deployed WAIS on a ground robot TurtleBot2 platform that is equipped with a Hokuyo LiDAR and Intel Realsense D455 RGB-D camera with a built-in IMU for deploying cartographer [18] and Kimera [44]

[3]Loop closure provides corrections to these global drifts only when the robot revisits a region in the environment.

[4]https://github.com/ucsdwcsng/WiROS

Table 1: Literature Review: In comparison to the existing wireless, visual, and LiDAR-based SLAM systems, WAIS is the only system that can be real-time, robust to NLOS, does not need 'Loop-Closures,' and any prior knowledge of the environment.

Algorithm	Real-Time SLAM	Robust to NLOS	Do not need 'Loop-Closure'	Doesn't need prior Env. Info.	Integrable into ROS
WSR [22]	✓	✓	✗	✗	✗
P^{2}SLAM [4]	✗	✓	✗	✗	✗
WiDrone [10]	✓	✓	✓	✗	✗
Kimera (VIO) [44]	✓	✗	✗	✓	✓
Cartographer (LIO) [18]	✓	✗	✗	✓	✓
RSSI SLAM [13, 17]	✓	✗	✗	✓	✗
WAIS	✓	✓	✓	✓	✓

respectively. We equip it with a four-antenna off-the-shelf Wi-Fi radio [48] as the Wi-Fi sensor and use WiROS to integrate the Wi-Fi measurements with the ROS framework. We deploy the robot in one large environment to collect data for demonstrating WAIS's compatibility with Kimera's visual-inertial odometry (VIO) outputs. *Additionally*, we use three open-sourced datasets [4] to demonstrate WAIS's deployability with Cartographer. Across these deployments, the robot traverses for an overall time of 108 minutes and a distance of 1625 m. Access points are deployed in the environment with realistic deployment densities in all these datasets [1]. From these evaluations, we observe WAIS

(1) achieves a median translation error of 70.8 cm and a median orientation error of 2.6°, on par with the state-of-the-art Kimera [44] and Cartographer [18]. WAIS's increased robustness by integrating Wi-Fi measurements is seen via the improvements at the 90$^{\text{th}}$ percentile translation and orientation errors by $\sim$ 40% and $\sim$ 60%, respectively.
(2) only needs a total of 0.72 GB whereas Kimera needs 2.82 GB, almost 4× lower. Cartographer and WAIS consume approximately equal amounts of memory.
(3) utilizes, on average, 0.75 fraction of a single CPU core, whereas Cartographer utilizes over 3.2 CPU cores. WAIS is 4× more compute-efficient. Additionally, Kimera consumes $\sim$ 2× more maximum compute during its operation [5].

2 RELATED WORK

WAIS's design motivation is to develop a resource-efficient and online SLAM algorithm. However, other works have considered similar problems, and we discuss them in detail in this section. These existing works span the fields of visual sensor-based SLAM, RF sensors in SLAM, and RF sensor-based localization, mapping, and tracking algorithms.

Table 1 summarizes these works at a high level considering five essential requirements – SLAM systems should produce real-time or online results, their sensing systems should be robust in non-line-of-sight (NLOS) conditions, they should wholly circumvent the need for loop closures, they should be deployable without prior environment information, and they should be readily integrable into ROS to ensure widescale deployment. The rest of the section will carefully explore the current state of the art.

[5]The memory and compute consumption disparity between Kimera and Cartographer is explored in Section 7; it is a consequence of LiDAR features' sparsity compared to Cameras.

Visual SLAM: SLAM for Indoor navigation has been actively pursued by the robotics community for the past decade and has led to a myriad of SLAM algorithms [18, 28, 37, 40, 55]. Most of these algorithms extensively rely on LiDAR [18, 55] and/or cameras (both monocular or RGBD) [28, 37, 40]. While these algorithms achieve cm-accurate localization and feature-rich mapping of the environment, they tend to fail under poor lighting and feature-degenerate conditions. These algorithms also rely on 'loop closures' for larger spaces that require intensive computing and memory resources.

However, some algorithms try to resolve memory issues in loop-closure detection and identification by (a) optimizing key-frame and key-point detection [37] to reduce the number of key-frames stored or (b) using a smart representation, like bag-of-words [44] for efficient storage and retrieval. While these solutions reduce the amount of memory required per step, the underlying problem of linear memory consumption with distance traveled remains. Alternatively, Navion [51] builds an ASIC implementation to explore highly power-efficient VIO systems to provide accurate local odometry. Nonetheless, this system will suffer from the high computing and memory costs of loop closure modules. We foresee future work combining the ideas presented in WAIS and Navion.

Decentralized SLAM: Some Decentralized SLAM works off-load the memory intensive loop-closure and map updates [2, 8, 11] to the cloud or edge devices. However, they rely on explicit loop closures for motion drift corrections. Instead, we observe WAIS's dual-layered design can potentially be incorporated within these works further to reduce the load on the robot's computer.

RF-sensors for SLAM: RF-technology based SLAM implementations based on UWB [38, 53], BLE [23, 47], RFID [32, 33], or backscatter devices [56] exist. However, these RF technologies are not as ubiquitously deployed and have limited ranges compared to Wi-Fi. Additionally, these systems do not provide an extensible framework to incorporate other SLAM works, do not allow simultaneous mapping of the environment, and do not fuse these measurements while keeping resource efficiency in mind. Instead, WAIS's Wi-Fi implementation can be extended to other RF sensors that can measure the signal's incoming angle of arrival [7, 58]. This opens up the possibility of incorporating a large variety of BLE beacons, UWB tags, and other WiFi devices as RF landmarks in SLAM systems.

Wi-Fi-SLAM: There are a few existing works [13, 17, 19, 30, 30] that try to utilize Wi-Fi sensor readings to improve SLAM algorithms. Unfortunately, most of these works use RSSI as the features metric, which needs intensive spatial fingerprinting. RSSI is also known

to vary drastically in dynamic indoor scenarios [34], making them unsuitable metrics for Wi-Fi signals. Extending these RSSI-centric methods to utilize fine-grained phase information used in WAIS would not avoid the need for 'loop-closure' modules. In contrast, WAIS defines more formal Wi-Fi-bearing-based constraints for joint optimization of robot and Wi-Fi AP poses while removing loop-closure modules.

While there are recent works [4, 22] that utilize bearing measurements for their SLAM implementations, WSR [22] does not perform localization but just robot reconnaissance. On the other hand, P^{2}SLAM [4] performs an end-to-end optimization, precluding online SLAM predictions due to the convergence time of the optimizer. Alternatively, [10] needs apriori information like the Wi-Fi anchor's locations within the environment. They additionally require 1000 packets to be transmitted per second to furnish robot location measurements. Alternatively, WAIS collects Wi-Fi packets from the environment at 20 Hz, performs iterative optimization to generate online drift-free trajectory estimates, and does not need apriori information about the environment.

Wireless Calibration: Current works [15, 52] in wireless calibration requires exhaustive search and non-convex optimization over a large number of variables, which could lead to a non-optimal calibration solution or even lead to incorrect results due to local minima in multipath-rich environments. Instead, WAIS re-formulates the calibration estimation as a linear least squares to accurately estimate the calibration values in a convex manner with minimal operational overhead.

3 WHY IS VISUAL SLAM RESOURCE INEFFICIENT?

As we claimed in Section 1, current SLAM systems are memory and compute expensive due to relying on loop closure modules [49]. We first describe the loop-closure modules and then demonstrate their detrimental effect on achieving resource-efficient SLAM through an extensive measurement survey. We will then discuss how existing Wi-Fi SLAM systems fall short of the requirements for the SLAM algorithms.

3.1 Deeper look at loop closures

SLAM algorithms achieve accurate localization and mapping through 'local' and 'global' drift corrections. Most wheeled robots use odometry measurements from a combination of sensors, including IMUs, Wheel Encoders, and Gyroscopes. These odometry measurements are obtained by integrating the measurements from the wheel encoders, gyroscopes, or accelerometers, and they quickly accumulate drift. These integration drifts can lead to trajectory errors of many meters. Visual sensors like LiDAR and Cameras are used to overcome these limitations. The change in positions of distinct visual features is tracked for tens of seconds (tens to hundreds of frames of data) to discern the relative motion of the robot. Fittingly, these corrections are called '*local drift*' corrections. As these visual features are believed to be static, any changes in their position would imply a specific robot motion, which is then estimated and used to correct the drifts accrued by the odometry measurements. However, the trajectory correction applied through feature tracking is still incomplete, and when a robot traverses many hundreds of meters, these incomplete corrections cause trajectory drift on the order of meters. To make corrections for these '*global drifts*,' we can use a simple observation. When the robot revisits a region, its location prediction must be identical to its previous prediction. This observation is realized by the 'loop-closure' modules [3] in SLAM systems and is often the key to their performance over large distances.

A key step in applying loop closures is to detect when a region is revisited. This is done by first identifying distinguishing visual features and storing the locations of these features in the working memory. Second, the currently observed features are correlated with this bank of past observations to detect a loop closure [26, 50]. However, the storage of visual features and repetitive correlation are culprits that increase memory and compute consumption, respectively. Recognizing this drawback, recent works [37, 44] improve upon this naive loop-closure system by employing key-frame selection strategies and utilizing bag-of-words models to reduce the number of frames and the size of features stored in memory. Nonetheless, in the following subsection, we find that these techniques are insufficient.

