Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping

-Rishika Juloori

Introduction

- Augmented Reality is becoming a part of our lives. It has diverse applications in our daily lives.
- Spatial sensing has been a topic of interest in for a long time, which is useful in place recognition, tracking, and localization.
- Simultaneous Localization and Mapping (SLAM) is a set of algorithms for accurate spatial context.
- It has been of interest in using visual sensing (cameras, LiDARs, depth sensors) for SLAM leading to several of such algorithms.

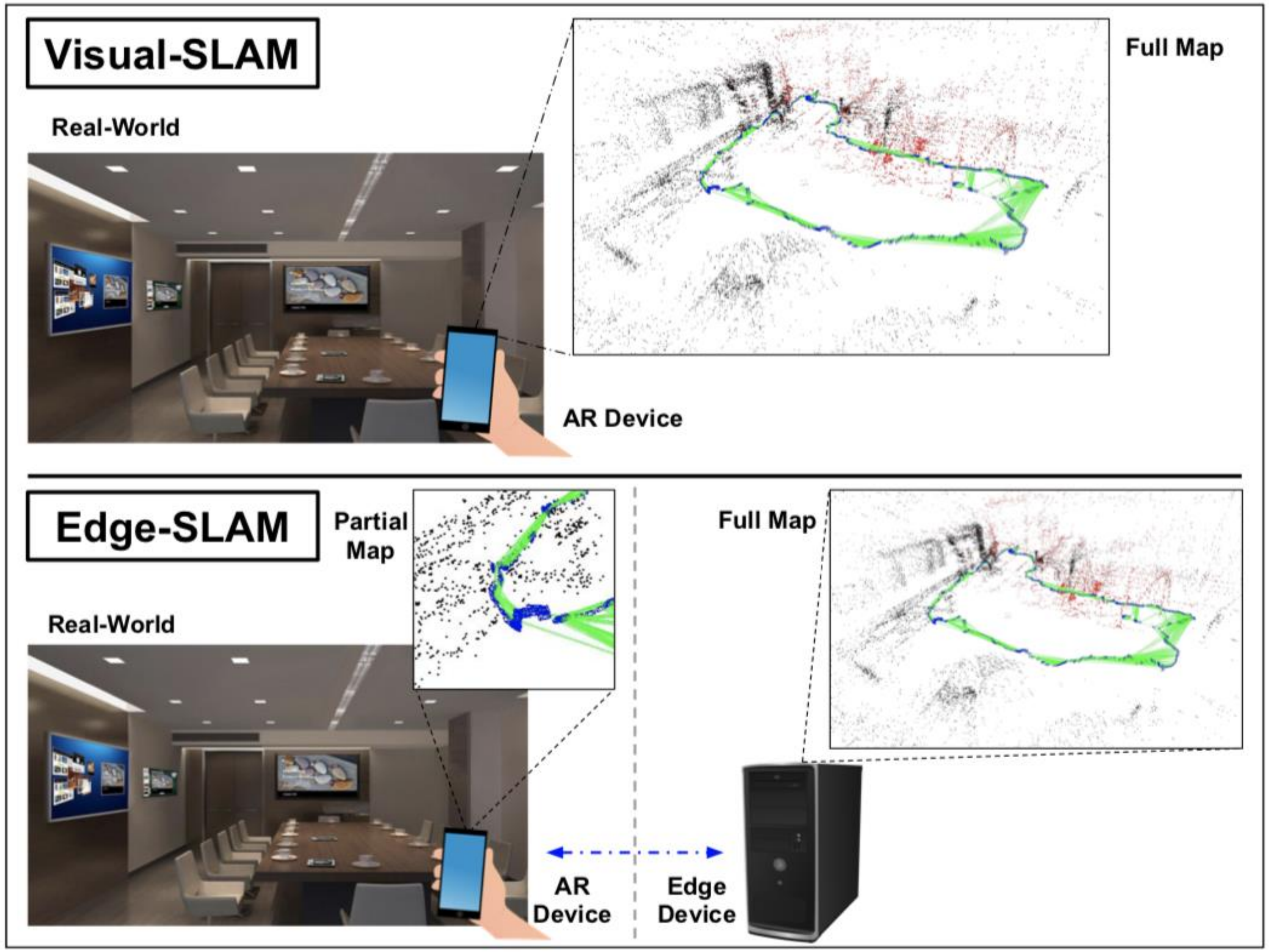


Figure 1: Visual-SLAM vs. Edge-SLAM. An augmented reality device running Visual-SLAM (top), and an augmented reality device running Edge-SLAM in collaboration with an edge device in the environment (bottom) [1–3, 26, 41]

System design (Visual-SLAM System)

- Many SLAM systems use this architecture (PTAM,LSD-SLAM,ORM-SLAM2,ORB-SLAM).
- Input: Series of images captured on camera
  - Accepts regular images (RGB)
  - Many accept stereo images
  - Depth images
  - Depth + colour images

