SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Optimizing Visual SLAM on Embedded Devices Using TPU Acceleration

Anas Shahzad

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Optimizing Visual SLAM on Embedded Devices Using TPU Acceleration

Visual SLAM auf eingebetteten Geräten mithilfe der TPU-Beschleunigung

Author:	Anas Shahzad
Supervisor:	Edwin Babaians(Olive Robotics)
Advisor:	Prof. Dr.-Ing. Eckehard Steinbach
Submission Date:	1st August 2024

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 1st August 2024 Anas Shahzad

Acknowledgments

I want to thank my family for helping me navigate through tough situations during my master's thesis. It would be difficult for me to power through the last months of my master's thesis without continuous motivation and advice.

I would also like to thank my supervisor for advising me throughout the master's thesis. He played a pivotal role in referring me to the right code and research literature. I have learnt a lot of technical expertise throughout this whole process together with a lot of grit.

Finally, I would like to thank my family for their unconditional love and support. Your support through this whole process means a lot to me.

Abstract

The thesis explores the usage of edge devices to speed up the visual SLAM pipeline. Visual SLAM is a popular framework that uses visual data from cameras to localize robots in an environment. Visual SLAM uses monocular, stereo or RGB-D data in order to build a map of the surrounding environment. In our thesis, we use RGB-D data to extract features that help us to calculate camera poses in real-time. However, to reduce the computational demand on RGB-D SLAM, we use a edgeTPU to help us perform visual SLAM.

The first part of the thesis, lays the ground work for visual SLAM by exploring the state of the art research in RGB-D SLAM. We also develop the API for the orbbec camera that enables us to use the olive edgeROS device to perform Visual SLAM. Afterwards, we repurpose code from the ORB-SLAM2 pipeline so that we can split the visual SLAM pipeline into frontend and backend.

Orb features are an essential part of the SLAM pipeline, however, they are expensive to compute and rely on the CPU. In the second part of the thesis we use the superpoint network as an alternative to the orb framework to calculate the keypoints and the deep features required for orb matching. We also explore the usage of denoising networks to increase the superpoint detection accuracy for visual slam. Our experiments show that incorporating the superpoint network has some advantages while performing visual SLAM on an edgeTPU device.

Contents

1 Introduction

1.1 Overview

SLAM (also known as simultaneous localization and mapping) is one of the four pillars of an operating robot. The three are action, behaviour and planning. If a robot is given a task to execute, it should know which object it has to manipulate, where the object is present, which obstacles to avoid, and where the robot is concerning the object. The "where am I" part of the problem is known as localization while the "what does the environment look like" problem is known as map building. The SLAM process comprises both of these tasks where the robot learns to construct the environment while finding its own position within the environment.

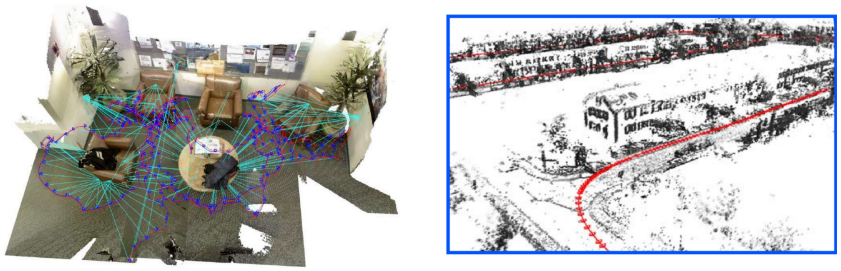
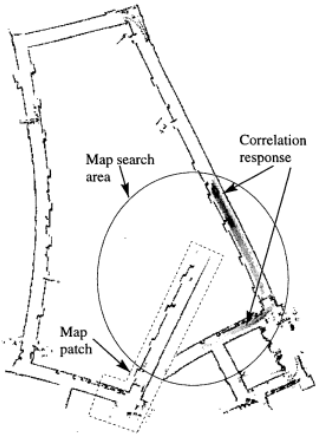


Figure 1.1: Mono-SLAM

Research on SLAM began around 1986 with the work of Durant et al [DB06] in which the authors introduced methods for robot localization and mapping. However, it wasn't until 1986 IEEE Robotics and Automation Conference in San Francisco that the slam problem was officially formalized. During this time a lot of probabilistic methods were being introduced into robotics and the need to come up with a consistent framework arose. The researchers concluded that in order to construct a map of the robotic environment, all relevant methods should use the correlation between landmarks in the environment to construct a map. The same landmarks would have a high degree of correlation when the data is acquired from the sensors, while different landmarks would have an opposite effect. By comparing and merging information from successive sensor measurements, the robot would be able to construct a semi-complete version of the map.

The initial efforts to solve the "SLAM" problem were one in a vacuum whereby researchers used a variety of different sensors and frameworks to implement their own version of the SLAM problem. One of the first works to emerge for SLAM was the J. Leonard et al[LD91] which used ultra-sonic sensors and odometry. While K. Konolige et al [GK99] used lasers to map large cyclic environments.

However, it wasn't until the duo of Sebastian Thrun and Wolfram Bugaard, that the theoretical domain of SLAM was formalized. The early theoretical domains of SLAM were based on Bayesian filtering and EKFs. By using a running aggregate of the landmarks and data points (while factoring the noise of course), it is possible to create a map of the environment even in the presence of outliers. Their concepts were laid out in their hallmark book, "Probabilistic Robotics" [TBF05].



from 2024-05-13 23-52-32.png

Figure 1.2: Ultra-sonic SLAM

During this time, robotic vision was making a lot of progress due to the emergence of cheap commodity cameras. Due to the ubiquitous presence of these sensors and the large amount of data present, work in SLAM shifted towards the use of visual information. Using RGB images and depth data provides improved perception, a wider field of view and richer environment information. Tasks that are similar to the SLAM framework were already making a lot of progress, namely Visual Odometry and Structure from motion. Visual SLAM, visual odometry and Structure from motion(SFM) together make up the domain of multi-view geometry. Visual Odometry is the task of computing the camera poses from a set of images while SFM constructs the 3D representation of a scene given a set of images. SLAM merges these two tasks by simultaneously creating a map and estimating the current pose of the camera. The visual odometry problem was already partially solved around 1999 thanks to NASA research [LMC99] while structure from motion was largely solved by Fitz Gibbon et al

[FZ98].

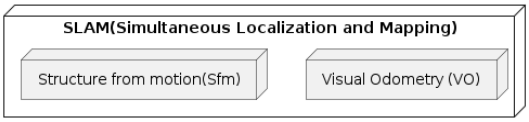


Figure 1.3: Multi-view geometry

Feature	**Visual SLAM**	**Structure from Motion (SfM)**	**Visual Odometry**
Primary Goal	Simultaneously build a map of the environment and localize within it	Reconstruct 3D structure from a series of 2D images	Only estimate the motion of the camera
Map Generation	Yes	Yes(But not to be used for navigation)	No
Loop Closure	Includes loop closure	No	No
Sensor input	Monocular camera, RGB-D camera, IMUs, LIDAR and sonar	Monocular cameras only	Monocular cameras mostly
Application	Robotics, motion planning and navigation	Documentation and VR	Short range path planning

However, both SFM and VO suffer from the fact that the 3D map they construct is not accurate enough for robot planning. Furthermore, all of these methods accumulate drift over time due to the increasing amount of errors in the estimation process. These methods also do not make use of other sensors like IMU and depth sensors to refine the process of trajectory estimation and map building.

