MultiEdge-SLAM: Realizing ORB-SLAM for Multi-Edge Computing

Muhammad Ahmed*
Firas Abdullah*
24100027@lums.edu.pk
24100032@lums.edu.pk
Department of Computer Science
SBASSE, LUMS
Lahore, Punjab, Pakistan

Adil Rahim Hyder
Department of Computer Science
SBASSE, LUMS
Lahore, Punjab, Pakistan
24100049@lums.edu.pk

Muhammad Raahim Nadeem
Department of Computer Science
SBASSE, LUMS
Lahore, Punjab, Pakistan
24100072@lums.edu.pk

Abstract

Simultaneous Localization And Mapping (SLAM) systems have previously been adapted for edge-based deployment in Edge-SLAM. They are widely used in robotics and computer vision applications to determine the position and orientation of a robot or a camera while simultaneously building a map of the environment. However, the system therein does not account for mobile endpoints switching between edges, presenting performance bottlenecks that we are able to illustrate via baseline tests. We propose a new design based on Edge-SLAM that seeks to enable seamless state migration across edge devices and have begun work on implementing these changes in the Edge-SLAM codebase.

CCS Concepts: • **Computer systems organization** → **Cloud computing**; **Client-server architectures**; **Embedded systems**; **Real-time system architecture**; • **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → **Multi-agent systems**; **Computer vision**.

Keywords: Visual simultaneous localization and mapping, edge computing, split architecture, mobile systems, localization

ACM Reference Format:
Muhammad Ahmed, Firas Abdullah, Adil Rahim Hyder, and Muhammad Raahim Nadeem. 2023. MultiEdge-SLAM: Realizing ORB-SLAM for Multi-Edge Computing. In *Proceedings of Topics in Internet Research '23 (CS 678 Spring'23).* ACM, New York, NY, USA, 9 pages. https://doi.org/3090567.5789793

*Both authors contributed equally to this research.

1 INTRODUCTION

1.1 Description

In the modern era, Edge Computing [1] has driven various innovations with regards to remote processing for applications, and this subject area has been a popular area for further research [2]. The primary benefit of Edge Computing is that expensive computations can be effectively offloaded to a nearby edge server. This allows devices on the lower end of the processing power spectrum to access many services that they would not be able to deliver locally, and that too, with significantly less latency [3]. These services encompass a range of augmented reality (AR) and virtual reality (VR) applications that utilize 3D map creation and device tracking technologies to perceive the environment around them, resulting in highly immersive or pseudo-immersive user experiences. Examples of such applications include Google's ARCore [4] and automotive AR solutions such as Apostera [5]. At the heart of these applications is a class of visual Simultaneous Localization and Mapping (SLAM) algorithms [6], which have been modified for the edge computing sphere through advancements such as Edge-SLAM [7], SLAM-Share [8], and SwarmMap [9]. However, the state of the art has not yet produced an efficient solution for edge-synchronization to facilitate mobile endpoints.

This is a pressing issue as several modern use cases for Visual SLAM involve fast-moving endpoints (such as surveillance drones [15] or self-driving cars [16]) that may, by necessity, have to cover a wide area of space. In such cases, connectivity with a single edge node may not be conducive to low-latency performance or, in some cases, feasible altogether. This is due to the toll of maintaining a connection with a distant edge node or the cost of establishing a remote-computation relationship with a new edge node. Via experiments outlined below, we were able to simulate these hand-offs and illustrate the performance bottlenecks they pose.

1.2 Motivation

Currently, advanced edge-based processing designs such as Edge-SLAM perform all significant computations (across modules that we describe below) at the server-side [7], while others like SLAM-Share [8] and SwarmMap [9] aim to consolidate multiple local views of the environment into one Global

Map. This is due to the limited computational capacities of the endpoints in question (since many AR applications, for example, target a smartphone user base). All these proposals, however, are designed for single endpoint-edge connections. For the purposes of this paper, we focused specifically on Edge-SLAM, which is an edge-based revision of the ORB-SLAM2 system.

We propose MultiEdge-SLAM, a new design meant to encompass multiple edges as an endpoint moves between them, providing a high-level overview of various design choices and their motivations. Work on implementing this design is ongoing, and although we have realized many aspects of it on a conceptual level, much still needs to be done for practical realization.

2 DISCUSSION OF RELATED WORKS

This section is meant to provide an overview of the relevant subject matter that we studied during the literature review portion of the project timeline.

2.1 An Introduction to ORB-SLAM

Visual SLAM algorithms are designed for Mapping and Localization. In this context, Mapping refers to the process of using notable features (usually corners) from a visual view of an environment to build a map of that environment. In contrast, Localization refers to the act of estimating an agent's location inside that map. With that said, ORB-SLAM [13] is one of the most popular Visual SLAM algorithms that has received consistent updates as modern-day application demands have evolved [14][10].