3.2 Detrimental effects of loop closure modules

To demonstrate these systems' memory and compute requirements, we run a robot across multiple environments spanning about 1.5km of travel distance. The robot collects LiDARs, Cameras, Wi-Fi, and Odometry (IMU) data. These sensing modalities are then fused using Kimera [44], Cartographer [18], P2SLAM [4] and WAIS to get optimized maps and the paths traversed.

Parallely, we record these algorithms' memory and compute consumption and report them as shown in Table 2. Column two mentions the sensors used by each of the algorithms. We care about compute, which is measured as a fraction of the total available compute, to ensure the real-timeliness of our SLAM system. Additionally, the working memory (measured as the rate of increase in RAM consumption) needed to correct the robot trajectories dictates the distance (or the number of hours) of error-free operation.

To evaluate the scalability of current systems to smaller form factor devices, we contrast the resources used by these SLAM algorithms to the available resources on a Raspberry-Pi 4B [14] (a typical 4 GB-RAM and a 4 CPU-core system[6]). The first two rows showcase Kimera's compute and memory consumption with and without their loop closure module operational. In the latter, as expected, we see a 2× worse performance at the median. However, loop closures are responsible for the 3× increase in compute and a 7× reduction in error-free run time, indicating these modules' detrimental effect on the SLAM resource efficiency. Similarly, we can also see that the Cartographer [18] has high computing and memory needs and cannot operate on a single RPI. Visual SLAM systems, which rely on loop closures, cannot scale to lower form factor devices.

3.3 Leveraging Wi-Fi for global corrections

An interesting observation is that the existing Wi-Fi-based SLAM algorithms [4, 19, 22] demonstrate that they do not require loop

[6]Assuming a Raspberry Pi 4B operating at 32-bit Floating point precision and 27 GFLOPS [16]

Table 2: Motivation for WAIS: Accuracy, Memory and Compute Requirements for Various Sensor Combinations

	Sensors Used	Median Accuracy (cm)	Real-Time			Broad Coverage		
			Frame Rate	Compute (GFLOPS)	Fraction of RPI Compute	Memory per Frame (KB)	Memory Rate (Mbps)	Life on RPi4 (Hrs)
Kimera	Camera+IMU	105	15	62.1	2.3	37	0.56	1.98
Kimera w/o LC	Camera+IMU	196.6	15	20.8	0.77	5.3	0.08	13.9
Cartographer	LiDAR+IMU	47.0	40	156.6	5.8	7.5	0.3	3.7
P2SLAM	Wi-Fi+IMU	65.2	20	310.5	11.5	24.5	0.49	2.25
Odom Only	IMU	441.3	100	4.05	0.15	0.09	0.01	111.2
Wi-Fi Only	Wi-Fi+IMU	90	15	12.42	0.46	8	0.16	6.9

closure modules and thus, ideally, should have optimized memory and compute requirements. However, this is not the case due to non-ideal sensor fusion techniques. The recent state of the art, P2SLAM [4], needs at least 11.5 fractions of RPI's (clearly impossible on a single RPI) and can only run for 2.25hrs before it runs out of memory (assuming sufficient compute). In P2SLAM, the authors get the optimized robot path at the end of the robot's navigation, where they perform an end-to-end optimization. This end-to-end optimization may be ideal for applying final corrections to the reconstructed maps, but it makes the system non-real-time and memory-hungry. Thus, one of the key contributions of WAIS is to design a real-time Wi-Fi SLAM (Section 4.2) and a more efficient feature extraction module for the Wi-Fi sensors (Section 5.1) that significantly reduces the memory and compute requirements as shown in the table as, 'Wi-Fi Only.' In the following section, we will demonstrate WAIS's ability to run on a single RPI module in real-time with a 7-hour error-free operation time while also providing 2.5× and 6× improvement over Kimera's trajectory estimates without loop closures and odometry only based dead-reckoning.

4 WI-FI LAYER AND RESOURCE EFFICIENT SLAM

WAIS circumvents the need for loop-closures by leveraging Wi-Fi access points in the environments as much-needed landmarks. Specifically, we seek to optimize the robot's position and heading direction (yaw) in the 3D space [7] by efficiently fusing Wi-Fi measurements and odometry measurements with visual measurements to map the environment simultaneously as well. However, systems [4], which naively fuse Wi-Fi measurements, do not provide the required improvements in computing and memory. Finally, many of the advantages discussed here are obtained by using visual landmarks [20, 39] to anchor the robot to its environment and provide accurate trajectory estimates without loop closures. Similarly, this work takes a significant step in bringing "RF-landmarks" into the purview of the SLAM stack.

WAIS makes three concrete contributions to optimally fuse Wi-Fi measurements to enable resource-efficient SLAM. *First*, it splits the estimation of local and global drift corrections into two independent optimization processes, develops a message-passing framework between the two processes, and reduces the number of variables each optimization encounters (Section 4.1-4.3). This provides a compute-efficient way to incorporate Wi-Fi measurements into a SLAM system. *Second*, WAIS observes angle of arrival (or bearing) estimates of the Wi-Fi signals received at the robot from APs are the most reliable way to track the APs and utilize them as landmarks. However, current bearing estimation frameworks [4, 27] are computed inefficiently. Instead, WAIS develops a new framework for bearing estimation, improving compute efficiency by ∼ 200× (Section 5.1). *Third*, to improve WAIS's deployability, we open-source our Wi-Fi sensing platform WiROS and develop a wireless calibration framework to furnish bias-free bearing estimates in a real-time and time-synchronized manner to WAIS's optimization frameworks (Section 5.2).

4.1 Dual layered optimization

The first idea to integrate Wi-Fi measurements would be to co-optimize them along with visual measurements [31, 45] in a tightly coupled fashion. For instance, by incorporating them within the factor graph of an available SLAM system [44]. Unfortunately, the discovery and addition of new APs and global drift corrections introduce brief periods of instability (order of a few seconds) to the robot's trajectory estimates. These instabilities can introduce large computation overheads as they may also demand corrections to the tracked visual landmarks and map.

Instead, WAIS proposes a dual-layered design to tackle local and global drifts independently. The local drift optimizer corrects the robot's trajectory and builds a map based on visual features. The finer resolution of visual features allows for fine-grained corrections for a few tens of meters. The baton is then passed to WAIS's Wi-Fi graph to handle larger errors due to global drift. This factor graph optimizer relies on the bearings of the incoming Wi-Fi signals [8]. These bearing measurements allow the robot to anchor itself to the environment and adequately correct for the global drifts. But unlike prior work [4, 13], WAIS does not use end-end optimization, instead opts to use incremental smoothing and mapping (iSAM [24]) to provide real-time pose estimates.

4.2 Building the Wi-Fi-Graph

We build the Wi-Fi factor graph discussed in the previous section as shown in Fig. 2. Specifically, consider the state space at time t, S_t. It is a set of robot poses and access point locations over t time steps

[7]The visual map and features built by the VIO/LIO are in 3D space, whereas the robot is constrained to the ground plane

[8]Previous works [4, 19] have shown bearings to be a preferred Wi-Fi-based measurement

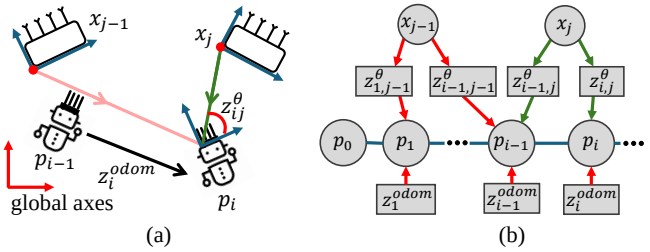

Figure 2: WAIS's Wi-Fi Graph (a) Shows the various measurements made across robot and AP poses. (b) Shows the details of how these measurements are laid as factors in implementing the Wi-Fi Factor graph.

in our graph, $S_t = \{\vec{p}_i = (p_i^t, p_i^q),\ \forall i \in [1,t]\} \cup \{\vec{x}_j,\ \forall j \in [1,N]\}$, with the robot pose (position (p_i^t) + orientation (p_i^q)), $\vec{p}_i \in SE(3)$ and the N access points positions observed till time t, $x_j \in \mathbb{R}^3$. We can define odometry measurements (from VIO/LIO or wheel odometry, as shown by the blue edge in Fig. 2) between poses $\vec{p}_i$ and $\vec{p}_{i+1}$ at two consecutive time steps as:

$$\hat{z}_i^{odom}(\vec{p}_i, \vec{p}_{i+1}) = \begin{bmatrix} R(p_i^q)^{-1}(p_{i+1}^t - p_i^t)^T \\ (p_{i+1}^q \ominus p_i^q)_{[1:3]} \end{bmatrix}$$

where, $R(\cdot) \in SO(3)$ is the rotation matrix corresponding to the given quaternion, $\ominus$ is the relative difference between two quaternions, and $[1:3]$ chooses only the first three elements of the quaternion. Similarly, the bearing factor from AP j at robot position x_i (shown by the red/green edges in Fig. 2)is $\hat{z}_{ij}^{Bearing}(p_i, x_j) \in \mathbb{R}^2$:

$$\begin{aligned} \hat{z}_{ij}^{Bearing}(p_i, x_j) &= \text{Local}(\text{TransformTo}(x_j, p_i), p_i) \\ &= \text{Local}([\cos(\phi)\cos(\theta), \cos(\phi)\sin(\theta), \sin(\phi)]^T, p_i) \end{aligned} \tag{1}$$

where, $TransformTo(\cdot)$ transforms the coordinates of the AP x_j to the coordinate system provided by the robot at p_i, in which the AP subtends an elevation angle of ϕ and an azimuth angle of θ. $Local(\cdot)$ projects this bearing measurement from $SO(3)$ to the tangent plane in $\mathfrak{se}(2)$ defined by the robot's current pose, p_i.