Due to these endemic problems within these methods, researchers decided to switch to a different paradigm known as visual slam. Visual SLAM computes the position and orientation of the camera in a trajectory while estimating the 3D map of the environment simultaneously. It uses input from either a monocular camera(Monocular vSLAM), Stereo camera(Stereo vSLAM) or an RGB-D camera(RGB-D vslam). Often,

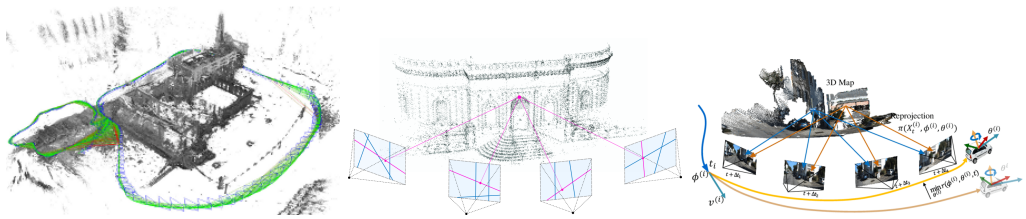


Figure 1.4: (a) Visual slam (b) Sfm (c) Visual Odometry

vSLAM can fuse information from other sensors such as IMU and lidar to refine the estimate of the camera position and orientation.

1.2 Visual Slam

Visual SLAM can be roughly divided into feature-based visual slam and direct method visual slam. The figure 3.17 shows the general pipeline of the slam framework that uses feature matching in order to calculate the position and orientation of the camera. Each frame(sensor data) is fed into a frontend whereby features are calculated and matched to adjacent frames. Feature-based methods are faster than the direct method since they do not rely on the entire information within a frame. Most feature-based algorithms perform sparse reconstruction whereby a lot of information within the environment is lost. The user of such SLAM algorithms can append some post-processing methods such as point-cloud reconstruction that would allow the user to perform visual SLAM.

from 2024-05-20 20-42-14.png

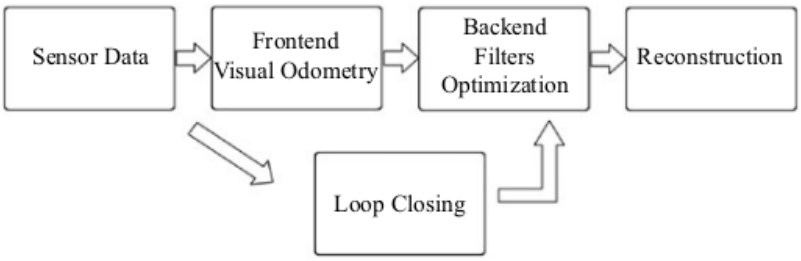


Figure 1.5: Feature-based Visual Slam general pipeline

Direct method-based algorithms are much more flexible in their general framework whereby they incorporate sections from the feature-based methods as well such as loop closure. The direct methods not only optimize for camera position and orientation but also for the dense structure of the environment. Hence, direct methods are not suitable

for embedded applications.

The first attempt to use visual data to measure the camera trajectory in real-time probabilistically was done by Pupilli et al [PC05]. The authors used a simple particle filter to estimate the camera position and orientation. The authors use a SIFT feature detector to populate the map in a semi-dense method. Then matching was done frame by frame using correlations between these SIFT detectors.

from 2024-05-22 23-10-22.png

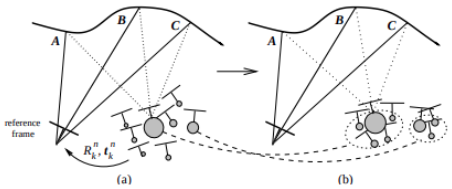


Figure 1.6: Particle filter SLAM

The formalization of the frontend part of visual SLAM was done by Nister et al [NNB04] in which the authors used feature detection frame by frame to calculate the camera motion. It essentially calculates feature correlation and outlier removal to identify similar landmarks within a certain frame in order to calculate the relative camera position and orientation. It was the first paper that employed RANSAC(a fast optimization method that measures inliers and outliers based on similarity to a model) in order to detect feature mismatch and outlier detection.

The hall-mark paper for visual SLAM David et al [Dav+07] is called Mono-SLAM which use Extended Kalman filters in order to estimate the trajectory of the cameras. The extended kalman filter is an extension of the standard kalman filter for non-linear paradigms(e.g estimation of 3D position of the camera). In this framework, we first linearized a problem via Taylor expansion and then applied the kalman filter update equation to the state vector. The state vector in this scenario would be the camera position and orientation. The mathematical formulation of the kalman filter is based on a recursive implementation where the predicted state of the model depends upon the noise as camera motion model. In most cases, we assume the camera motion model to be linear without any fast changes or occlusion. In the prediction step, the robot's pose and the map are predicted based on the motion model.

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

where:

- $\hat{x}_{k|k-1}$ is the predicted state vector (robot's pose and map features).
- f is the nonlinear process model (motion model).
- $\hat{x}_{k-1|k-1}$ is the previous state estimate.
- u_k is the control input (e.g., odometry data).

Mono-SLAM uses a probability density of the environment which it updates and populates via features from an RGB camera. The features are calculated using the shi and tomasi operators. Mono-SLAM suffers from the fact the user needs to provide a known target to initialize the probability map which is often not available. Although the results of mono-slam are not accurate enough to be used in real-time robot navigation, they still pave the way by providing the feature-based extraction and matching framework for future vSLAM algorithms.

from 2024-05-20 21-41-14.png

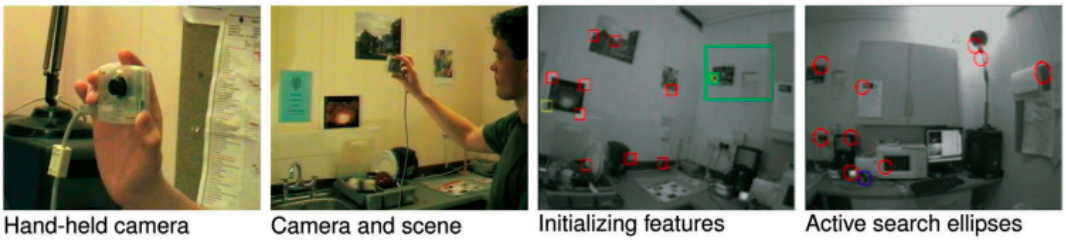
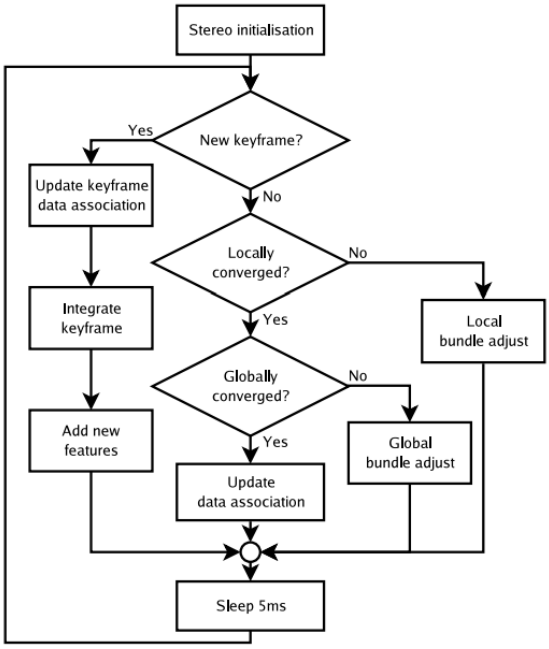


Figure 1.7: Mono-SLAM with feature initialization and matching shown on the two left frames

Until now the SLAM procedure has been very computationally expensive without reliable real-time performance. Salient features which are represented as landmarks are represented in a state vector. As the number of such landmarks grows the size of the covariance matrix increases exponentially. We would need a method which utilizes the sparse nature of this covariance matrix. This is where the work of Klein et al [KM07] is very important such that it separated the task of separating the tracking and mapping thread in visual SLAM. The mapping thread initializes the map and populates the depth information using triangulation. This is also the first method that formally used Bundle adjustment on a local and global level while constructing a map. Bundle adjustment is an optimization process that reduces the re-projection error when going from 3D to 2D. When new keyframes are added, the algorithm performs local bundle adjustment to adjust the camera poses while it performs global bundle adjustment periodically to adjust the camera poses. Furthermore, PTAM stores

the matrix using creative data structures and Cholesky decomposition to reduce the memory and computation footprint. PTAM still is not fully automatic since it requires user input to identify initial keyframes. Furthermore, there is no loop closure involved with PTAM hence, recurring landmark features are not utilized effectively. To date, PTAM has only shown good results for small desktop environments.