The system can be broken down into three modules that operate concurrently on a shared global map.

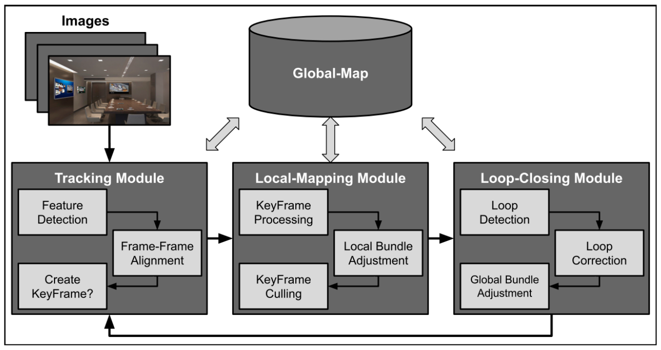

Figure 1. ORB-SLAM System Architecture

2.1.1 Module One: Tracking.

As can be seen in Figure 1, the process begins when a video feed from a single camera is passed to the tracking module in the form of a sequence of images (or frames). This module uses ORB (Oriented FAST and Rotated BRIEF) feature detection [17] to identify distinctive features in each frame, particularly corners, whilst encoding descriptors that make them invariant to rotation and scale. This is a crucial optimization for better recall of features, especially when they are viewed from varying distances and angles in subsequent frames.

After feature extraction, the camera's pose (position and orientation) is estimated with the previously tracked frame (by applying a velocity motion model to approximate how the camera could have been displaced and rotated from the previously tracked frame). If the motion model gets violated (i.e, the camera moves in an unpredictable way), estimation fails as the module is not able to find enough map point correspondences to verify its approximation.

In such an event, tracking is said to be lost, forcing the module to take the brute-force route of Global Relocalization. This involves referring to the Global Map and scanning neighboring keyframes to locate which one is most similar to the current frame (via comparison of Word Histograms constructed for each keyframe with DBoW2 [20]). It hence relocalizes with respect to that keyframe (whose pose and orientation is already known).

After successful pose estimation, the system decides whether the current frame should be stored as a point of reference based on the distinctiveness of its features. If the frame is distinct enough (based on a set of criteria that includes the count of unique features and time elapsed since relocalization), it is passed to the Local Mapping module where it is made into a keyframe.

2.1.2 Module Two: Local Mapping.

The Local Mapping module adds the keyframes to a Covisibility graph and an Essential Graph which make up the Global Map. The Covisibility graph represents how "covisible" two keyframes are (i.e to what extent are they viewing the same part of an environment, or, more specifically, map points); the vertices denote keyframes while the edges between them denote covisibility (quantified in the edge's weight). On the other hand, the Essential Graph is fundamentally a "Highest Covisibility" spanning tree extracted from the Covisibility Graph.

The Covisibility graph is used for tasks such as map optimization and relocalization, while the Essential Graph is used for the triangulation of new map points. Ultimately, the system builds a reliable and accurate 3D map by enforcing strict quality control measures for map points during creation and retention. After creating new map points it performs optimizations such as Local Bundle Adjustment [18], ensuring that the points selected are mathematically refined. It also culls redundant keyframes to maintain a compact reconstruction. These processes ensure accurate and consistent results across different kinds of environments.

2.1.3 Module Three: Loop Closing.

Finally, the Loop Closure module actively performs Place Recognition (i.e., it checks to see if the camera is viewing a part of the environment that has previously been visited) whenever a keyframe is added. This is vital as otherwise, the system would perceive previously visited regions as unexplored parts of the environment.

When loop closure is detected, the system calculates similarity transformations to perform Loop Fusion, aligning the two views of the same object by minimizing and distributing the error along the Essential Graph. This error reflects the uncertainty in measurements garnered from the camera, which, if left unchecked, will increasingly skew our map.

It is also worth noting that loop closure must be triggered via a robust series of steps (since it is better to miss a loop closure event than to detect a false positive event that would skew our map permanently). Thus, at least three candidate keyframes should correspond to the current keyframe, both temporally and geometrically, in order for the module to infer a loop closure event.

2.2 Edge-SLAM: Adapting ORB-SLAM to the Edge

ORB-SLAM is a robust tool for visual mapping and localization, but the modules described above involve many computationally expensive operations that are infeasible to perform on low-powered devices like smartphones. As such, Edge-SLAM [7] modifies the ORB-SLAM2 (an optimized version of the foundational ORB-SLAM system) architecture to address this constraint and offload computation to the edge.

As all three modules share and operate on a Global Map, splitting the architecture across two devices is quite challenging. Nevertheless, Edge-SLAM imposes three main changes in order to accomplish this.