Having defined the measurement models, we can estimate the optimized robot poses S_t^{opt} for time t by minimizing the total error between our predictions and actual measurements,

$$\begin{aligned} S_t^{opt} = \underset{S_t}{\text{argmin}} \sum_{i \le t} \sum_{j \le N} \rho\left((e_{ij}^{bearing})^T \Sigma_{\text{bearing}}^{-1} e_{ij}^{bearing}; c \right) \\ + \sum_{i \in I_{sub}} (e_i^{odom})^T \Sigma_{\text{odom}}^{-1} e_i^{odom} \end{aligned} \tag{2}$$

where $e_{ij}^{bearing} = z_{ij}^{Bearing} - \hat{z}^{Bearing}(p_i, x_j)$ is the bearing factor's error between robot and AP poses i and j; $e_i^{odom}(p_i, p_{i+1}) = z_i^{odom} - \hat{z}_i^{odom}(p_i, p_{i+1})$ is the odom factor's error; ρ is the Huber cost function with parameter c [57]. $\Sigma_{\text{odom}} \in \mathbb{R}^{6\times6}$ and $\Sigma_{\text{bearing}} \in \mathbb{R}^{2\times2}$ are diagonal covariance matrices for odometry and bearing measurements respectively. Further note that the bearings measured (θ) in Section 5.1 provide the azimuth angle in the robot's local frame. We assume that the elevation, ϕ, of the incoming signal is 0 [9], hence $z_{ij}^{Bearing} = [\cos(\theta),\ \sin(\theta)]$.

4.3 Initialization of Factors

Before optimizing Eq. (2), we need to initialize the t robot and N AP positions. Similar to prior works [4], the robot positions are initialized using relative odometry measurements. These poses have accumulated drift over time, which we seek to correct.

Additionally, as new APs are observed in the environment, they must be initialized within the factor graph. Still, unlike Wi-Fi localization systems [27, 54], we assume the location of the AP is *unknown*. A naive initialization for the APs would be to place them at the origin and allow the optimizer to place them appropriately in the environment [4]. Unfortunately, as shown in Fig. 3(a), an AP initialization error of more than 5 m leads to the optimizer failure. Due to the limited number of Wi-Fi-bearing measurements observed, the iSAM optimizer fails to converge in real-time with largely incorrect initialization.

Alternatively, Wi-Fi's received signal strength indicator (RSSI) has been studied extensively for localization and provides a general sense of proximity [46], accurate to within 5 m. Thus, we can identify the robot's position $p_{ap-j}^T \in \mathbb{R}^3$ at which the j^{th} AP's RSSI measurement has a maxima inflection point among all current robot's poses $p_i \forall i \in [1,t]$, with the intuition that this is where the robot passed closest to the AP. We can then initialize the j^{th} AP's pose x_j as $x_j = p_{ap-j}^T + \Delta$, where $\Delta \in \mathbb{R}^3$ is a small $(\le 0.1m)$ random perturbation. This random perturbation is added to avoid in-determinancy with the optimizer. Across hundreds of optimization trials of the 13 unique AP locations present within the datasets, we found this AP initialization scheme allows for reliable real-time convergence of the iSAM optimizer.

5 REAL-TIME WI-FI MEASUREMENT PROCESSING

As mentioned in Section 4, the lack of compute-efficient bearing estimation frameworks and wireless calibration methods still preclude the widespread adoption of Wi-Fi sensing for SLAM. In this section, we delineate two contributions in WAIS. First, a resource-efficient and accurate bearing estimation algorithm, PCAB (Section 5.1), and second, automated wireless calibration techniques to furnish bias-free bearing estimates (Section 5.2). Additionally, to enable broader adoption of Wi-Fi for SLAM, we open source these contributions in addition to incorporating our Wi-Fi sensor with the ROS framework to provide real-time and time-synchronized Wi-Fi measurements [10].

5.1 Efficient Bearing Estimation

In indoor environments, the primary reason for inaccuracies in the bearing measurements is multipath [6, 27, 34]. A Wi-Fi signal, with wavelength λ, is broadcast, and multiple reflections of the signal

[9] AP's placed at differing heights, leading to non-zero elevation angles, has little effect on the azimuth bearing estimation [4].

[10] WiROS's Index page: https://github.com/ucsdwcsng/WiROS;
Sub links: CSI Node: https://github.com/ucsdwcsng/wiros_csi_node;
Feature-extraction Node: https://github.com/ucsdwcsng/wiros_processing_node;
Custom RF messages: https://github.com/ucsdwcsng/rf_msgs

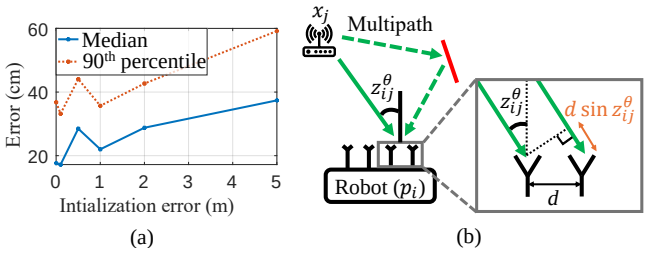

Figure 3: (a) Impact of AP initialization errors: APs have been initialized at varying errors up to 5 m compared to ground truth. The median and 90^{th} percentile errors have been plotted. The optimizer fails to converge above 5 m of error. (b) Wi-Fi bearings (θ) measured using a linear antenna array with antenna separation d. The additional distance $d\sin(\theta)$ traveled by the signals can be exploited to estimate the bearing

(*multipath*) along with the direct path, impinge at the receiver as shown in Fig. 3 (b). The 'direct path' (solid line) is the only path helpful in estimating the bearing (θ) to the source, and the rest of the 'reflected paths' (dotted lines) are the cause for erroneous bearing estimates.

To understand these effects, we provide a simple mathematical model. The receiver measures, at time t, a complex-valued channel state information (CSI) describing the phase delay and attenuation across each of the M_{ant} receiver antennas and N_{sub} orthogonal frequencies [27] as

$$X_t^{m,n} = a_{m,n} e^{j\phi_m(\theta)} e^{-j(2\pi f_n \tau + \psi)}; \quad X_t \in \mathbb{C}^{M_{\text{ant}} \times N_{\text{sub}}}$$
$$\phi_m(\theta) = -\frac{2\pi}{\lambda} md \sin(\theta) \tag{3}$$

where $a_{m,n}$ is the attenuation; d is the antenna separation for the linear antenna array; f_n is the orthogonal frequency subcarriers; τ is the time-of-travel of the signal, which is often corrupted by the random ψ phase offset due to lack of transmitter-receiver clock synchronization; and $\phi_m(\theta)$ is the additional phase accumulated at the m^{th} antenna (Fig. 3(b)) due to the additional distance traveled by the signal. The various reflected paths impinging on this linear array will add to each antenna and frequency component in a similar manner.

To identify the direct signal's path among the multiple reflected paths, past Wi-Fi systems[34] have used algorithms that 'super-resolve' the measured signal. They measure the relative time offset between different signal paths using information across these N_{sub} different frequency bins (or subcarriers). The direct path can then be readily resolved as the path with the least time offset, as it travels the shortest straight-line path to the receiver. However, this extra dimension of time offset adds computation overhead, making them unsuitable for resource-efficient SLAM algorithms.

So, in WAIS, we reduce the dimensionality of our problem and yet reliably segregate the direct path signals from the clutter of the reflected paths. Specifically, from Eq. (3), the largest eigenvector ($U_t \in \mathbb{C}^{M_{\text{ant}}}$) of $X_t X_t^H \in \mathbb{C}^{M_{\text{ant}} \times M_{\text{ant}}}$ provides the largest contribution to the channel measured across the M receive antennas.