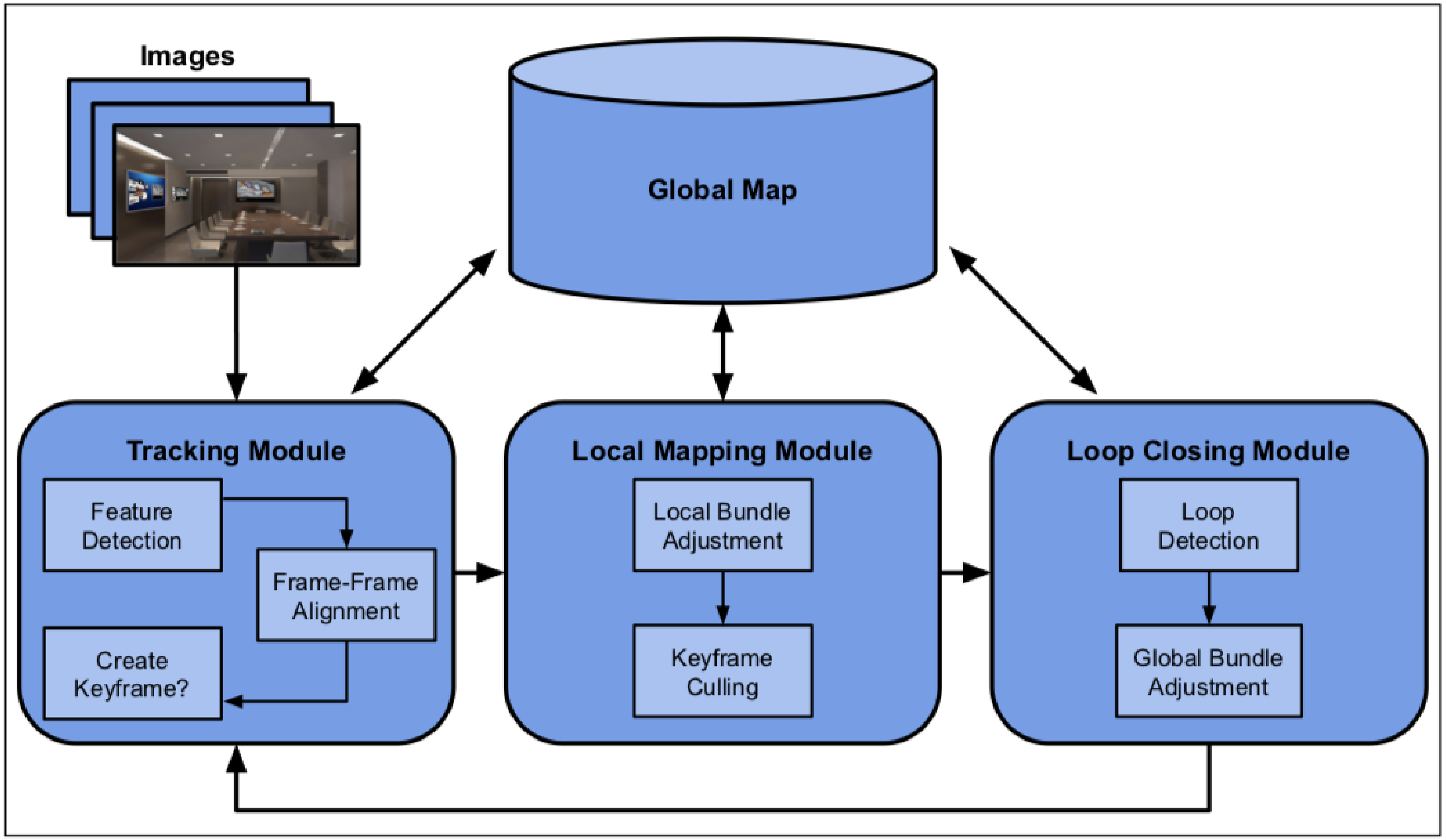


Figure 2: Architecture of a typical Visual-SLAM system [2]

Continued

- Tracking:
  - Detects features in the image: SIFT, SURF, ORB, corners
  - Finds correspondences with previous image using these features
  - Calculates relative change of position over time
- Local Mapping:
  - Creates correspondences between current image and other images in the map.
  - Performs local bundle adjustment
  - It is local as it uses images with common features.

- Loop Closure:
    - When new keyframe added, it is compared to all available keyframes
    - If found a similar keyframe, it performs fusion or graph optimization

Problems with Visual-SLAM

- Computational Complexity:
  - Loop-closure very time expensive
  - Merging map data structure – very complex
  - Refining poses after merge – very complex
- Tight Coupling between Modules:
  - Could use edge/cloud to run some modules but,
  - Algorithm operates on global map – compare, modify, trim
  - May not be able to access shared data leading to improper functioning.

Edge-SLAM Design

Goals

- Reduce computational and memory over head without compromising accuracy
- Keep overall resource usage constant (CPU, memory) for easier workability on mobile phones.

Edge-SLAM architecture

- Goal – offload some computing to nearby edge device.
- The tracking module could work just on local map and hence is on the mobile device.
- The local mapping and loop closing modules need global map and hence were moved to the edge.
- Provide communication mechanisms between the two