2.2.1 Moving the Tracking Module to the Device. First, it decouples the modules such that tracking is performed on the endpoint while the Global Map and remaining modules are processed at the edge device. This architecture reconfiguration raises the question of how the tracking module will perform localization without access to the Global Map.

2.2.2 Introducing the Local Map. The aforementioned concern is addressed by Edge-SLAM's second contribution: a new data structure known as the Local Map, which is situated at the endpoint. This Local Map is a subset of the Global Map and contains keyframes relevant to the endpoint's current location. As we have established, the endpoint will be moving through the Global Map, and hence its Local Map needs to be frequently updated and synchronized in response to its mobility.

2.2.3 Facilitating Two-Way Communication. The requirement for frequent synchronization is the basis of Edge-SLAM's third major contribution: the introduction of three TCP-based communication streams between the Tracking and Local Mapping modules.

- **Local Map Updates (Edge to Device):** Local Map updates originate from the Global Map and are sent to the device for synchronization in accordance with the device's current position.
- **KeyFrames (Device to Edge):** Recall that the Tracking module only makes the decision to create KeyFrames while the Local Mapping module performs the necessary operations to facilitate that. Since these two modules exist on separate devices, a communication stream is necessary to supply to-be-created keyframes to the Local Mapping module.
- **Frames (Device to Edge):** Recall that if the Tracking module is unable to localize the current frame with respect to the previously tracked frame, then Global Relocalization is performed with respect to the Global Map. In this case, since the Tracking module lacks the Global Map, it must forward frames that it could not track to the edge for Global Relocalization.

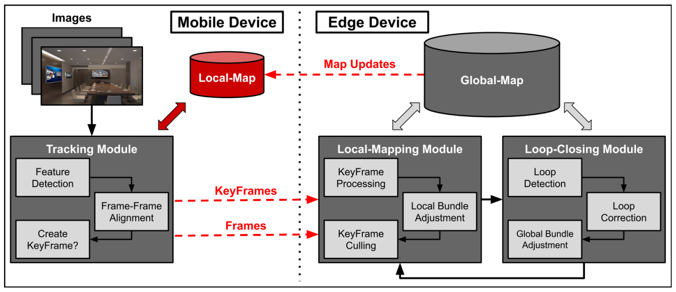

Figure 2. Edge-SLAM System Architecture (modifications shown in red)

By making these three modifications to the ORB-SLAM pipeline, this system effectively enables a robust visual SLAM architecture for Single-Edge Computing. However, in the case of mobile endpoints that connect to multiple edges, there are limitations to this system. In the case where a mobile endpoint connects to a new edge after dropping a connection to a newly-distant edge, full reinitialization is required on the next edge as it has no map data to begin with (even though map data is present at the previously connected edge). The cost of reinitialization is significant on mapping accuracy as tracking is completely paused until the reinitialization process is successfully completed.

An intuitive hypothesis would be that this represents a bottleneck in terms of accuracy and performance, as the process of SLAM would have to be paused and reset after each new edge connection. To verify this, we conducted some tests that are described in section 4 of this paper.

3 DESIGN

As discussed earlier, the reinitialization of state in the event of a change in edge devices poses a significant bottleneck in terms of accuracy. Thus, some means of migrating state across edges would allow us to enable a seamless handover between them and eliminate the costs of reinitialization, thus reducing latency in tracking and yielding more mapping accuracy.

Our primary objective is to enable a seamless handover from one edge to another, achieved by eliminating the initialization cost on the subsequent edge. By doing so, we aim to maximize the accuracy of mapping while ensuring a smooth transition between edges. For this, we present three schemes, each of which involves the migration of SLAM state from one edge of the network to the other as the endpoint moves between them. The three schemes differ in the approach taken for state migration, and each has its tradeoffs. We analyze the tradeoffs of each scheme and discuss their effectiveness with intentions to ultimately and robustly evaluate them on a wide collection of datasets in terms of mapping accuracy.

3.1 Edge Sends a Subset of the Global Map

In this scheme, the previous edge sends a **subset of its Global Map** to the next edge (this makes the assumption that edges are able to communicate with each other with relatively low latency). Once the currently connected edge determines that a specific endpoint is about to connect to the next edge (via the localization it performs for that endpoint), it will select the k latest keyframes from its Global Map and start sending portions of the Covisibility and Essential Graph containing said keyframes to the next edge.

It's important to note that the edge will select and transmit only the k latest keyframes. This is necessary since the next edge will require access to the area tracked during the period in which the endpoint is transitioning from one edge to the other. By sending over the latest keyframes, we only select the keyframes tracked during this transition, which is vital in order to ensure a seamless handoff and the continuation of smooth tracking. The specific number of keyframes selected is a parameter that may be fine tuned through the process of experimentation.