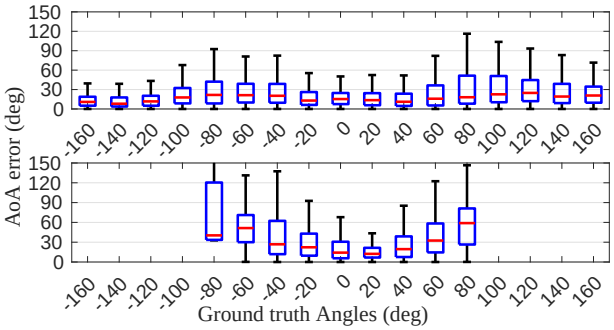

Figure 4: Bearing errors are accumulated in steps of 10° for the the square antenna array (*Top*) and linear array (*Bottom*).

However, this largest component could potentially be corrupted by multipath. To remove the effect of multipath, we make a simple observation – reflected paths are susceptible to small changes in the robot's position as opposed to the direct path, which will arrive at a consistent bearing. Hence, we can effectively 'average out' multipath effects from our bearing estimation if we combine our measurements across time (over multiple packets). We find the largest eigenvector over the most recent T measurements across the past $0.5-1$ seconds, during which our robot has moved negligibly:

$$U_t = \boldsymbol{\lambda}\Big(\sum_{i=t-T}^{t} X_i X_i^H\Big),$$

Where $\boldsymbol{\lambda}(\cdot)$ extracts the largest eigenvector, which is trivial to compute for our $M_{\text{ant}} \times M_{\text{ant}}$ autocorrelation matrix. From here, our direct path signal U_t can be mapped to a bearing by a coarse search over the space of possible bearings,

$$\theta^* = \underset{\theta \in [-90^\circ, 90^\circ]}{\text{argmax}} \sum_{i=1}^{M_{\text{ant}}} \phi_m(\theta) U_t(m) \tag{4}$$

This method, dubbed Principle Component Analysis based Bearing (PCAB), allows us to achieve similar bearing estimation performance to the standard 2D-FFT algorithm [4] and Spotfi [27] at just a fraction of the compute. Specifically, PCAB reduces the problem to just the antenna domain, ignoring the frequency-subcarrier domain information. This provides an $N_{\text{sub}}\times$ speedup compared to 2D-FFT (for example, for an 80Mhz Wi-Fi signal, consisting of $N_{\text{sub}} = 234$ subcarriers, this provides an $\approx 200\times$ speedup).

Resiliency to NLOS: We have made an implicit assumption in PCAB that a direct path signal from an access point is present. But, in cases where a large reflector blocks the direct path, no information can be obtained about the direct path's bearing, leading to inconsistent bearing measurements. However, overall received signal power (RSSI) is typically very low in these situations, as most of the signal has been blocked. Hence, we reject all measurements with RSSI below -65dBm, which we empirically observe filters out most of these obstructed packets. This RSSI threshold is environment and device-independent [4] but manufacturer-dependent as different access point manufacturers report signal strength relative to different baselines.

Unambiguous Bearing Measurements: The uniform linear arrays used by existing bearing estimation algorithms [4, 27] and

shown in Fig. 3(b) are vulnerable to aliasing - they can only measure bearings in a 180° range (*i.e.* the top half plane in Fig. 3(b)) at a time, as there is no distinction between signals coming from opposite sides of the array. This limitation is reflected in the search space of bearing angles in Eq. (4). Furthermore, as shown in Fig. 4(bottom), these arrays have poorer accuracy when Wi-Fi signals arrive nearly parallel to the array (near ±90). To maximize the range of measured angles, we resolve this ambiguity by adopting a square antenna array, which does not suffer from aliasing as seen from Fig. 4(top). However, we cannot extend the range to the entire 360° due to ambiguity present for Wi-Fi signals arriving from behind the robot. Transitioning to a square array changes the differential phases measured (Eq. (3)) as,

$$\phi_m(\theta) = -\frac{2\pi}{\lambda}(a_m^x \cos(\theta) + a_m^y \sin(\theta)), \tag{5}$$

the relative position of antenna m is (a_m^x, a_m^y) with respect to the first antenna. Hence, WAIS finds a trade-off between resolution and aliasing to resolve angles from $-160°$ to $160°$.

5.2 Quick and easy calibration

The bearing measurements computed in Section 5.1 will be erroneous without appropriately calibrating the Wi-Fi sensor. Calibrating our Wi-Fi sensor should remove any measurement bias and be easy to perform. Generally, the calibration can vary for different Wi-Fi channels and is unique for each hardware. Hence, WAIS develops an easy-to-deploy and accurate calibration framework for the tractability of our Wi-Fi sensor for SLAM applications.

We apply independent phase corrections across each antenna and frequency measurement to calibrate our raw Wi-Fi measurements, as given by the phase calibration matrix C,

$$C = \exp(j\Phi) \in \mathbb{C}^{M_{\text{ant}} \times N_{\text{sub}}},$$

across M antennas and N frequencies, where $\Phi \in \mathbb{R}^{M_{\text{ant}} \times N_{\text{sub}}}$. Given a raw CSI measurement from the CSI Node, $X_t \in \mathbb{C}^{M_{\text{ant}} \times N_{\text{sub}}}$, the calibration is applied as

$$X_t^{\text{cal}} = C \odot X_t,$$

where $\odot$ is the Hadamard (element-wise) product.

We deploy the robot in a relatively multipath-free environment. Using the robot poses ($\vec{p}_t = (p_t^x, p_t^y, p_t^\theta) \in SE(2)$) and transmitter location ($\vec{z} \in \mathbb{R}^2$), we first compute the expected ground truth bearings (θ_t). These can then be converted to expected Wi-Fi CSI measurements ($\hat{X}_t \in \mathbb{C}^{M_{\text{ant}} \times N_{\text{sub}}}$).

$$\theta_t = \frac{\pi}{2} - \left(\arctan\left(\frac{p_t^y - z^y}{p_t^x - z^x}\right) - p_t^\theta\right);\ \hat{X}_t^{i,j} = \exp(j\phi_i(\theta_t))$$

where, ϕ_i is the relative phase accumulated at the i^{th} antenna from Eq. (3) or Eq. (5). λ is the wavelength of the center frequency for the Wi-Fi channel in consideration. Note that we assume this expected measurement has equal response across the frequency subcarriers as we assume time-of-flight is zero.

Assuming a strong line-of-sight path signal is present in our measurements, we can expect the phases $\angle\hat{X}_t \approx \angle X_t^{\text{cal}}$. Note that our assumption is reasonable as the calibration data is collected in a relatively open environment with no blockages to the signal. Consequently, we can suppress the phase difference induced by bearings as

$$X_t^{\text{sup}} = X_t \odot \text{conj}(\hat{X}_t)$$

This leaves the remaining calibration phase C in X_t^{sup}. However, each Wi-Fi measurement may have multiple reflected paths and hardware-centric Gaussian noise. To suppress these noise terms, we can recall the hints previously discussed in Section 5.1. One, reflections are inconsistent across different locations; two, averaging can suppress Gaussian noise. Hence, the best calibration estimate is the strongest remaining component in the suppressed X_t^{sup} measurements. We can leverage Principle Component Analysis, as in PCAB (Section 5.1), to extract this strongest component in our calibration data as

$$\begin{aligned} X_t^{\text{flat}} &= \text{flatten}(X_t^{\text{sup}}), \quad X_t^{\text{flat}} \in \mathbb{C}^{M_{\text{ant}} N_{\text{sub}}} \\ \mathbb{W} &= \begin{bmatrix} W_0^{\text{flat}} & W_1^{\text{flat}} & \cdots & W_T^{\text{flat}} \end{bmatrix} \\ \mathbb{U}, \mathbb{S}, \mathbb{V} &= \text{SVD}(\mathbb{X}) \\ \Phi^{\text{coarse}} &= \angle\text{reshape}(\mathbb{U}_0), \quad C^{\text{coarse}} = \exp(j\Phi^{\text{coarse}}) \end{aligned}$$

where 'flatten' converts the matrix into a vector, SVD computes the full singular-value decomposition, 'reshape' converts the vector back into a matrix of the original dimensions, and $\angle$ computes the phase of the complex numbers. $\mathbb{U}_0$ is the first and strongest principal component of $\mathbb{X}$. However, as indicated, this calibration is a coarse estimate. The expectation is for the calibration matrix to consist of unit-norm *elements*, but the principle component, $\mathbb{U}_0$, is a unit-norm *vector*, violating this property. Hence, we need to re-project C^{coarse} onto a valid space of calibrations. To find a valid calibration, C^{fine}, that is close to C^{coarse}, we note that C^{fine} must be orthogonal to the other vectors in $\mathbb{U}$, so we try to find a fine-tuned calibration Φ^{fine} which has the lowest norm when it is projected onto $\mathbb{U}_{[1:]}$.

$$\Phi^{\text{fine}} = \min_{\Phi} ||\mathbb{U}_{[1:]}^T \text{flatten}(\exp(j\Phi))||_2^2;\ C^{\text{fine}} = \exp(j\Phi^{\text{fine}})$$

where $\mathbb{U}_{[1:]}$ is the orthogonal space, and we minimize for Φ using Levenberg-Marquardt [25] algorithm with an initialization of Φ^{coarse}. The wireless phase calibration matrix C^{fine} is recovered through this fine-tuning. We will evaluate the accuracy and versatility of this calibration in Section 7.