Architecture

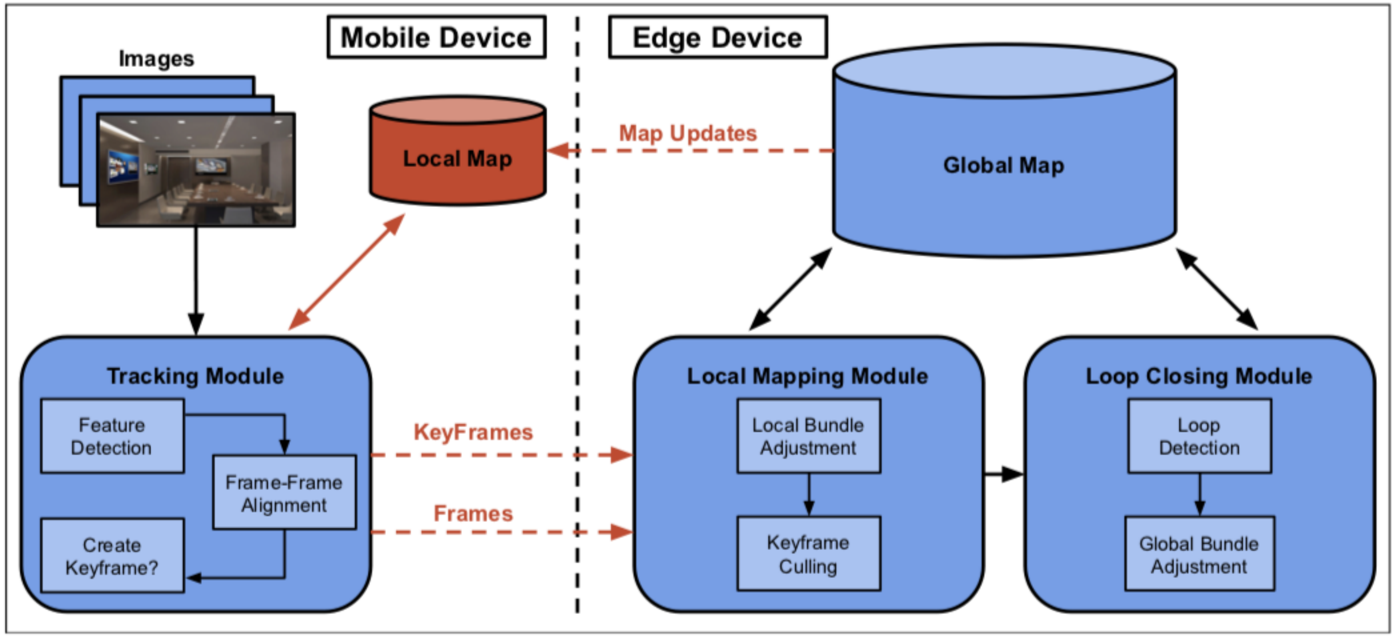


Figure 3: Envisioned architecture of the Edge-SLAM system—our modifications are shown in red [2]

Edge-SLAM in ORB-SLAM2

- ORB-SLAM2: open-source graph-based Visual-SLAM algorithm
- Uses the split architecture described previously
- Global map:
  - Contains keyframes, map-points, Co-Visibility graph, Spanning Tree
  - Co-visibility graph: connects keyframes with similar map-points
  - Spanning tree: Subset of graph
- Local Map
- Map Synchronization
  - Mobile device instantly sends keyframes with map-points to edge
  - Mobile device can choose to update the local map or not from edge