One major advantage of this scheme is that the state migration mechanism is now offloaded to the two edges. The user device does not have to maintain another TCP connection with the next edge to migrate state and does not have to run any extra threads to migrate state. Moreover, the number of messages that need to be transmitted in order to migrate state is also relatively low (as the edge devices will have more bandwidth in this regard). Fine tuning may even allow for just a single message to be transmitted between edges.

One potential drawback of such a scheme is the possibility of an outdated map being created on the next edge. This is feasible since the global map created on the next edge may be outdated by the time the edge to edge update is complete, as the endpoint may have tracked more ground during the transmission. Since these newly-tracked keyframes were not a part of the Global Map when migration was triggered, they will not be present on the next edge. Hence, when the user device connects to the next edge it will be tracking using an outdated global map. Additional map updates from the user device to the edge will be required in order to update the map fully. Moreover, the state update message from one edge to the other will be relatively complex in nature since it will consist of a Covisibility and Essential Graph.

3.2 User Device Sends Keyframes

In this scheme, the mobile device sends **keyframes to the next edge** once it has confirmed that it is nearing the point of transition. By sending those keyframes, the endpoint ensures that it will have access to an initial map upon connection to the next edge. However, during the transitory period in which the endpoint is nearing the change in edges, it will continue tracking and localizing using the previous edge.

The endpoint will hence have to send keyframes to both the previous edge and the next edge during this transitory period. The next edge receives keyframes from the endpoint and builds a partial Global Map. Its behavior here would be similar to the edge behavior defined by Edge-SLAM but would skip the step in which the edge sends map updates to the endpoint (since the previous edge will be fulfilling that requirement).

In this scheme, the map created on the next edge will be guaranteed to be up to date on the endpoint's progress through the global map, since it will be receiving keyframes directly from the endpoint. Here, the individual messages sent to migrate state would include only keyframes; hence, the complexity of individual messages would be very low.

However, the map created on the next edge will not benefit from the optimizations performed on the global map by the previous edge. This is obvious since the next edge is only receiving keyframes from the mobile device and does not have access to all the data that the previous edge was using to perform Bundle Adjustment or Loop Closing. As a result, the map created at the next edge will be somewhat inaccurate compared to the map that was present at the previous edge.

Furthermore, the number of messages that would need to be transmitted in this scheme in order to migrate state would also vary. If the endpoint generates a large number of keyframes during the transition across edges, the number of keyframes sent to the next edge will be significant. This may lead to network congestion since the endpoint will be sending keyframes to both the previous edge as well as the next edge, which will incur a cost in terms of performance as well.

3.3 User Device Sends Local Map

In this scheme, the mobile device sends its **local map to the next edge** once it has confirmed that it is nearing the point of transition. At this junction, the endpoint will start transmitting its Local Map to the next edge, allowing the next edge to construct a Global Map from the transmitted data.

In this scenario, the Global Map created on the next edge will be partially in sync with the endpoint. Moreover, since the endpoint's Local Map will include the updates from the previous edge (via the update process), the optimizations performed to produce those updates will be effectively transferred to the Global Map being constructed at the next edge. The number of map updates (in the form of messages) that will need to be sent to the next edge will also be lower as compared to the amount required when transferring keyframes from endpoint to edge.

It is important to note, however, that the process of consolidating multiple local map updates into a single Global Map at the next edge will require a highly sophisticated algorithm that is not part of Edge-SLAM's design. Previously, since the edge was receiving keyframes, it could operate normally. However, in such a scheme the edge will be receiving complete maps with potentially disparate structures (due to the optimizations performed by the previous edge). Accommodating these divergent maps into a generalized and accurate global view on an absolute coordinate system would be a non-trivial task.

Such a scheme could be potentially optimized by configuring the previous edge to send the local map update not only to the user device but also the next edge as well. However, this may fall into the same pitfall as previous schemes in which we end up with an outdated map on the next edge.

4 EVALUATION

4.1 Experimental Setup

Concurrent to the process of preparing a high-level design, our team also worked on deploying Edge-SLAM on a local system and familiarizing ourselves with the codebase (which comprises 15,543 lines of C++ code across 10 modules) [21], with the goal of running baseline tests to verify our hypothesis.

Before that, however, we familiarized ourselves with deploying and running an ORB-SLAM system (specifically ORB-SLAM3) on our local machines. That process was described in our midterm report and ended in us having to request remote server access in order to be able to run even basic tests. The server access proved invaluable as the authors of Edge-SLAM deployed their system using a DELL XPS desktop—Intel Core i7 9700 K, NVIDIA GeForce GTX 1080, 32 GB Memory—running Ubuntu 18.04 LTS (to emulate the edge device) and a NVIDIA JETSON TX2—64-bit NVIDIA Denver (to emulate the endpoint). Our plan was to use the server to emulate both the edge device and the endpoint, but even with the increased power, we faced several issues on the path to deployment.