from 2024-05-21 23-26-37.png

Figure 1.8: PTAM mapping thread

During this time, some advances were also being made in the direct method. The first proper implementation of the direct method was done by Necombie et al [NLD11]. The authors divided the slam process into dense mapping and tracking threads. The dense mapping thread computed the depth values using average photometric error for multiple frames. During the tracking part, the optimization method aligns the camera position and orientation parameters by projecting the frame from the dense mapping part to a virtual camera.

$$\mathbf{C}_r(\mathbf{u}, d) = \frac{1}{|\mathcal{I}(r)|} \sum_{m \in \mathcal{I}(r)} \|\rho_r\left(\mathbf{I}_m, \mathbf{u}, d\right)\|_1 ,$$

where the photometric error for each overlapping image is:

$$\rho_r\left(\mathbf{I}_m, \mathbf{u}, d\right) = \mathbf{I}_r(\mathbf{u}) - \mathbf{I}_m\left(\pi\left(\mathrm{KT}_{mr}\pi^{-1}(\mathbf{u}, d)\right)\right).$$

The authors also detailed their optimization approach whereby they perform gradient ascent and then descent by after selecting matching points during an iterative search. The method yields excellent results when considering the mapping environment's reconstruction. However, this optimization procedure is computationally costly and requires a CPU+GPU setup to run efficiently.

Another, attempt to perform semi-direct visual odometry(SVO) was done by Forster et al. They followed a similar process of separating the mapping and tracking threads. This method uses direct motion model estimation instead of feature matching and tracking to speed up SLAM. However, they used a probabilistic map to do optimum depth map estimation. A 3D point is included only if a Bayesian depth filter converges. Furthermore, the algorithm keys a fixed number of frames within the map to reduce the demand for exponential complexity. A frame is only included if it matches within 12 percent of the Euclidean distance while the frame that is the furthest from all keyframes is excluded. These restrictions make SVO the first SLAM algorithm to work in real-time on an embedded device such as a drone. However, due to the short range of data association, SVO suffers from a lack of accuracy.

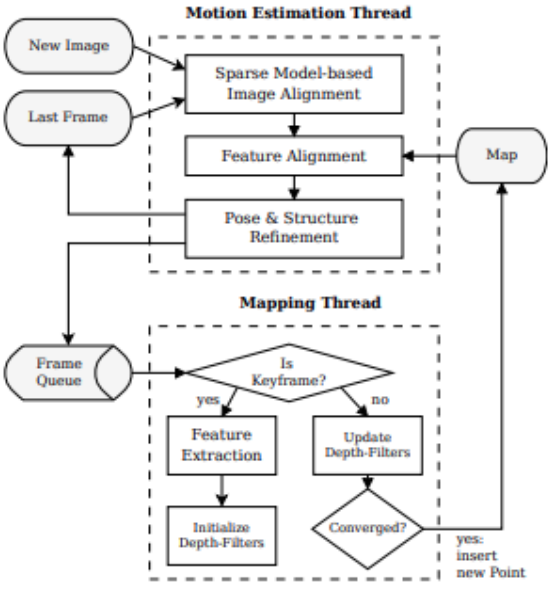


Fig. 1: Tracking and mapping pipeline

from 2024-05-23 21-54-53.png

Figure 1.9: SVO

Around 2017, convolutional neural networks were ubiquitous in computer vision, hence Tateno et al [Tat+17] decided to apply CNN to the task of visual slam. The

authors also introduced a specific normalization procedure to increase robustness when estimating the depth using different cameras. The network also outputs the semantic category of each point inside the map. Apart from the first frame, all other frames are used to perform local optimization.

from 2024-05-23 22-54-00.png

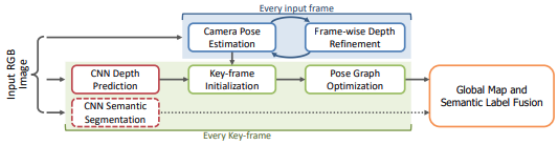


Figure 1.10: CNN-SLAM

Until now, visual SLAM was largely being solved in isolated communities with no general library or framework that ties all of the research together. This problem was overcome by Kummerle et al [Küm+11] by introducing the library g20. This optimisation library allowed any researcher to write any slam algorithm in less than 20 lines of C++ code. The design of the library is supposed to be highly modular with robust kernels that are required for reliable non-linear optimization. G2o also supports variable of the Levenberg-Marquardt Algorithm which can deal with sparse matrices that frequently occur within the SLAM problem.

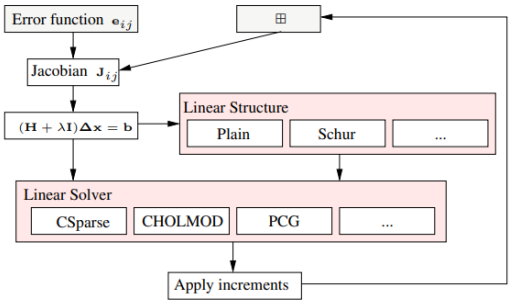


Figure 1.11: g2o library

Modern SLAM systems often employ the use of 3D object recognition, GPU computation or graph-based optimization methods to improve the accuracy of SLAM. One of the hallmarks of modern SLAM is the implementation by Engel et al [ESC15]. LSD-SLAM[ESC15], was the first direct method to run on a CPU that can output a semi-dense map of the environment while providing an accurate estimate of the camera position and orientation. The method estimates each keyframe as a node in a graph and all the edges as similarity transforms of se3. The method also takes into account

varying noise in depth estimate which solves one of the most pertinent problems in in monocular slam. To detect loop closure once a new frame is added, the algorithm selects the closest 10 frames as suitable candidates. The reasoning behind the choice lies in the fact that for loop closures, using a small number of key points yields a better initialization for the loop closure procedure.

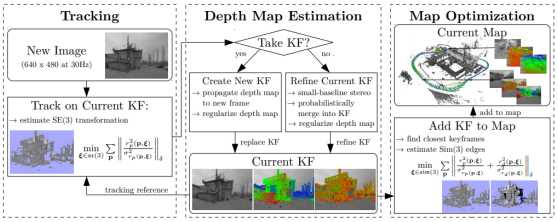


Figure 1.12: LSD slam algorithm

Another method that uses sparse reconstruction with local optimization is the work by Engel et al [EKC18] that models the entire process from camera calibration(camera intrinsic and extrinsic), affine brightness parameters to inverse depth values. After all of the parameters listed previously are calculated, the algorithm then uses all of these parameters to create new keyframes. During this process, the algorithm decides if each point should be included or not and whether the keyframe should be marginalized. The optimization is computationally expensive and done via the following formula: $E_{photo} := \sum_{i \in \mathcal{F}} \sum_{\mathbf{p} \in \mathcal{P}_i} \sum_{j \in obs(\mathbf{p})} E_{\mathbf{p}j}$

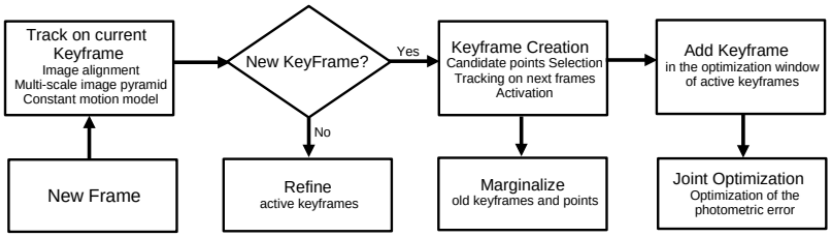


Figure 1.13: DSO slam algorithm

Another method that uses non-linear optimization is the work Leutenegger et al[Leu+13] integrated IMU measurements in a probabilistic fashion. The method is able to do away with the costly RANSAC method and uses brute-force matching for feature detection and matching. Any outliers are removed by a simple chi-square test. The method also distinguishes between keyframes and non-keyframes by seeing if the new data overlaps less than 60 percent of the given area.