Local map update

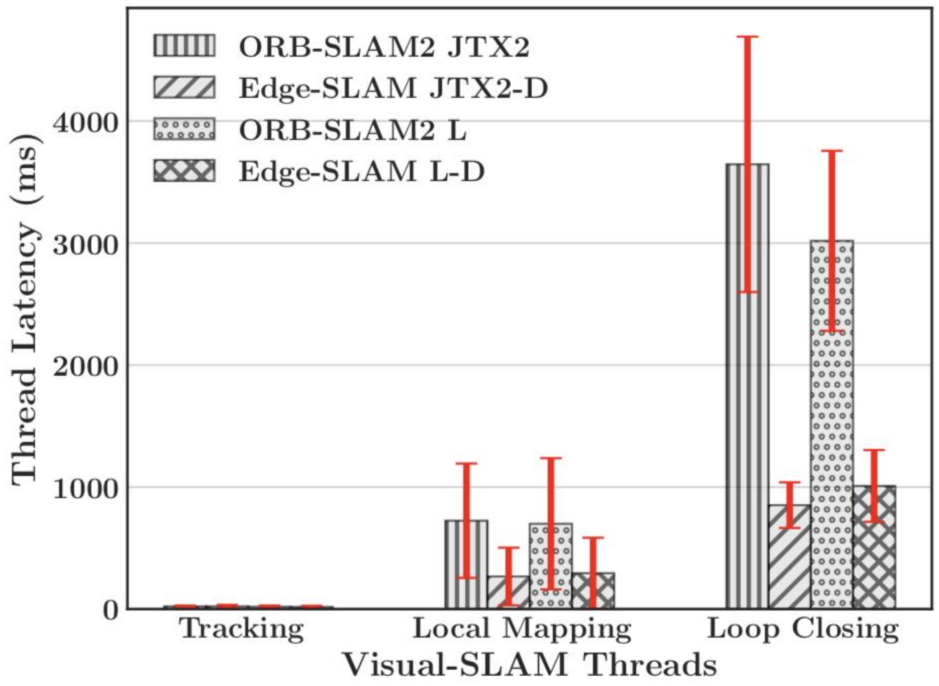
- Two ways:
  - Apply edge changes to mobiles current local map
  - Delete the current map and replace it with the newly received map
- The first one is more efficient than the second.
- It will accumulate lot of unprocessed and unoptimized keyframes and map-points -> inaccuracy, higher chances of drift
- Is expensive to search every single keyframe to find in local map -> higher time complexity, lower performance.
- Complex structure, includes cyclic references -> memory leaks
- Local map shared between various threads -> time consuming