The first issue was related to compatibility issues. Edge-SLAM is built on top of the codebase for ORB-SLAM2 [22], and is hence incompatible with newer versions of the involved libraries. Furthermore, linking the several libraries involved was a very time-consuming process (since Edge-SLAM involves Pangolin [23], Eigen [24], OpenCV [25], Boost, and the Robot Operating System (ROS), many of which conflicted with each other across different versions).

After we resolved these dependency issues, we were able to deploy the system on the server but observed the CPU shooting to > 100% utilization after running for TUM's freiburg2_desk [11] dataset. This resulted in tracking being lost repeatedly across the Tracking and Local Mapping modules, and left us with very few keyframes (<5) in our final mapped trajectory (while the expected number of keyframes was around 100 [19]).

We then pivoted to deploying the system on one of our own systems which is equipped with a GPU, and this presented its own roadblock as the version of ROS that Edge-SLAM requires is only compatible with version 18.04 of the operating system Ubuntu (due to a conflict with OpenCV), while the version installed on our machine was 20.04 (which we had to downgrade). We did not face this issue while deploying on the server as the server has Ubuntu 18.04 installed.

Meanwhile, we persisted in our attempts to run Edge-SLAM on the server. To handle the CPU constraints, we turned to the ROS .bag file that plays the dataset for the Edge-SLAM instance (i.e. feeds it to the system). The running of this file made up the majority of our CPU consumption, so we attempted to reduce its footprint. We were able to do this by reducing the amount of data that the .bag file is responsible for, which appears in the form of information feeds called Topics. We removed all the topics that ORB-SLAM2 would not utilize (such as IMU data), which left us with two feeds: Depth and RGB data. We included these two topics in a new .bag file and used that to play the dataset instead.

While this did not greatly reduce our CPU usage, it did allow the parts of the dataset that we require to consume more processing power in their respective threads. We were essentially able to provide a more relevant dataset to the system and this resulted in more functional runs of Edge-SLAM that did not consistently lose tracking state. While we are now seeing a larger number of keyframes being collected in the mapped trajectory, the number is still quite variable due to the massive CPU footprint of the entire process (for the freiburg2_desk dataset, we have ranged across 110 and 45 keyframes across successive runs).

4.2 Baseline Test and Observations

We first needed a control experiment that communicates the drop in accuracy if a user device starts tracking on a new Edge without any prior state. To carry out such an experiment, we needed to induce an initialization phase on the edge between tracking.

To this end, we used the in-built reset functionality of Edge-SLAM. The user device would trigger reset on the edge on an arbitrary frame number (that we calibrated to