Here is a brief overview of all the methods that are discussed in this section:

SLAM Method	Method
SLAM for an Autonomous Mobile Robot	Feature-based
Large-Scale Map-Making	Feature-based
Probabilistic Robotics	Feature-based
Real-Time Camera Tracking using a Particle Filter	Filtering based
Visual Odometry	Feature-based
Mono-SLAM: Real-Time Single Camera SLAM	Feature-based
Parallel Tracking and Mapping for Small AR Workspaces	Feature-based
Dense Visual SLAM for RGB-D Cameras	Direct
SVO	Direct and feature-based
CNN-SLAM	Direct
g2o: A General Framework for Graph Optimization	Direct
LSD-SLAM: Large-Scale Direct Monocular SLAM	Direct
Direct Sparse Odometry	Direct
Keyframe-based Visual-Inertial SLAM using Nonlinear Optimization	Direct,feature-based

Table 1.1: SLAM Methods and Citation Links

1.3 RGB-D visual Slam

The SLAM method described above is mostly general frameworks that work on monocular or stereo datasets. However, due to the rise of commercial RGB-D cameras, it makes more sense to delve deeper into RGB-D SLAM for dense reconstruction. RGDB-D SLAM offers the following advantages over monocular/stereo slam:

- **Triangulation**: Computing depth in monocular/stereo slam involves computing the 3D position of a point present in two different frames. Computing this via brute force is computationally expensive and is very sensitive to small changes in values.
- **Scale ambiguity**: Methods other than RGB-D slam suffer from the fact of not knowing the relative size of the environment. This size must be either provided by the user or computed and adjusted on the fly. In either case, good initialization is required.
- **Search radius**: Due to the presence of the third dimension, searching for surrounding points is easier due to the curse of dimensionality problem. In monocular slam, the algorithm needs to have thresholds/clustering methods to find out the neighbouring 3D points in a frame. This reduces the computational complexity of most of SLAM algorithms.

- **Drift accumulation**: Since triangulation is very sensitive to small changes in illumination and position, errors during tracing quickly accumulate over time. Loop closure is also difficult in this situation since loop closure only relies on 2D features instead of depth of closing the loop.
- **Edge devices**: RGB-D cameras work seamlessly with embedded devices and edge devices in the area of visual slam. They do not suffer from the problem of missed frames since RGB-D slam does not involve the more computationally expensive procedures that are associated with Monocular slam.

Due to the considerations above we decided to work on RGB-D slam during our master's thesis. The following section contains an overview of the most important RGB-D slam method followed by the use of RGB-D slam in an edge device context.

1.3.1 RGBD-SLAM

Like generic SLAM methods, tracing in RGB-D slam can be broadly divided into three categories: direct method, feature-based methods and filter-based methods. However, unlike SLAM methods, mapping is divided into point-based mapping and volumetric mapping. Volumetric mapping uses a truncated signed distance field to represent a 3D environment. All cells with negative values lie inside the object while those with a positive value lie outside the object. Volumetric methods have gained in popularity due to the implosion of deep neural networks. However, these methods usually rely on accurate priors to deal with partially unseen objects and they routinely fail in outdoor environments due to their inherent limitations.

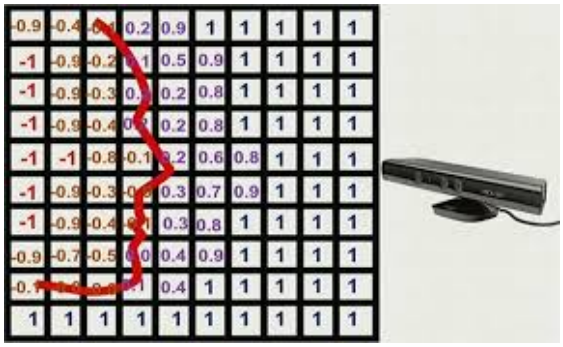


Figure 1.14: truncated signed distance filed

The first coherent attempt for RGB-D slam, was done by Newcombe et al[Iza+11] called KinectFusion. Kinect fusion takes raw depth data from kinect sensor and uses

the data to initialize a normal map and a vertex map. The depth data is filtered through a bilateral filter to reduce the amount of noise. Kinect fusion involves ICP to calculate the transformation matrices of the camera rather than a bundle adjustment or loop closure method. At the end of each cycle, the algorithm produces a signed TSDF of the surface.

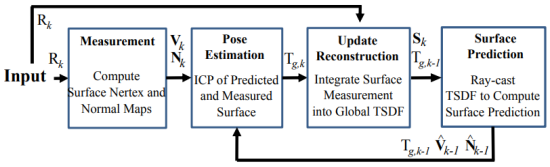


Figure 1.15: KinectFusion

The successor model to KinectFusion is BundleFusion which collects depth data from a Microsoft Kinect[Dai+16] or intell real-sense sensor. The algorithm does intra-frame optimization for camera pose and then does global optimization according to Dense and sparse matching energy functions:

$$E_{dense}(\mathcal{T}) = w_{photo}E_{photo}(\mathcal{T}) + w_{geo}E_{geo}(\mathcal{T})$$

$$E_{sparse}(\mathcal{X}) = \sum_{i=1}^{|\mathbf{S}|}\sum_{j=1}^{|\mathbf{S}|}\sum_{(k,l)\in\mathbf{C}(i,j)} \left\|\mathcal{T}_i\mathbf{p}_{i,k} - \mathcal{T}_j\mathbf{p}_{j,l}\right\|_2^2.$$

BundleFusion uses SIFT features for each camera frame and optimizes PCG(Preconditioned conjugate gradient) as an optimization technique. After each frame is processed and updated using this global optimization technique, the algorithm then updates the 3D scene using a clever cache technique.

An extension of the KinectFusion and Bundle fusion methods is the one proposed by Whelan et al [Whe+16] in which they performed SLAM without global optimization. Their method can be divided into four stages:

- Estimate the surfel-based model of the environment
- While tracking, segment parts of the model are not observed
- Each frame is fit into the model using non-rigid deformation. The part of the model that fits with the frame is reactivated
- A database of views is maintained from which each frame is compared. If a match is detected, then all views are registered and aligned again for global consistency.

Another method that doesn't rely on global optimization is the work by Niessner et al[Nie+13]. SLAM methods that rely on volumetric-based methods usually either suffer from computational overhead due to the hierarchical nature of the data structure or memory inefficiency. Using a simple data structure and a hashing technique, the Voxelhashing algorithm calculates each voxel's TSDF value. After calculating the TSDF, the algorithm then goes over the values to do garbage collection to remove all voxels which have TSDF values that lie outside of a certain threshold. After that, the 6Dof of the current frame to the reference frame is calculated by a point-plane ICP(iterative closest points) method. Point-plane ICP is a method which seeks to minimize the distance between a point in frame A and the closest point in a plane in frame B. The ICP algorithm goes over the minimization over several iterations before converging:

$$E(\mathbf{R}, \mathbf{t}) = \sum_i (\mathbf{n}_i \cdot (\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i))^2 \tag{1.1}$$

where:

- $\mathbf{p}_i$ are the points in the source point cloud.
- $\mathbf{q}_i$ are the corresponding closest points in the target point cloud.
- $\mathbf{n}_i$ are the normal vectors at the points $\mathbf{q}_i$ in the target point cloud.