Experimental set-up

- JETSON TX2 – Mobile device
- Dell Latitude laptop – Mobile device
- edge Dell XPS desktop – Edge device

Edge-SLAM Performance

- Accomplishes reduction of overall computational load on the mobile device
  - Edge-SLAM reduces the latency in the local mapping and loop closing modules by offloading them to the edge
  - By offloading the intensive tasks, we reduce the variability of performance on the mobile device
- Accomplishes keeping the resource use on the mobile device constant.
  - The CPU usage for ORB-SLAM2 is at ≈30% while using the JETSON TX2 and ≈40% while using the laptop. In comparison, the CPU usage is ≈15% when using the JETSON TX2 for Edge-SLAM and ≈25% when using the laptop. Overall, there is ≈35-50% reduction in CPU use while using Edge-SLAM.
  - The memory usage of Edge-SLAM is constant regardless of the size of the input



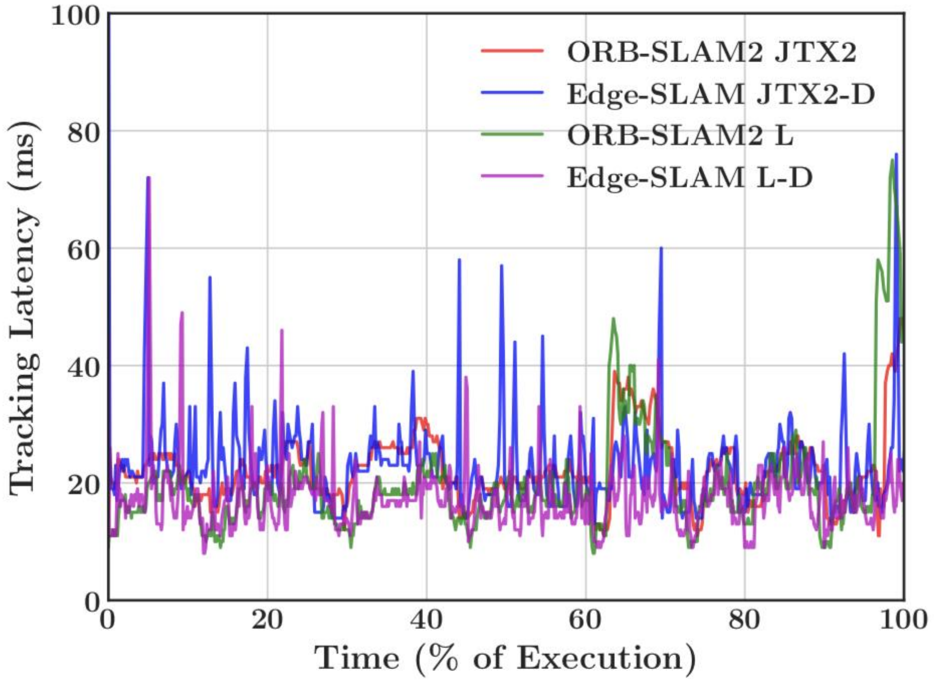


Figure 6: Overall latency of ORB-SLAM2 and Edge-SLAM. The average latency per-module (left) shows that Edge-SLAM offloads the two CPU-intensive tasks. Tracking module latency on the mobile device over time (right) better shows the latency for that module in each configuration