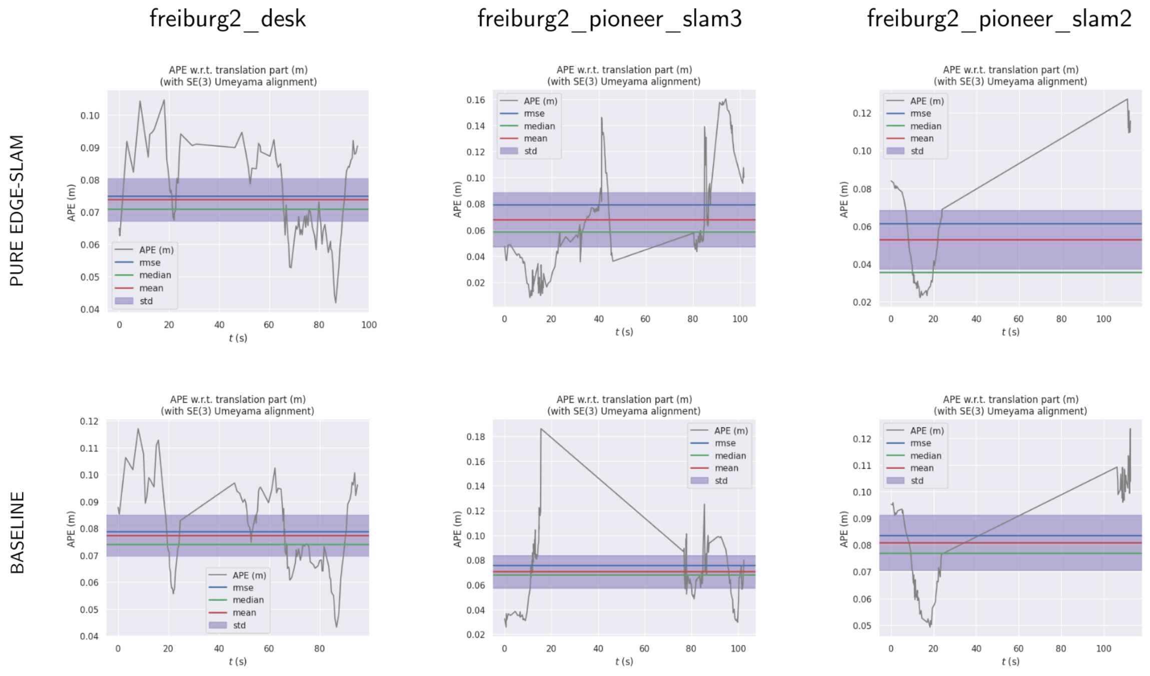

Figure 3. Experimental Results Across Three Datasets

be roughly in the middle of a dataset). The edge would then clear out all the necessary state required to track, including the Keyframe Database, the Bag of Word database and the Global Map. Before resetting, we ensured that the previously created map was stored persistently. After the reset, the edge would start tracking again. At the end of the experiment, the keyframes that were collected after the reset were appended to the keyframes stored before the reset. This generated a complete map across the dataset.

We then use Evo [28] to determine the Absolute Pose Error (APE) between the mapped and ground truth trajectories. We do this twice, once with the normal Edge-SLAM operation and once with our induced reset. This allowed us to gauge the mapping accuracy drop with and without the reset.

We obtained appreciable results from 3 datasets freiburg2-_desk, freiburg2_pioneer_slam3, and freiburg2_pioneer_sl-am2. These datasets were taken from datasets uploaded by the Computer Vision Group at the Technical University of Munich [11]. All these datasets were RGB-D datasets. The results of each experiment are collected in Figure 3.

- **freidburg2_desk:** The freidburg2_desk dataset was the sample dataset given by droneslab (from where Edge-SLAM emerged) on their GitHub; hence this was the first dataset we ran. This is a relatively small dataset where the camera revolves around a desk. The mean APE of the baseline test and normal operation was around 0.07. This showed that the initialisation phase did not induce any loss in mapping accuracy. While this does not seem to make sense at face value, we hypothesize that, given the dataset pertains to a very small environment, the Local Map present on the user device is sufficient to track and localize effectively when the edge does not have any map data. Hence, there is considerable overlap between the untracked and tracked area, allowing for relatively better mapping even without the Global Map on the edge.
- **freigburg2_pioneer_slam3:** To further test our hypothesis, we chose a large dataset that tracks new regions, i.e. it has no small loops. The dataset involves a robot mapping a hallway, meaning that tracking is done linearly, with no small loops. In this case, the mean APE was a little over 0.07 meters with the reset and about 0.065 meters with normal Edge-SLAM operation. This was more along the lines of expected behaviour but nothing particularly significant.
- **freigburg2_pioneer_slam2:** Lastly, we chose to run our tests on the most extensive dataset we could find at the time. A robot also maps a hallway in this dataset, but the map is very large. The mean APE for the normal operating Edge-SLAM was around 0.05 meters and 0.08 meters with the reset. This showed a clear and significant drop in accuracy with reset, which indicates that state

migration to circumvent the reinitialization pause is vital to maintain the mapping accuracy.

We would also like to point out that we ran Edge-SLAM with several other datasets, such as freiburg3_long_office_household and freiburg2_large_no_loop. The results of these experiments were not mentioned above since even in normal Edge-SLAM operation; tracking was being lost in these datasets; hence we could not produce robust experimental results to compare with the results gathered from our baseline reset test. Additionally, we would also like to point out that Edge-SLAM had to relocalize fairly frequently while playing the freiburg2_desk, freiburg2_pioneer_slam3, and freiburg2_pioneer_slam2 datasets. This indicates a potential lack of robustness within Edge-SLAM itself. Furthermore, we have only run tests on TUM RGB-D datasets; further experimentation is needed with different types of datasets from sources such as the KITTI [29] and EUROC [30] sets. Further testing is also required on large-scale datasets that emulate the large environments we believe mobile endpoints in the real world will encounter.

5 FUTURE WORK

As things stand, MultiEdge-SLAM presents a promising extension of the existing Edge-SLAM framework by enabling the integration of multiple edges with a single mobile endpoint. While the current implementation demonstrates the feasibility of this concept, realizing its full potential in real-world scenarios requires addressing several key challenges.