The first method that used the feature-based SLAM pipeline formally was the work by Felix Endres et al [End+12]. The pipeline follows the same procedure of feature extraction(e.g SIFT, SURF), feature matching and RANSAC. The authors also made use of g20, which is the publicly available library mentioned before:

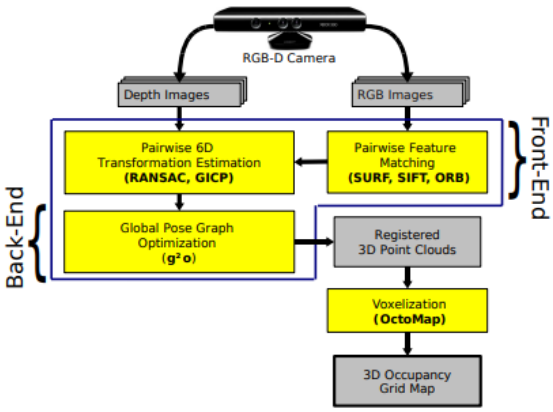


Figure 1.16: RGBD slam

A third approach that emerged after the success of convolutional neural networks was the use of objects rather than features to track and map SLAM. The pioneers of this approach were the authors of SLAM++[Sal+13] who decided to use semantically classified objects as landmarks in SLAM mapping. The algorithm in question represents the objects and camera poses as graph nodes that are continuously optimized in each iteration. Objects are recognized in the environment by Hough voting, where samples are collected based on the normal estimates of the object, which allows efficient binary search of objects. The method unfortunately suffers from its high computational load due to the expensive dense ICP protocol it uses to find the pose estimate. Furthermore, the method can only detect objects already present within a manually constructed library. However, the algorithm offered a lot of insights about camera tracking, namely the use of relocalization once tracking is lost. Once tracking is lost, a new local graph is created which is matched to the original graph once 3 objects reappear which were present in the original graph.

from 2024-05-30 22-46-37.png

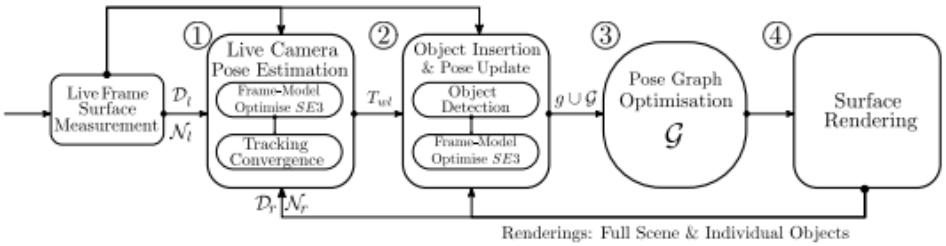


Figure 1.17: SLAM++ pipeline

A more recent RGB-D slam improvement was done by the authors of Plane-edge SLAM [Sun+21]. In feature-based slam methods, the algorithms usually use a fix of point features (e.g SIFT and SURF). However, these point-based features are usually not very robust against laminar surfaces and indoor environments. Hence, to improve the accuracy of feature matching, the authors decided to use planar surfaces with point features as features in the SLAM pipeline as well. The algorithm then uses an ICP-like plane-edge alignment procedure. This addition is very important for RGB-D data since depth data is very noisy.

$$F(\xi) = \sum_{i=1}^{N_\pi} e_{\pi i}^T \Omega_{\pi i} e_{\pi i} + W_p \sum_{k=1}^{N_p} w_{pk} e_{pk}^T \Omega_{pk} e_{pk}$$

ORB-SLAM 2

One of the hallmark papers on which our master thesis is based is the work by Mur-Artal et al [MT16] called ORB-SLAM 2. Orb-slam 2 is the successor to ORB-SLAM which is a generic framework that performs feature-based tracking and mapping using ORB-features rather than SIFT and SURF features. The ORB-SLAM 2 library is characterized by several salient features that result in it being used in several research projects regarding SLAM:

- **Modular design:** Every thread/sub-process in ORB-SLAM 2 is coded separately in a C++ library for use.
- **Key frame management:** Map-points culling strategy ensures the system is efficient.
- **Open Source:** All code and results are freely available.
- **Multiple camera types:** Intel and Kinect cameras can be integrated with the code without much overhead.

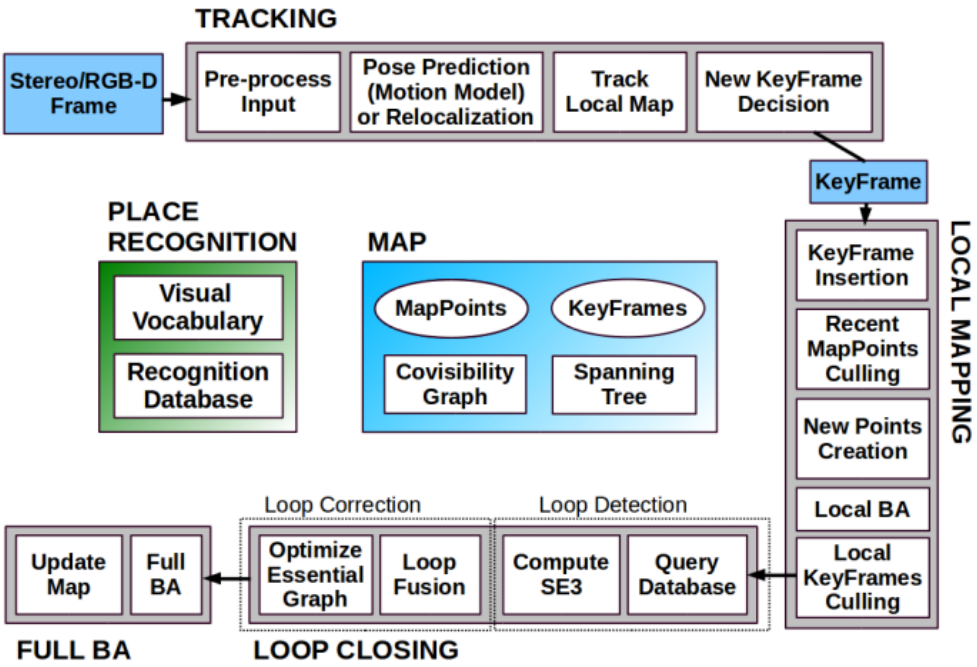


Figure 1.18: ORB-SLAM 2

Orb-features:
As the name suggest, the framework uses orb features instead of SIFT/SURF features in order to perform tracking and sparse mapping. ORB features offer a lot of advantages over the standard SIFT and SURF features. Due to the inherent nature of ORB features, the key points detected do not suffer from rotation or scale invariance. Furthermore, orb features are much faster to compute and require fewer memory-intensive resources. Moreover, since we can only have a finite amount of orb descriptors, the matching process with orb features is much faster and more efficient.

Orb features are calculated via a four-step process:

1. **Keypoint detection:** keypoints are extracted from a image using FAST keypoint extraction. In this process, we use a filter to find if the candidate pixel has a sufficient number of neighbours that are darker or lighter than itself.

2. **Harris score calculation:** A score is assigned to each keypoint so that we can select the best N candidates

3. **Orientation assignment:** An orientation is assigned to each keypoint using the following formulas: $\mathrm{m}_{10} = \sum_{x,y} x \cdot I(x,y)$
$m_{01} = \sum_{x,y} y \cdot I(x,y)$
$M_{00} = \sum_{x,y} I(x,y)$
$C_x = \frac{m_{10}}{M_{00}}$
$C_y = \frac{m_{01}}{M_{00}}$
$\theta = atan2(m_{01}, m_{10})$

4. **Brief descriptor:** A 31X31 patch is chosen around each keypoint. From a predefined sequence of the pixel, comparisons are made, which are then fed into the descriptor vector for each pixel.

1.3.2 Edge RGBD-SLAM

In our thesis, we plan to repurpose ORB-SLAM 2 for use in an embedded setting, whereby the EdgeROS item from Olive Robotics can run a SLAM algorithm in real-time. Running SLAM via an embedded device has a lot of advantages over running SLAM in the main server.

- **Real-time processing:** By mapping the environment in real-time, robots such as drones can easily navigate a fast-paced dynamic environment without waiting for the main server.