6 IMPLEMENTATION

We use the following setup to deploy the dual-layered Wi-Fi and VIO/LIO graph optimization system.

Hardware: We deploy WAIS on the Turtlebot 2 platform. We attach an off-the-shelf Wi-Fi radio [5] to the Turtlebot for Wi-Fi CSI data collection. Then, we place a few similar APs near the ceiling at a density of roughly one every ten meters. We transmit on channel 42 of 5GHz Wi-Fi. We then use WiROS (discussed in Section 5) to collect channel state information (CSI) data from all the APs at the Wi-Fi radio on the robot. Our robot also has a Hokuyo UTM-30LX LiDAR and an Intel D455 RGBD camera with an IMU.

Software: The Turtlebot is controlled by a laptop running the Robot Operating System (ROS-Kinetic), which manages all sensor

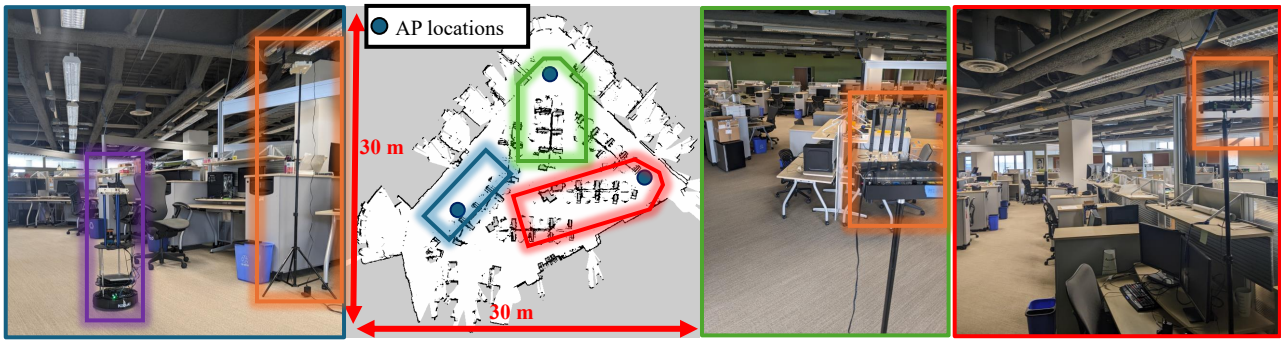

Figure 5: WAIS is evaluated on realistic office environments as shown here. The colored outlines for the images indicate the viewpoint in the top-down LiDAR map. The deployed AP's and robot are highlighted in the orange and purple box respectively.

input. Wi-Fi CSI measurements are also integrated into ROS, and WAIS is implemented as a C++ application to ensure real-time operation and integrated within ROS as a ROS node, allowing for easy integration with other robotics systems. The Wi-Fi factor graph is implemented in GTSAM [24], and we utilize the open-sourced library to implement Kimera and Cartographer as the Visual/LiDAR Inertial Odometry measurements for the Wi-Fi graph described in Section 4.2. Kimera [44] is the state-of-the-art VIO that uses the Intel D455 + IMU for its SLAM, and Cartographer [18], a state-of-the-art LIO that uses the wheel encoders and the Hokuyo LiDAR for its SLAM. An Intel Core i7-9750H CPU with 12 cores and 32 GB of RAM is used for all evaluations and data collection.

7 RESULTS

We demonstrate WAIS's performance in 4 datasets (3 public datasets [4] and one WAIS's dataset) and compare it with state-of-the-art SLAM systems Kimera [44] and Cartographer [18]. Across these datasets, we compare three aspects of each algorithm's performance:
(a) 3-DoF Navigation Accuracy: XY translation error and absolute orientation error,
(b) Memory Consumption[11] The total memory consumed for a run of the dataset, and also the rate of memory consumption, to understand scalability to larger indoor spaces, and
(c) Compute Cost: The maximum and average number of compute cores required by the algorithm to perform online SLAM.
Datasets: To demonstrate our system against Kimera (which requires a stereo-camera), we deploy a robot in a 30 × 30 m environment with 3 Wi-Fi APs, placed near the ceiling, 2 m above the robot's Wi-Fi sensor. The environment is shown in Fig. 5. The robot traverses a total distance of 403 m over 23 minutes (called WAIS DS[12]). We also use the open-sourced datasets [4] that we call DS 1/2/3. But, these three datasets do not have stereo-camera data, and we cannot deploy Kimera on them. Instead, we compare these datasets with Cartographer. They also use a linear antenna array on the robot, so we resolve the AoA aliasing (see Section 5.1) using ground truth information. In all the datasets, APs are deployed at least 5 m apart, in a realistic fashion [1]. Additionally, the environment has interfering Wi-Fi networks, active Wi-Fi devices, and dynamic people movement, ensuring a natural deployment setting.

Through these four datasets, we have thoroughly evaluated WAIS across three distinct environments, four different and realistic access point placements, and a cumulative traversed distance of 1625 m with a total travel time of 108 minutes. We allow Cartographer to run offline with extensive search parameters to obtain ground truth location and orientation labels, which we have observed converge to a near ground truth trajectory as characterized in DLoc [6].
Baselines: We compare WAIS's performance with (a) Kimera in the WAIS DS and (b) Cartographer in all the datasets. We have finetuned the "loop-closure" module parameters for Kimera and Cartographer to obtain the best real-time performance.

WAIS's modular design enables the Wi-Fi-graph to receive odometry measurements from different SLAM systems. Thus, we give WAIS the following sources of *local* odometry:

(a) Wi-Fi+VIO: Visual inertial odometry (VIO) from online Kimera *without* the Loop Closure detection node. Specifically, we use Stereo-camera features along with IMU to furnish local odometry, *without* performing loop closures,

(b) Wi-Fi+LIO: Lidar inertial odometry (LIO) from online Cartographer without global scan matching. Specifically, we use Lidar scan matching on a frame-to-frame basis fused with wheel odometry, *without* performing loop closures for global correction, and

(c) Wi-Fi+Odom: The local odometry is provided only by the built-in wheel encoder and the gyroscope on the TurtleBot.

We compare Kimera (with loop closures turned on and tuned to our best ability) with Wi-Fi+VIO and Cartographer (with global scan matching turned on and tuned to our best knowledge) with Wi-Fi+LIO. Wi-Fi+Odom is independently characterized across the board. Additionally, we turned off visualization on Kimera and Cartographer to accurately measure the resource consumption of the optimization backend and keyframe storage. Specifically, we ensure that only the tasks of robot localization and, when enabled, loop closure detection and loop closure corrections are performed over the entire trajectory of the robot.

We provide an in-depth analysis of results for Kimera Section 7.1.1 and summarize the results for Cartographer in a table in Section 7.1.2 due to space constraints. Section 7.2 demonstrates our baselines'

[11] computed with https://github.com/alspitz/cpu_monitor
[12] this dataset can be accessed at https://forms.gle/XWLLBnWsMct1BRnR8

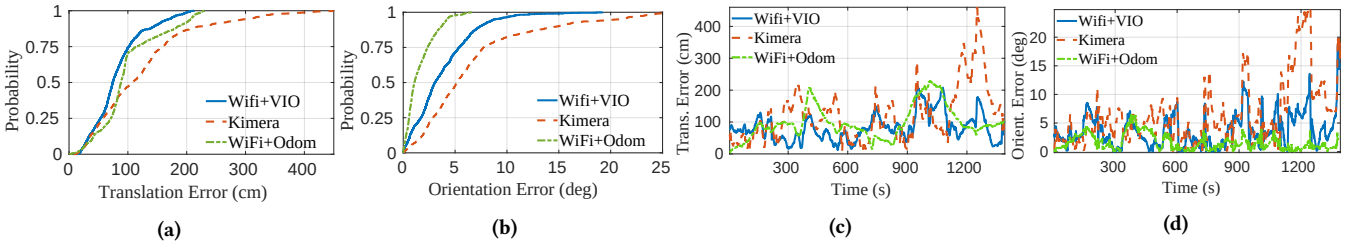

Figure 6: End-to-end evaluation: (a, b) Translation and orientation errors comparing Kimera with loop closure against WAIS's predictions. WAIS uses odometry from Kimera without loop closures and just wheel odometry. (c, d) Time series of translation and orientation error comparing Kimera with loop closure against WAIS's predictions. WAIS uses odometry from Kimera without loop closures.

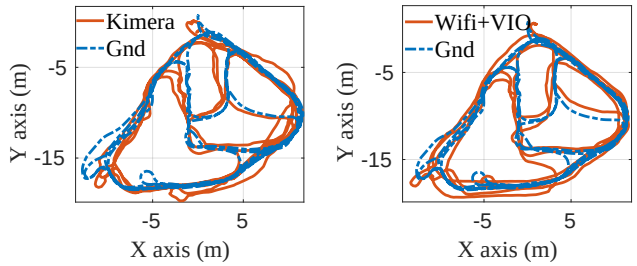

Figure 7: Trajectories: A top-down view showing the Trajectories estimated by (a) Kimera with loop closures and ground truth. (b) WAIS using odometry predictions from Kimera without loop closures (*Wi-Fi + VIO*) and ground truth.

compute/memory trade-offs for different loop-closure parameters and the accuracy of wheel odometry as compared to VIO and LIO. Finally, Section 7.3.1 evaluates WiROS's wireless calibration and bearing prediction algorithms.