- **Leveraging Multiple Endpoints for Mapping:** In MultiEdge-SLAM thus far, we have maintained the assumption that only one mobile endpoint traverses multiple edges. A by-product of this assumption is that, on each edge, there exists no map prior to the one created with the keyframes supplied by the endpoint. However, in a more real-world setting, there will be multiple endpoints involved that will be sending keyframes to the edge serving them, which is why it is of great significance to leverage that for time-efficient mapping.
  SLAM-Share [8] utilizes shared memory and efficient map merging to build and update a global map from different clients in the context of Augmented Reality, while SwarmMap [9] scales up collaborative visual SLAM service in the context of drones whilst avoiding overheads like data redundancy, bandwidth consumption, and localization errors.
- **Adapting ORB-SLAM3 to the Edge:** The current Edge-SLAM framework is based on ORB-SLAM2, although ORB-SLAM3 has also been developed [10]. One key optimization in ORB-SLAM3 is the Atlas, where ORB-SLAM3 starts a new Global Map once tracking is lost instead of carrying out Global relocalization, ultimately ending up with an Atlas of different maps over time. To maintain a single Global Map, their maps are welded into one upon Loop Closing events.
  Integrating a similar consolidation scheme into MultiEdge-SLAM, where multiple maps hosted on different edges are merged into a unified map, holds the potential for improved performance and enhanced mapping capabilities. However, Edge-SLAM is a complex and highly concurrent framework, and adapting ORB-SLAM3 would likely necessitate significant modifications to the existing Edge-SLAM architecture.
- **Measuring the Desirability to Connect to an Edge:** In a more real-world setting, where there exists an ecosystem of edge servers, it is non-trivial to ascertain how exactly a device will decide which edge to connect to, both at the start of offloading and during handover. Given that the key motivation behind MultiEdge-SLAM is to enable emerging applications like autonomous vehicles, drones, and AR/VR, it is fair to objectify the fact that, at any point in time, the device should ideally be offloading computation to the edge computer with which it can achieve the best possible experience. This must be quantified in terms of some metric that measures the desirability to connect to a particular edge.

6 CONCLUSION

In conclusion, by enabling seamless state migration within the Edge-SLAM architecture, MultiEdge-SLAM aims to enhance the performance and applicability of ORB-SLAM for emerging applications like drones, autonomous vehicles, and AR/VR. Through baseline tests, we have highlighted the performance bottlenecks associated with mobile endpoint switching in Edge-SLAM and demonstrated the need for a more robust solution.

Although we have thoroughly evaluated all three of our design ideas on a conceptual level, much still needs to be done in order to practically realize MultiEdge-SLAM. State migration is a small subset of the problem as it is just as important to address the consolidation of different local map updates into one global map and to ensure that the entire loop closure process (including Global Bundle Adjustment and Pose Graph Optimization) is properly reflected across all edges involved in the loop. Moreover, the Edge-SLAM codebase is highly concurrent and tightly interlaced with ROS; in other words, introducing a new edge will most likely require us to delve into the unchartered territory of ROS.

ACKNOWLEDGEMENTS

We extend our heartfelt gratitude to our esteemed supervisors, Dr. Fawad Ahmad and Dr. Zafar Ayyub Qazi, for their invaluable guidance and unwavering support throughout our research journey. Their expertise, constructive feedback, and provision of abundant resources have been instrumental in shaping our research and enhancing its quality. Additionally, we would also like to extend our thanks to members of Robotics and Control Systems Lab of the Electrical Engineering Department at LUMS for their much-needed assistance in facilitating the successful deployment of Edge-SLAM for our baseline tests. Lastly, we would like to express our sincere appreciation to the members of our research group for their collaborative spirit and support. Their contributions, discussions, and shared insights have significantly enriched our project and made this research endeavor a truly collective effort.

GROUP 9

Muhammad Ahmed (24100027)
Firas Abdullah (24100032)
Adil Rahim Hyder (24100049)
Muhammad Raahim Nadeem (24100072)

REFERENCES

[1] Mahadev Satyanarayanan. 2017. The emergence of edge computing. Computer (Long Beach Calif.) 50, 1 (2017), 30–39. DOI:https://doi.org/10.1109/mc.2017.9

[2] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. 2020. An overview on edge computing research. IEEE Access 8, (2020), 85714–85728. DOI:https://doi.org/10.1109/access.2020.2991734

[3] Meysam Masoudi and Cicek Cavdar. 2021. Device vs edge computing for mobile services: Delay-aware decision making to minimize power consumption. IEEE Trans. Mob. Comput. 20, 12 (2021), 3324–3337. DOI:https://doi.org/10.1109/tmc.2020.2999784

[4] Paweł Nowacki and Marek Woda. 2020. Capabilities of ARCore and ARKit platforms for AR/VR applications. In Advances in Intelligent Systems and Computing. Springer International Publishing, Cham, 358–370.