- **Security:** By distributing environment mapping among autonomous agents, edge SLAM can provide *security through obscurity*. This would be beneficial for warehouse environments.
- **Cost efficiency and low latency:** By reducing the computing power required in SLAM, cheap embedded devices could be mass-produced for swarms of robots which could then communicate with each other without the need to transfer large amounts of 3D information.
- **Scalibility and resilience:** Since there is no need to communicate with the main server at all times, loss of communication with the main server would not result in cessation of operations.

Running SLAM on edge or embedded devices is not a new phenomenon. Since the inception of SLAM, there have been a large number of pipelines introduced that have wanted to transfer SLAM computation to edge devices exclusively. The approaches can be divided into **edge-only SLAM** and **edge-assisted SLAM**. Among the first works to introduce a distributed approach to SLAM, comes from the authors of C2TAM[RCM14]. In their framework, they plan to do computationally intensive bundle-adjustment and mapp fusion are done in a main server while camera tracking is done on the edge device. The intuition behind this split is the fact that bundle adjustment can take several frames at the same time and hence is ideal for latency. Mapp fusion follows a similar line of reasoning provided that all clients should register their maps individually to the main server as well.

The researchers achieved some promising results when it comes to reducing the computational load on a single-edge device. However, the edge device that they use in this setup is a laptop (Intel Core i7 M 620 at 2.67 GHz, 4 GB RAM) instead of a bare-bones embedded device. Furthermore, coveying and registering changes in the map, requires a lot of communication with the main server, which can introduce more latency within the system. The method also sends individual frames to the main server from time to time, which actually erases all of the security benefits.

EM-SLAM[Wu+18] also follows a similar approach which runs the entire slam pipeline on a laptop. The SLAM framework is reduced to three stages. In the first stage, the algorithm performs initial pose estimation using matching ORB-features. In the second stage, the algorithm refines the prediction and in the third stage, they use a greedy algorithm to match key points between non-adjacent frames. The greedy algorithm introduces a lot of shortcomings in the algorithm, namely if the sequence of frames is too long, then the processing time required to match the orb features increases exponentially. Furthermore, the algorithm only works with monocular SLAM with no open-source support for RGB-D datasets.

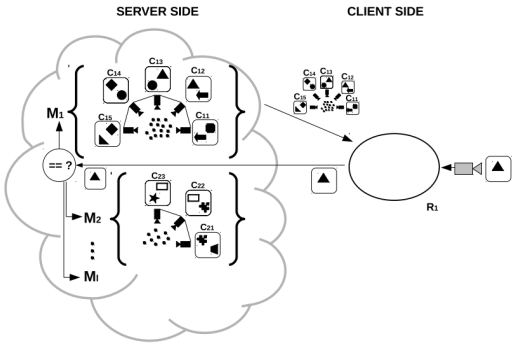


Figure 1.19: C2TAM architecture

Another implementation that focuses on AR/VR devices is the work by Ben Ali et al[Ben+22]. Like C2TAM, the slam framework requires excessive communication between different threads. Furthermore, the framework is implemented with a lot of mutex locks which might lead to race conditions in edge cases if not implemented correctly. The authors also failed to justify the new framework for use in embedded devices exclusively since they computed their results on aa NVIDIA JETSON TX2—64-bit NVIDIA Denver and ARM Cortex-A57 CPUs, NVIDIA Pascal GPU with 256 CUDA cores, 8 GB Memory(same architecture as Microsoft Hololense), intel i-5 laptop and intel i-7 with 1080 Ti Nvidia GPU. The devices mentioned are powerful enough to run the orb-slam framework without any distributed framework of setting in itself.

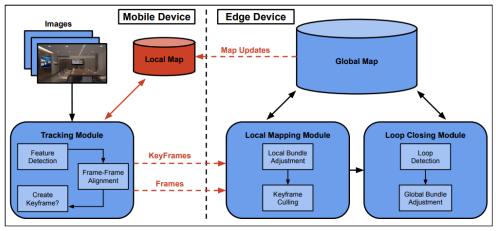


Figure 1.20: C2TAM architecture

Another honorable mention for the use of SLAM in embedded devices is Adapt SLAM [CIG23] which offloaded loop closure and bundle adjustment to the main server. Meanwhile, AdaptSLAM performs visual slam using load balancing of various computationally intensive tasks among candidate main servers. In a nutshell, most edge-slam techniques utilize some form of computational off-loading to achieve their ends to perform collaborative SLAM. However, to our knowledge, utilizing cheap devices on the TPU is unprecedented in visual SLAM.

1.3.3 SLAM infrastructure

Cameras

1. **ASUS Xtion Pro Live** is an RGB-D camera that captures point clouds in real-time using the structured motion from the light phenomenon. The structure motion of light phenomenon projects an infrared pattern like a dot matrix on a surface using an infrared sensor. The depth camera receives the infrared pattern and calculates the distance to the surface by calculating the deformations. The drawback of the sensor is that it has a limited range(around 3.5m in the case of ASUS XTION Pro). Furthermore, this method of obtaining the point cloud is susceptible to occlusion, sensitivity to calibration and ambient lighting(reflective surfaces etc).

2. **Microsoft kinect** is a discontinued RGB-D camera that calculates the depth image using the time of flight principle. It calculates the object's depth by measuring how long it takes for a pulse of light to travel to the object and back. This principle is also employed in the Intel Realsense sensors. Microsoft Kinect has been very instrumental in collecting large datasets for visual SLAM. The TUM dataset and Kitti dataset use the sensor to collect their massive corpus of data.

3. **ZED Stereo Camera** The ZED stereo camera uses stereo vision to create a depth map. In stereo vision, the device uses images from both cameras to produce a depth of disparity map. The depth from the disparity map is then scaled to produce the depth map. However, using the depth from the disparity map can cause issues in environments with uneven texture. Furthermore, stereo vision is very sensitive to small changes in calibration.

4. **Orbbec Depth Camera** Orbbec Astra Mini cameras use the structured motion from light phenomenon to calculate the depth map of an image. The Astra mini cameras are ideal for use in edge-devices due to their small size and low voltage requirements. Hence, we would be using the Astra Mini S camera in our visual slam pipeline.

Edge-devices

1. **Intel Core-i5 laptop** Most edge SLAM papers use an intel i5 processor as an "edge" device to perform visual slam. Having a laptop in the SLAM pipeline has a lot of advantages for debugging/visualizing visual SLAM. However, the setup is not practical to scale up due the size of the machine and the high costs involved.

2. **Microsoft hollolense** is an all-in-one device that is built for AR/VR applications for the end user. It has Qualcomm Snapdragon 850 Compute Platform and a custom-built HPU 2.0, hence it packs a lot of computation power to conduct visual SLAM. The device uses sensor fusion from IMU and depth camera images to perform visual SLAM. Nevertheless, the hololense, due to its price and usage, is not a suitable candidate for visual SLAM.

3. **Nvidia Jetson Nano** is a compact custom-made device that is made for use in AI and machine learning applications. It has a Quad-core ARM Cortex-A57 CPU and a 128-core NVIDIA Maxwell architecture GPU. However, the device is still cannot be classified as an "edge" device due to its large size and price range.

4. **Olive Edge-ROS + Coral TPU** The Olive Edge-ROS and Coral TPU is the smallest edge-device on which visual SLAM is applied. The device is a 5cmx5cm in size with the TPU chip installed underneath the PCB. The device also has a USB 2.0 connection(for Orbbec cameras) and a ROS 2.0 linux operating system.

Communication protocol

1. **Cloud-computing** Uses services provided by AWS(Amazon web services), offloading computationally expensive tasks to the cloud is an approach that has been explored extensively in the visual SLAM framework. However, in real-time visual SLAM, the network has to be low latency and high bandwidth if images are sent over the communication protocol. Furthermore, these cloud computing platform requires batch processing, which is not ideal for SLAM due to the low margin of error when it comes to skipping frames.