7.1 End-to-end evaluation

7.1.1 WAIS with Kimera.

First, we show how WAIS provides a resource-efficient alternative to Kimera's loop closure detection module with on-par or better navigation accuracy.

Navigation Accuracy: We compare the CDF and time-series translation errors in Fig. 6(a, c) for WAIS's Wi-Fi+VIO, Wi-Fi+Odom fusion and Kimera. From these plots we can see that the median (90^{th}%) translation errors for WAIS's Wi-Fi+VIO is 85cm (185cm), and Wi-Fi+Odom is 90cm (205cm), which are 60% lower compared to Kimera with median (90^{th}%) translation errors of 105cm (300cm). Similarly, Fig. 6(b, d) compares the cumulative and time-series orientation errors. From these plots we can see that the median (90^{th}%) orientation errors for WAIS's Wi-Fi+VIO is 3° (8°), and Wi-Fi+Odom is 1° (4°), i.e. 60% lower compared to Kimera with median (90^{th}%) orientation errors of 5° (20°). Here, we make two interesting observations. First, compared to translation error, pairing WAIS with Turtlebot's in-built wheel odometry and gyroscope performs better than pairing WAIS with VIO. This is because orientation predictions are inherently better with wheel odometry than VIO, providing a better initialization for WAIS's optimizer. Second, the time series graphs show five distinct peaks in Kimera's error. The reductions in errors happen when we close a loop within our trajectory. However, for "Wi-Fi+Odom" and "Wi-Fi + VIO," we frequently see trajectory corrections when the robot is in view of an AP, regardless of revisiting the environment.

While the median translation and orientation errors for Kimera are comparable to WAIS running on Kimera's odometry, the reason for high errors in the 90^{th}% are due to an incorrect loop-closure that occurs in Kimera at around 1100 sec time-mark as can be seen from the sudden spike in errors in both Fig. 6(c,d). This can be seen in the two top-down views of the estimated trajectories shown in Fig. 7. This further demonstrates the strength of Wi-Fi measurements for global corrections.

Memory Consumption: We have seen that Wi-Fi provides more accurate and real-time global drift corrections than Kimera's loop closures. Now, we evaluate the memory consumption of the individual components of Kimera and WAIS and observe the trends shown in Fig. 8 (left). Kimera needs a start-up memory of 2 GB and, from then on, accumulates memory at a rate of 0.56 MBps. In contrast, Wi-Fi + VIO only has a start-up memory of 0.4 GB and a memory accumulation rate of 0.2 MBps, which will, on average, enable WAIS to run 3.3× longer than Kimera on a typical RAM of 8 GB before crashing. We can further verify that Kimera's memory consumption is dominated by loop closures – the black line in Fig. 8 (left) for the VIO, which is Kimera without loop-closures, indicates the memory remains constant at 0.2 GB. It is important to note here that since Kimera's VIO has near-constant memory consumption, the accruing memory consumption of WAIS's stack is purely due to the Wi-Fi-graph as shown by the purple dashed line in Fig. 8 (left). This is a consequence of storing the past trajectory in memory as we build the Wi-Fi graph. However, if the full robot trajectory is not necessary for downstream applications, a fixed-lag smoother [21] can be employed to optimize memory consumption further.

Compute Requirements: Finally, we can see from Fig. 8 (right) that WAIS requires only one core to run on our system, while Kimera takes up to 1.5 cores owing to its loop closure detection and correction algorithms. Thus, WAIS's Wi-Fi+VIO design requires about 3.3× less memory and 1.5× less compute than a state-of-the-art Kimera system while achieving about 60% better navigation performance.

7.1.2 WAIS with Cartographer.

Table 3: Translation, Orientation, and resource analysis; median (99^{th} percentile in parenthesis) errors reported

	Cartographer [18]				Wi-Fi + LIO			
	WAIS DS	**DS 1**	**DS 2**	**DS 3**	**WAIS DS**	**DS 1**	**DS 2**	**DS 3**
Translation Error (cm)	**47.0 (98.9)**	74.0 (224)	134.1 (1097)	**23.3 (37.4)**	50.34 (152.4)	**65.9 (92.2)**	**67.3 (182.6)**	48.6 (114.4)
Orientation Error (°)	4.8 (7.9)	**0.8 (4.66)**	5.3 (12.6)	**0.6 (1.4)**	**3.5 (6.9)**	2.4 (4.66)	**2.8 (12)**	1.4 (3.3)
Total Memory (MB)	520	702	**658**	423	**486**	**613**	706	**356**
Rate of Memory (MBps)	**0.30**	0.29	0.33	0.38	0.32	**0.22**	**0.33**	**0.26**
CPU (fraction of cores)	3.8 (4.4)	3.2 (4.2)	3.8 (4.2)	1.85 (4.4)	**0.73 (1.90)**	**0.62 (2.1)**	**0.85 (2.8)**	**0.7 (1.1)**

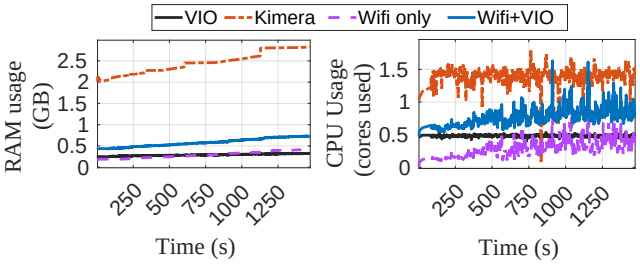

Figure 8: Timeseries of memory (*left*) and CPU consumption (*right*) of VIO, Kimera and Wi-Fi + VIO. The resource consumption of the WAIS's Wi-Fi graph is analysed (*Wi-Fi only*)

We also test WAIS running on Cartographer's LIO and compare it online with full-stack Cartographer across all the four datasets, DS 1/2/3 and WAIS DS, and present a summary of the results in Table 3 and boldface the better-performing metrics.

Navigation Accuracy: From this table, we can see that WAIS's trajectory estimation performs on par with Cartographer's loop closure detector across most datasets in terms of translation and orientation accuracy. There are two notable differences in navigation accuracy performance:

(a) in *DS 2* WAIS's (Wi-Fi+LIO) median (90^{th}) translation and orientation errors are 2× (5×) lower than Cartographer. Cartographer makes an incorrect loop closure in this scenario, leading to a very inconsistent trajectory prediction, as can be common with visual-based loop closures.

(b) in *DS 3*, Cartographer has 2× lesser translation and orientation errors than Wi-Fi+LIO. We notice this performance degradation from using a linear array on the robot. As discussed in Section 5.1, bearing measurements closer to ±90° suffer from higher errors. A non-optimal orientation of the linear array on the robot further exacerbates the problem, leading to a larger number of Wi-Fi signals received parallel to the antenna array. Utilizing a square array resolves these issues.

Memory Consumption: WAIS and Cartographer have similar memory consumption. This occurs because, unlike Kimera's VIO system, where each camera frame consists of dense features, Cartographer's LIO system's 2D LiDAR scan features [55] are much smaller, reducing Cartographer's memory consumption.

Compute Requirements: Due to the sparsity of LiDAR features, Cartographer needs to run extensive scan-matching algorithms to detect and apply loop closures, increasing Cartographer's compute requirements. In contrast, WAIS demands fewer compute resources since WAIS running on Cartographer's local odometry does not require scan matching to correct the global trajectory. This can be observed in the last row of Table 3, which shows the average number of cores used over the entire run of the robot navigation with the maximum number of cores used in parentheses. We can see that WAIS needs 4× lesser peak compute than the full-stack Cartographer implementation across all four datasets.

7.2 WAIS's Microbenchmarks

VIO Memory vs Accuracy: To characterize Kimera's memory usage, we alter the rate at which it records keyframes for loop closure detection. A lower time between keyframes will lead to more loop closure detections due to increased memory consumption. To understand how Kimera's loop closure detection accuracy is limited by memory, we plot the median translation error and the total memory consumed in WAIS DS (Fig. 9(a)). From this plot, we can see that the best median translation error with reasonably low memory consumption occurs at a keyframe period of 1 second, and we use this keyframe rate for our baseline.

LIO Compute vs Accuracy: To understand Cartographer's performance, we first note that 2D LIO systems have sparse features in LiDAR scans when compared to denser stereo data. This sparser representation demands extensive scan-matching algorithms [18] with higher compute. Thus, to understand Cartographer's computing requirements, we run Cartographer on WAIS DS and limit the number of CPU cores it can access. We plot the median translation accuracy vs the number of CPU cores accessible in Fig. 9(b). We can see that median translation error improves with more cores, verifying that compute power is a bottleneck for accurate scan matching and, thus, navigation performance. Since many low-compute platforms are limited to 4 cores [41], we restrict the Cartographer to 4 compute cores in the end-to-end comparison.