[5] Bloomberg. Bloomberg.com. Retrieved May 5, 2023 from https://www.bloomberg.com/press-releases/2021-06-22/apostera-mixed-reality-breakthrough-in-the-driving-experience

[6] Charalambos Theodorou, Vladan Velisavljevic, Vladimir Dyo, and Fredi Nonyelu. 2022. Visual SLAM algorithms and their application for AR, mapping, localization and wayfinding. Array (N. Y.) 15, 100222 (2022), 100222. DOI:https://doi.org/10.1016/j.array.2022.100222

[7] 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 Trans. Embedd. Comput. Syst. 22, 1, Article 18 (October 2022), 31 pages. DOI:https://doi.org/10.1145/3561972

[8] Aditya Dhakal, Xukan Ran, Yunshu Wang, Jiasi Chen, and K. K. Ramakrishnan. 2022. SLAM-share: Visual simultaneous localization and mapping for real-time multi-user augmented reality. In Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies, ACM, New York, NY, USA.

[9] 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 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22), 977–993.

[10] Carlos Campos, Richard Elvira, Juan J. Gomez Rodriguez, Jose M. M. Montiel, and Juan D. Tardos. 2021. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. IEEE Trans. Robot. 37, 6 (2021), 1874–1890. DOI:https://doi.org/10.1109/tro.2021.3075644

[11] 2015. Computer Vision Group - Dataset Download. Tum.de. Retrieved May 5, 2023 from https://cvg.cit.tum.de/data/datasets/rgbd-dataset/dowload

[12] melodic - ROS Wiki. Ros.org. Retrieved May 5, 2023 from https://wiki.ros.org/melodic

[13] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. 2015. ORB-SLAM: A versatile and accurate monocular SLAM system. IEEE Trans. Robot. 31, 5 (2015), 1147–1163. DOI:https://doi.org/10.1109/tro.2015.2463671

[14] Raul Mur-Artal and Juan D. Tardos. 2017. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. IEEE Trans. Robot. 33, 5 (2017), 1255–1262. DOI:https://doi.org/10.1109/tro.2017.2705103

[15] Sander Krul, Christos Pantos, Mihai Frangulea, and João Valente. 2021. Visual SLAM for indoor livestock and farming using a small drone with a monocular camera: A feasibility study. Drones 5, 2 (2021), 41. DOI:https://doi.org/10.3390/drones5020041

[16] Jun Cheng, Liyan Zhang, Qihong Chen, Xinrong Hu, and Jingcao Cai. 2022. A review of visual SLAM methods for autonomous driving vehicles. Eng. Appl. Artif. Intell. 114, 104992 (2022), 104992. DOI: https://doi.org/10.1016/j.engappai.2022.104992

[17] 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.

[18] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. 2000. Bundle adjustment — A modern synthesis. In Vision Algorithms: Theory and Practice. Springer Berlin Heidelberg, Berlin, Heidelberg, 298–372.

[19] Raúl Mur Artal. 2015. ORB-SLAM in a challenging indoor sequence. Retrieved May 8, 2023 from https://www.youtube.com/watch?v=_9VcvGybsDA

[20] D. Galvez-López and J. D. Tardos. 2012. Bags of binary words for fast place recognition in image sequences. IEEE Trans. Robot. 28, 5 (2012), 1188–1197. DOI:https://doi.org/10.1109/tro.2012.2197158

[21] edgeslam: Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping. Retrieved May 5, 2023 from https://github.com/droneslab/edgeslam/tree/master

[22] Raul Mur-Artal. ORB_SLAM2: Real-Time SLAM for Monocular, Stereo and RGB-D Cameras, with Loop Detection and Relocalization Capabilities. Retrieved May 8, 2023 from https://github.com/raulmur/ORB_SLAM2

[23] Steven Lovegrove. Pangolin: Pangolin is a lightweight portable rapid development library for managing OpenGL display / interaction and abstracting video input. Retrieved May 8, 2023 from https://github.com/stevenlovegrove/Pangolin

[24] Christoph Hertzberg. Eigen. Tuxfamily.org. Retrieved May 10, 2023 from https://eigen.tuxfamily.org/index.php?title=Main_Page

[25] OpenCV Library. 2019. Releases. OpenCV. Retrieved May 10, 2023 from https://opencv.org/releases/

[26] Sandeep Koranne. 2011. Boost C++ Libraries. In Handbook of Open Source Tools. Springer US, Boston, MA, 127–143.

[27] ROS: Home. Ros.org. Retrieved May 10, 2023 from https://www.ros.org/

[28] Michael Grupp. evo: Python package for the evaluation of odometry and SLAM.

[29] The KITTI Vision Benchmark Suite. Cvlibs.net. Retrieved May 10, 2023 from https://www.cvlibs.net/datasets/kitti/raw_data.php

[30] Kmavvisualinertialdatasets – ASL datasets. Ethz.ch. Retrieved May 10, 2023 from https://shorturl.at/bCGT0