2. **ROS2** Recently a lot of open-source contributions have surfaced that use ROS 2.0 [] to conduct visual SLAM. ROS 2.0 provides a shared interface to gather and distribute pose information without duplication of messages or high latency. Furthermore, the modular nature of SLAM allows the user to modify the pipeline to offload computationally expensive low-security tasks to the main server.

Camera	Method	Field of View	fps RGB	fps Depth	Price	Range
ASUS XTION PRO LIVE	Sof	**58deg x 45deg**	**60**	**30**	**300 USD**	**3.5m**
MICROSOFT KINECT	Tof	**57.5deg x 43.5deg**	**30**	**30**	**NA**	**4.5m**
Intel Real sense	Tof	**87deg 58deg**	**90**	**90**	**314 USD**	**3m**
ZED STEREO	Stereo	**110deg x 70deg**	**100**	**100**	**449 USD**	**10m**
Orbbec Astra Mini	Sof	**58.4deg x 45.5deg**	**30**	**30**	**159 USD**	**3.5m**

Table 1.2: RGBD Cameras specification

2 Contribution

2.1 Setup

In order to perform visual SLAM we use the following setup:

1. EdgeROS + tpu : Processes the RGB-D images from the depth camera and calculates the ORB features. The features are then transferred to the main server for further computation. After processing is done at the main server, a pose for each frame is received at the end of the SLAM pipeline which is then used to construct an entire pointcloud of the environment.

2. GIGABYTE core i5 laptop with Nvidia GeForce RTX 3050: Acts as a main server. Receives frames of ORB-features and does RANSAC optimization, Bundle adjustment and Loop closure on the features. Returns back the camera poses.

3. Orbec Mini Astra Pro S depth camera: Returns the RGB and depth map images to the EdgeROS for computation. To increase the frame thoughput, a resolution of 640 x 480.

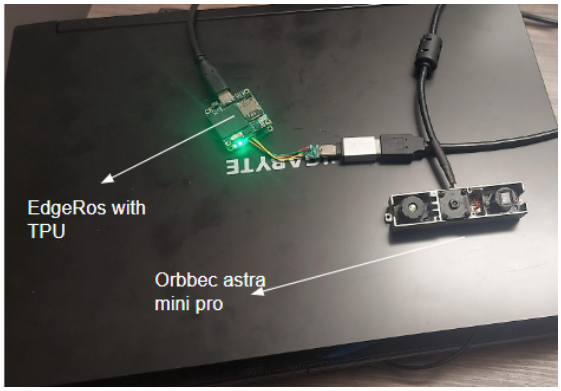


Figure 2.1: Our visual SLAM setup

2.2 Datasets

To evaluate our results we will be using the TUM RGB-D slam dataset [Stu+12]. Future work for evaluating the SLAM pipeline can also use the Kitti RGBD dataset [Gei+13] or the NYU dataset[NF12]. The TUM RGB-D dataset has around 19 sequences ranging from a variety of indoor environments with the kinect camera being held via a robot and a human hand. The authors estimate the ground truth trajectory via a Motion capture from MotionAnalysis to calculate the ground truth trajectory.

Sequence Name	Duration s>s	Avg. Trans. Vel. [m/s]	Avg. Rot. Vel. [deg/s]
Testing and Debugging			
fr1/xyz	30	0.24	8.92
fr1/rpy	28	0.06	50.15
fr2/xyz	123	0.06	1.72
fr2/rpy	110	0.01	5.77
Handheld SLAM			
fr1/360	29	0.21	41.60
fr1/floor	50	0.26	15.07
fr1/desk	23	0.41	23.33
fr1/desk2	25	0.43	29.31
fr1/room	49	0.33	29.88
fr2/360$_h emisphere$	91	0.16	20.57
fr2/360$_k idnap$	48	0.30	13.43
fr2/desk	99	0.19	6.34
fr2/desk$_w ith_p erson$	142	0.12	5.34
fr2/large$_n o_l oop$	112	0.24	15.09
fr2/large$_w ith_l oop$	173	0.23	17.21
Robot SLAM			
fr2/pioneer$_3$60	73	0.23	12.05
fr2/pioneer$_s lam$	156	0.26	13.38
fr2/pioneer$_s lam2$	116	0.19	12.21
fr2/pioneer$_s lam3$	112	0.16	12.34

Furthermore, we will be using four metrics to describe the performance of our model:

1. Absolute trajectory error (ATE): Measures the error between two trajectories. $ATE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|\mathbf{p}_i^{est} - \mathbf{p}_i^{gt}\|^2}$
2. Relative pose error(RPE): Measures the error between successive poses. $RPE = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n-1} \|(\mathbf{p}_{i+1}^{est} - \mathbf{p}_i^{est}) - (\mathbf{p}_{i+1}^{gt} - \mathbf{p}_i^{gt})\|^2}$

3. Root mean squared error(RMSE): $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(e_i)^2}$
4. Time taken(sec)

2.3 Architecture

As described earlier, we repurpose the code from the ORB-SLAM 2 authors for use in our visual SLAM pipeline. For convenience, we split the ORB-SLAM2 pipeline into front-end and back-end. The front-end runs on the edgeROS + TPU framework while the back-end runs on the main server (GIGABYTE laptop). All messages are sent/received via ROS2 communication. The following is a diagram explaining the pipeline:

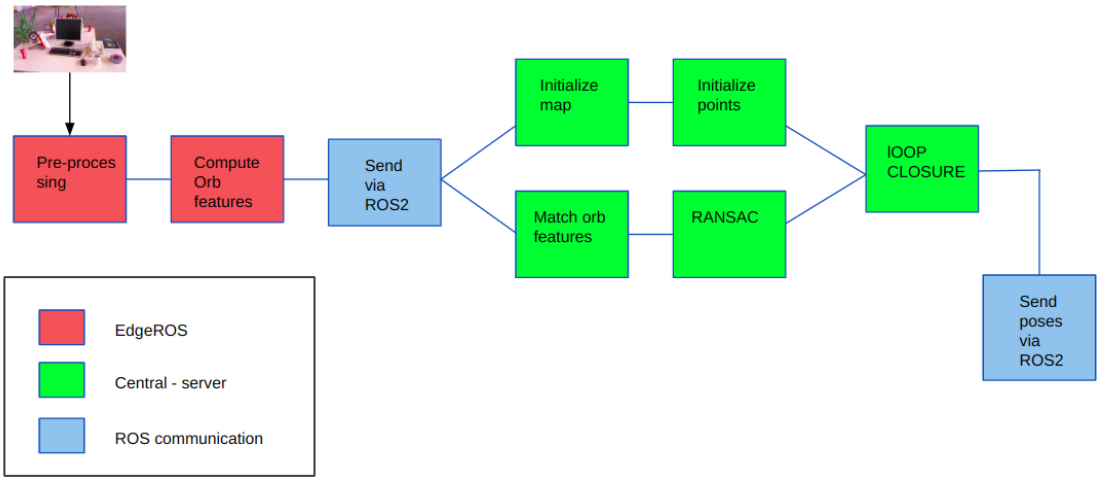


Figure 2.2: Our visual SLAM setup

1. **Front-end**:
   - Receive and process RGB-D dataset: Raw data in unsigned integer format is received by our proprietary driver. Each image is scaled to a 640 x 480 resolution and passed through a threshold filter to remove any obvious outliers. Our algorithm inversely scales the depth images and processes them.
   - Pre-processing: After receiving the RGB-D images, we pass each image through a denoising network(more details to be followed later). Afterwards, each depth image is passed through a k-d tree.

- Feature extraction: After pre-processing, for each RGB image, we calculate either the I)superpoints(which is done via the superpoint(to be explained later) network on the TPU) or the II)ORB-keypoints(which is done on the CPU). For each of these keypoints we compute the orb features and the depth data. Afterwards, the points are then passed to the main-server by publishing a ros-topic. The intuition behind broadcasting the message is in the event of scaling up this system any main-server can have access to the feature data from any robot to start pre-processing. Hence, this will allow rapid computation of camera poses while ensuring security.