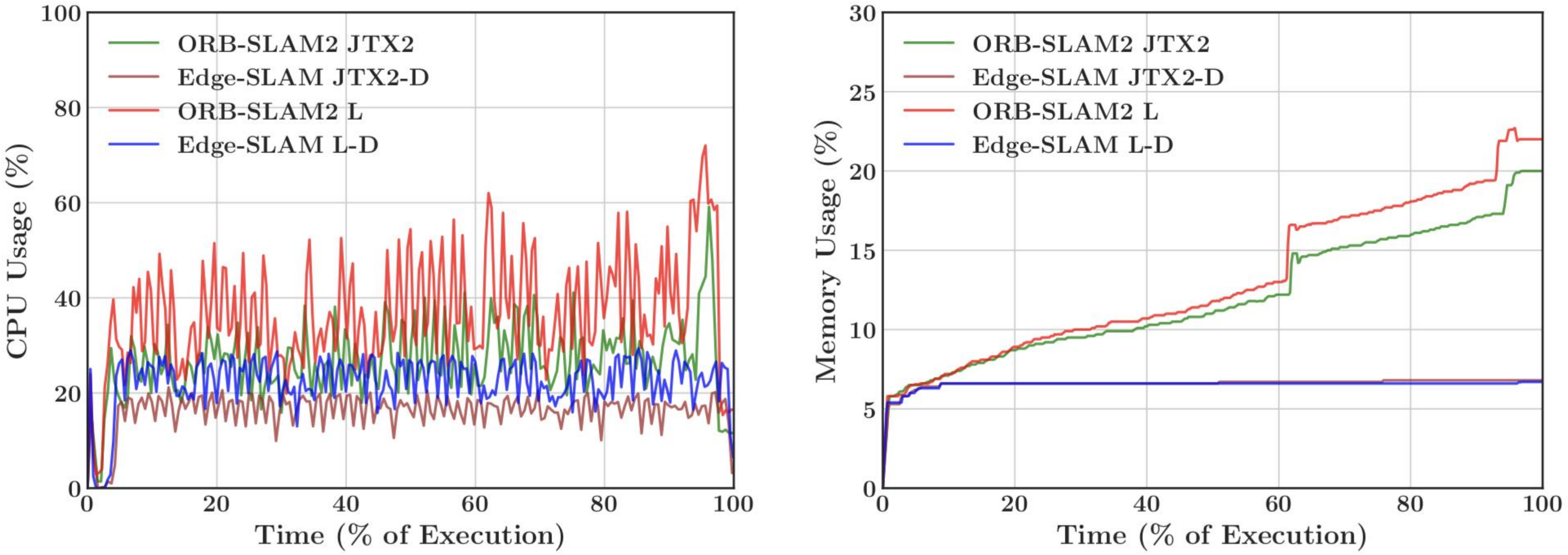


Figure 7: Resource usage of ORB-SLAM2 and Edge-SLAM on the mobile device—CPU (left) and Memory (right). The jumps in memory use at 60% time and 95% time (right) are due to loop closures in ORB-SLAM2

Mapping Accuracy

Visual-SLAM / Accuracy Measure	ORB-SLAM2 JTX2	Edge-SLAM JTX2-D	ORB-SLAM2 L	Edge-SLAM L-D
Mean Localization Error (cm)	20.59 ±10.92	19.23 ±11.32	20.90 ±12.77	21.39 ±9.16

Table 5: Mean Localization Error of ORB-SLAM2 and Edge-SLAM

Network Performance

Edge-SLAM / Map Update	Edge-SLAM JTX2-D (ms)	Edge-SLAM L-D (ms)
Construct Map Update on Edge	57.09 ±0.69	58.30 ±0.66
Re-Construct Map Update on Mobile Device	411.43 ±4.84	285.68 ±3.18

Table 2: Local map update latency on mobile device and edge

Conclusion

- There are lots of Augmented Reality related applications that use spatial localization.
- Though this could be achieved through Visual SLAM systems, Edge-SLAM serves to be a great improvement in terms of efficiency and resources.

References

- Ben Ali, Ali J., Zakieh Sadat Hashemifar, and Karthik Dantu. "Edge-slam: Edge-assisted visual simultaneous localization and mapping." In Proceedings of Mobisys, 2020.