Local Odometry Estimates: Now, let us compare how accurate the odometry measurements from Kimera and Cartographer without loop closures, VIO and LIO, respectively, are to wheel odometry provided by the Turtlebot. Fig. 9(c) shows the comparison of their translation errors, from which we can see that the odometry measurements from Kimera and Cartographer without loop closure have lesser errors at both the median and 90^{th}% than odometry from wheel-encoders. This supports the dual-graph design intuition of WAIS wherein the odometry from the VIO/LIO-graph is locally corrected and improves on pure dead-reckoning.

7.3 Characterizing Wi-Fi measurements

In the following section, we will verify the performance of the bearing estimation (Section 5.1) and the efficacy of our wireless calibration technique (Section 5.2).

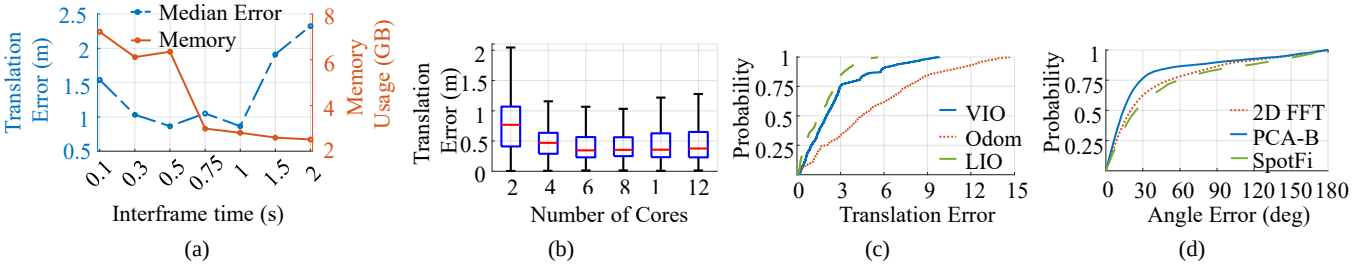

Figure 9: Microbenchmarks: (a) Kimera: Trade-off between memory consumption and the accuracy for various interframe rates. (b) Cartographer: Trade-off between the compute required and the accuracy for various scan-matching thresholds. (c) Odometry-Graph: Comparison of how various local odometry inputs the Wi-Fi graph. (d) Bearing Measurement Algorithms: Comparison between 2D-FFT from P^{2}SLAM, Spotfi and WAIS's PCAB.

7.3.1 *Bearing Estimation accuracy.*

To understand the accuracy and compute requirements of bearing-estimation algorithms and the need for PCAB (Section 5.1), we compare it with *2D-FFT* defined in P2SLAM [4] and *Spotfi* [27]. As shown in Fig. 9(d), PCAB outperforms 2D-FFT in bearing estimation by 1.43× (2×) at the median and 80$^{\text{th}}$%. Additionally, we note that the average CPU utilization of PCAB on our robot is 11%, whereas 2D-FFT required 565% core usage (an improvement of approx. 50×). Moreover, we compute the AoA predictions with Spotfi [27] and find PCAB performs 1.8× (2.2×) at the median and 80$^{\text{th}}$%. Moreover, Spotfi consumes 1200% of cores (all cores of our machine) and fails to run in real-time. Both the 2D-FFT and Spotfi estimations are unsuitable for low-compute SLAM applications.

7.3.2 *Calibration efficacy.*

We must effectively correct hardware biases and offsets to measure accurate bearings. Specifically, we observe that without appropriate calibration, our median bearing errors can be 115°. However, post calibration, we can expect median bearing errors of 5.3°. Hence, we provide a wireless calibration system described in Section 5.2 as a core part of WAIS. However, this technique must work consistently for different Wi-Fi channels and upon device reset.

We collect the data as explained in Section 5.2 to confirm the calibration performance. We place an additional sensor in the environment sending beacon packets and another on the robot receiving them. We run the robot in a random pattern, collecting CSI data within *bag* files, as shown in Fig. 10(a).

Fig. 10(b) shows bearing errors across different channels after the APs were reset post collecting the required training data. For this experiment, we measure calibration for different channels and test the quality of these calibrations by collecting a test dataset after restarting the AP by characterizing the bearing accuracy. We observe similar performance across all the 80 MhZ Wi-Fi channels [12]. These calibration properties are also specific to the Asus hardware used [5]. However, the wireless calibration procedure provided is agnostic to hardware.

8 DISCUSSION AND FUTURE WORK

In this work, we have demonstrated WAIS, and its modular design can integrate into any existing VIO/LIO systems to remove the need for compute and memory-intensive loop-closures. Thus, WAIS provides a framework that can enable accurate and resource-efficient

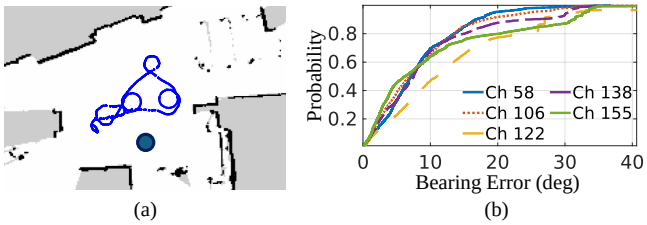

Figure 10: (a) Robot path and transmitter location (blue circle) to calibrate the robot's Wi-Fi sensor in 3×3 m, (b) Bearing errors when calibrated across different channels, after reset and channel switching

SLAM for indoor robotics. However, there are still some core limitations of WAIS's framework which open up multiple avenues for future work:

(a) Highly NLOS scenarios: We anticipate WAIS will fail in scenarios with highly NLOS measurements. As WAIS relies on the presence of LOS-bearing, in the presence of NLOS, most of these WiFi packets would be incorporated erroneously into the optimizer. LOS packets are also essential for AP initialization, as mentioned in 2.d above. Poor AP initialization can lead to optimization problems. This implies that WAIS's advantages cannot be utilized.

(b) Single antenna radios: We have seen how different antenna array geometry on the robot provides varied results. However, many small-factor devices may not have space to accommodate multiple antennas. Extending the work to perform single-antenna-based Wi-Fi-SAR [22] would make the system more scalable.

(c) Multi-robot systems: While WAIS demonstrates efficient SLAM for a single robot in the environment, extensions to collaborative SLAM for a fleet of robots present additional challenges, making compute and memory efficiency even more critical.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Hang Qiu, for their insightful feedback. Additionally, we are grateful for the feedback from the WCSNG group members.

REFERENCES

[1] 2021. High density wi-fi deployments. https://documentation.meraki.com/Architectures_and_Best_Practices/Cisco_Meraki_Best_Practice_Design/Best_

Practice_Design_-_MR_Wireless/High_Density_Wi-Fi_Deployments.
[2] Fawad Ahmad, Hang Qiu, Ray Eells, Fan Bai, and Ramesh Govindan. 2020. CarMap: Fast 3D Feature Map Updates for Automobiles.. In *NSDI*. 1063–1081.
[3] Saba Arshad and Gon-Woo Kim. 2021. Role of deep learning in loop closure detection for visual and lidar slam: A survey. *Sensors* 21, 4 (2021), 1243.
[4] Aditya Arun, Roshan Ayyalasomayajula, William Hunter, and Dinesh Bharadia. 2022. P2SLAM: Bearing based WiFi SLAM for Indoor Robots. *IEEE Robotics and Automation Letters* (2022).
[5] ASUS. 2022. Asus RT-AC86U router. https://www.asus.com/us/networking-iot-servers/wifi-routers/asus-wifi-routers/rt-ac86u/, journal=ASUS USA.
[6] Roshan Ayyalasomayajula, Aditya Arun, Chenfeng Wu, Sanatan Sharma, Abhishek Rajkumar Sethi, Deepak Vasisht, and Dinesh Bharadia. 2020. Deep learning based wireless localization for indoor navigation. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.
[7] Roshan Ayyalasomayajula, Deepak Vasisht, and Dinesh Bharadia. 2018. BLoc: CSI-based accurate localization for BLE tags. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. 126–138.
[8] Ali J Ben Ali, Marziye Kouroshli, Sofiya Semenova, Zakieh Sadat Hashemifar, Steven Y Ko, and Karthik Dantu. 2022. Edge-SLAM: Edge-assisted visual simultaneous localization and mapping. *ACM Transactions on Embedded Computing Systems* 22, 1 (2022), 1–31.
[9] Tara Boroushaki, Laura Dodds, Nazish Naeem, and Fadel Adib. 2022. FuseBot: RF-Visual Mechanical Search. *Robotics: Science and Systems 2022* (2022).
[10] Guoxuan Chi, Zheng Yang, Jingao Xu, Chenshu Wu, Jialin Zhang, Jianzhe Liang, and Yunhao Liu. 2022. Wi-drone: wi-fi-based 6-DoF tracking for indoor drone flight control. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 56–68.
[11] Titus Cieslewski, Siddharth Choudhary, and Davide Scaramuzza. 2018. Data-efficient decentralized visual SLAM. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2466–2473.
[12] IEEE Computer Society LAN/MAN Standards Committee et al. 2007. IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11ˆ* (2007).
[13] Brian Ferris, Dieter Fox, and Neil D Lawrence. 2007. Wifi-slam using gaussian process latent variable models.. In *IJCAI*, Vol. 7. 2480–2485.
[14] Raspberry Pi Foundation. [n. d.]. Teach, learn, and make with the Raspberry Pi Foundation. https://www.raspberrypi.org/
[15] Wei Gong and Jiangchuan Liu. 2018. RoArray: Towards more robust indoor localization using sparse recovery with commodity WiFi. *IEEE Transactions on Mobile Computing* 18, 6 (2018), 1380–1392.
[16] VMW Research Group. 2023. The GFLOPS/W of the various machines in the VMW Research Group. https://web.eece.maine.edu/~vweaver/group/green_machines.html Accessed: 2023-11-29.
[17] Zakieh S Hashemifar, Charuvahan Adhivarahan, Anand Balakrishnan, and Karthik Dantu. 2019. Augmenting visual SLAM with Wi-Fi sensing for indoor applications. *Autonomous Robots* 43, 8 (2019), 2245–2260.
[18] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. 2016. Real-time loop closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1271–1278.
[19] Joseph Huang, David Millman, Morgan Quigley, David Stavens, Sebastian Thrun, and Alok Aggarwal. 2011. Efficient, generalized indoor wifi graphslam. In *2011 IEEE international conference on robotics and automation*. IEEE, 1038–1043.
[20] Jiunn-Kai Huang, Shoutian Wang, Maani Ghaffari, and Jessy W Grizzle. 2021. LiDARTag: A real-time fiducial tag system for point clouds. *IEEE Robotics and Automation Letters* 6, 3 (2021), 4875–4882.
[21] Vadim Indelman, Stephen Williams, Michael Kaess, and Frank Dellaert. 2012. Factor graph based incremental smoothing in inertial navigation systems. In *2012 15th International Conference on Information Fusion*. IEEE, 2154–2161.
[22] Ninad Jadhav, Weiying Wang, Diana Zhang, Oussama Khatib, Swarun Kumar, and Stephanie Gil. 2020. WSR: A WiFi sensor for collaborative robotics. *arXiv preprint arXiv:2012.04174* (2020).
[23] Maani Ghaffari Jadidi, Mitesh Patel, Jaime Valls Miro, Gamini Dissanayake, Jacob Biehl, and Andreas Girgensohn. 2018. A radio-inertial localization and tracking system with BLE beacons prior maps. In *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 206–212.
[24] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. 2012. iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research* 31, 2 (2012), 216–235.
[25] Carl T Kelley. 1999. *Iterative methods for optimization*. SIAM.
[26] Giseop Kim and Ayoung Kim. 2018. Scan Context: Egocentric Spatial Descriptor for Place Recognition within 3D Point Cloud Map. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Madrid.
[27] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. 2015. SpotFi: Decimeter Level Localization Using Wi-Fi *(SIGCOMM)*.
[28] M. Labbé. 2018. RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation.
[29] Pierre-Yves Lajoie, Siyi Hu, Giovanni Beltrame, and Luca Carlone. 2019. Modeling perceptual aliasing in slam via discrete–continuous graphical models. *IEEE Robotics and Automation Letters* 4, 2 (2019), 1232–1239.
[30] Ran Liu, Sumudu Hasala Marakkalage, Madhushanka Padmal, Thiruketheeswaran Shaganan, Chau Yuen, Yong Liang Guan, and U-Xuan Tan. 2019. Collaborative SLAM based on Wifi Fingerprint Similarity and Motion Information. *IEEE Internet of Things Journal* 7, 3 (2019), 1826–1840.
[31] Bruce D Lucas, Takeo Kanade, et al. 1981. *An iterative image registration technique with an application to stereo vision*. Vol. 81. Vancouver.
[32] Zhihong Luo, Qiping Zhang, Yunfei Ma, Manish Singh, and Fadel Adib. 2019. 3D backscatter localization for fine-grained robotics. In *NSDI*. Boston, Massachusetts, 765–782.
[33] Yunfei Ma, Nicholas Selby, and Fadel Adib. 2017. Drone relays for battery-free networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 335–347.
[34] Yongsen Ma, Gang Zhou, and Shuangquan Wang. 2019. WiFi sensing with channel state information: A survey. *ACM Computing Surveys (CSUR)* 52, 3 (2019), 1–36.
[35] Nathan Melenbrink, Justin Werfel, and Achim Menges. 2020. On-site autonomous construction robots: Towards unsupervised building. *Automation in construction* 119 (2020), 103312.
[36] Piotr Mirowski, Tin Kam Ho, Saehoon Yi, and Michael MacDonald. 2013. SignalSLAM: Simultaneous localization and mapping with mixed WiFi, Bluetooth, LTE and magnetic signals. In *International Conference on Indoor Positioning and Indoor Navigation*. IEEE, 1–10.
[37] 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.
[38] Thien-Minh Nguyen, Shenghai Yuan, Muqing Cao, Thien Hoang Nguyen, and Lihua Xie. 2021. VIRAL SLAM: Tightly Coupled Camera-IMU-UWB-Lidar SLAM. *arXiv preprint arXiv:2105.03296* (2021).
[39] Edwin Olson. 2011. AprilTag: A robust and flexible visual fiducial system. In *2011 IEEE international conference on robotics and automation*. IEEE, 3400–3407.
[40] 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.
[41] Raspberry Pi Ltd. 2023. *Raspberry Pi 5*. Raspberry Pi Ltd. https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf.
[42] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto. 2020. Voxgraph: Globally Consistent, Volumetric Mapping Using Signed Distance Function Submaps. *IEEE Robotics and Automation Letters* (2020).
[43] All The Research. Oct 2021. Autonomous Last Mile Delivery Market by Platform - Global Forecasts 2021 to 2027. https://www.alltheresearch.com/report/787/autonomous-last-mile-delivery-market Accessed: 2022-03-23.
[44] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. 2021. Kimera: From SLAM to spatial perception with 3D dynamic scene graphs. *The International Journal of Robotics Research* 40, 12-14 (2021), 1510–1546.
[45] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *2011 International conference on computer vision*. Ieee, 2564–2571.
[46] Sebastian Sadowski and Petros Spachos. 2018. Rssi-based indoor localization with the internet of things. *IEEE access* 6 (2018), 30149–30161.
[47] Akihiro Sato, Madoka Nakajima, and Naohiko Kohtake. 2019. Rapid BLE beacon localization with range-only EKF-SLAM using beacon interval constraint. In *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 1–8.
[48] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. 2017. Nexmon: The C-based Firmware Patching Framework. https://nexmon.org.
[49] Sofiya Semenova, Steven Y Ko, Yu David Liu, Lukasz Ziarek, and Karthik Dantu. 2022. A quantitative analysis of system bottlenecks in visual SLAM. In *Proceedings of the 23rd Annual International Workshop on Mobile Computing Systems and Applications*. 74–80.
[50] Tixiao Shan and Brendan Englot. 2018. LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4758–4765.
[51] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. 2019. Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones. *IEEE Journal of Solid-State Circuits* 54, 4 (2019), 1106–1119.
[52] Ju Wang, Jie Xiong, Hongbo Jiang, Xiaojiang Chen, and Dingyi Fang. 2017. D-watch: Embracing "bad" multipaths for device-free localization with cots rfid devices. *IEEE/ACM Transactions on Networking* 25, 6 (2017), 3559–3572.
[53] Weiying Wang, Anne Kemmeren, Daniel Son, Javier Alonso-Mora, and Stephanie Gil. 2022. Wi-Closure: Reliable and Efficient Search of Inter-robot Loop Closures Using Wireless Sensing. *arXiv preprint arXiv:2210.01320* (2022).
[54] Jie Xiong and Kyle Jamieson. 2013. ArrayTrack: A Fine-grained Indoor Location System *(NSDI)*.

[55] Ji Zhang and Sanjiv Singh. 2014. LOAM: Lidar odometry and mapping in real-time.. In *Robotics: Science and systems*, Vol. 2. Berkeley, CA, 1–9.
[56] Shengkai Zhang, Wei Wang, Sheyang Tang, Shi Jin, and Tao Jiang. 2020. Robot-assisted backscatter localization for iot applications. *IEEE Transactions on Wireless Communications* 19, 9 (2020), 5807–5818.
[57] Zhengyou Zhang. 1997. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and vision Computing* 15, 1 (1997), 59–76.
[58] Minghui Zhao, Tyler Chang, Aditya Arun, Roshan Ayyalasomayajula, Chi Zhang, and Dinesh Bharadia. 2021. ULoc: Low-Power, Scalable and cm-Accurate UWB-Tag Localization and Tracking for Indoor Applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 3 (2021), 1–31.
[59] Han Zou, Chun-Lin Chen, Maoxun Li, Jianfei Yang, Yuxun Zhou, Lihua Xie, and Costas J. Spanos. 2020. Adversarial Learning-Enabled Automatic WiFi Indoor Radio Map Construction and Adaptation With Mobile Robot. *IEEE Internet of Things Journal* 7, 8 (2020), 6946–6954. https://doi.org/10.1109/JIOT.2020.2979413