2. **Back-end**
   - After receiving the first frame of ORB-features, the backend initializes the map in the background.
   - Matching: It matches keypoints in each frame with keypoints in the next frame. The number of keypoints is matched with a standard ORB matcher, which is evaluated using the Hamming distance between each feature descriptor. The number of viable keypoints should exceed 480 to ensure stable matching from frame to frame. Viable keypoints are calculated using a depth threshold, any points too close to the camera or too far away are not considered in the matching procedure.
   - RANSAC and bundle adjustment: After matching keypoints to each other, the algorithm then uses RANSAC to initialize the computation of the camera pose. RANSAC provides a good initialization to the bundle adjustment framework which refines the camera pose estimation procedure. The Bundle adjustment framework is implemented with the g2o library.
   - Loop closure: The main server also has a parallel thread running for loop closure which analyzes all incoming frames to see if the number of keypoints in the frame is already present in the local map. If the number of similar keypoints exceed 60 percent, a loop is detected and then the backend performs loop closure to refine all of the cameras poses.
   - Transmitting camera poses back: After the frontend signals that the visual SLAM pipeline has ended, it broadcasts the results back to the frontend via ROS.

Post-processing at the backend: After receiving the camera poses from the backend, the frontend creates a point-cloud map of the environment by first stitching the point-cloud together by transforming them according to the camera poses. The point cloud is then passed through a point-cloud reconstruction network(to be explained later) to

refine the point-cloud. The point-cloud can then be visualized using a 3D modeling tool like meshlab.

2.4 Camera calibration

In order to use the Orbbec camera for the visual slam pipeline, we first have to perform the intrinsic calibration of the Orbbec Astra Mini Pro S camera. The intrinsic parameters of a camera refer to the following constants:

1. Focal length
2. Principle points
3. Tangential distortion
4. Radial distortion

To start the calibration procedure, we fix the camera on a mount and use a calibration pattern to procure around 100 images for our calibration by moving the pattern at a distance of 1m in x,y and z directions. The calibration pattern that we use is called the aruco calibration pattern. A charuco pattern is a combination of a aruco pattern and a chessboard pattern. Using the charuco pattern has shown superior performance[Ray+23] than using a standalone aruco or chessboard. This is because it combines the best of both worlds by providing sub-pixel accuracy when detecting the pattern while still maintaining partial pattern detection due to the presence of black markers in the chessboard. Each charuco pattern has a different ID, hence we do not need to flip the images to align each pattern in adjacent frames.

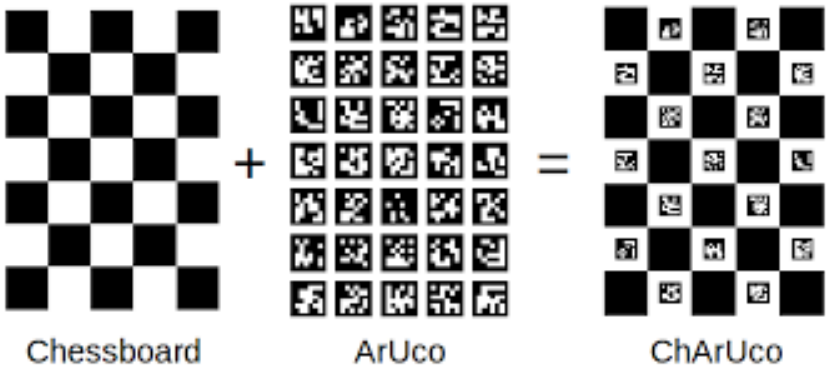


Figure 2.3: Charuco pattern

The images are selected according to the following criterion:

- Each images should provide at least 40 percent new locations of the charuco pattern in the detection window.
- Images that are distorted due to motion blur are discarded
- Images that are the charuco pattern covering less than 70 percent and more than 40 percent are retained.
- At the end of the calibration we tilt the frame in the x and y direction to provide more data for distortion parameters.
- A few images are captured at a distance of 0.8m and 1.4m away from the camera.

The images and the detected charuco pattern are then passed to the opencv2 library to optimize the parameters using the Levenberg-Marquardt Algorithm. The algorithm uses the depth data and the position of the charuco patterns to calculate the reprojection error. The parameters are then validated by 5 other images reserved as test data.

2.5 Superpoint Network

As described before, our algorithm computes two types of keypoints I)superpoint keypoints[DMR18] II)Orb keypoints. The orb keypoints are calculated on the edge-ros cpu while the superpoint keypoints are computed using a coral TPU chip present on the edge-ros. The decision to switch from ORB keypoints to superpoint keypoints is motivated by the following factors:

1. **Viewpoint changes:** Orb is very sensitive to changes in scale and illumination. Zooming in and out of an image slightly will give us totally different keypoints. This is not ideal for SLAM, since we require keypoints from similar landmarks to be present in several frames to perform loop detection. Furthermore, small changes in viewpoint can cause a reflective effect in indoor environments which might result in the loss of keypoints even though the underlying landmark has'nt changed.
2. **Outlier detection:** Orb keypoints often provide keypoints that are not used in the ORB matching and RANSAC pipeline at all. Furthormore, ORB often provides keypoints that are clustered around the same area because it cannot differentiate between similar keypoints in the region beyond a cursory non-maximal suppression step.

3. **Limited descriptive power:** Orb does not make use of the descriptive power of large vision models to extracts areas of special interest in an image automatically. An area that might have less variation in colour or texture may serve as a very useful landmark while performing SLAM.

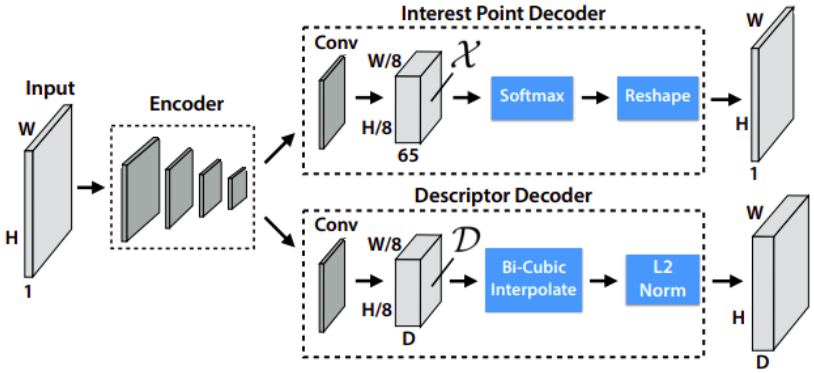


Figure 2.4: Superpoint architecture

The superpoint network alleviates a lot of these problems by being trained in a self-supervised fashion end to end. The superpoint architecture consists of two branch. One branch predicts the keypoints while the other branch predicts the descriptors. The descriptor branch is not used in our SLAM pipeline since it is incompatible with the SLAM backend at the moment. However, the descriptor calculation is still retained in the pipeline since it forces the keypoint detector to learn deep features in an image. The Encoder consists of 4 convolutional layers with 64, 64, 128 and 256 channels respectively. The stride and padding of the encoder are also kept to 1. The keypoint detector head has further two convolutional layers that predict around 600 keypoints and their respective scores. Only 500 keypoints with the highest scores are retained whilst the other are discarded(the post-processing step also has a manual threshold of 0.005).

The training for the superpoint detector is done on the Nvidia GeForce RTX 3050 using the dataset provided by the Magicleap organization for around 35 hours. The training procedure is customizable for indoor and outdoor environments. The procedure is divided into two sub-processes:

- **Synthetic pre-training:** The first step involves training the superpoint detector as a standard corner detector by feeding it a lot of images of geometric objects such as rectangles, ellipses, T-junctions and L-junctions. The images are rendered and labelled automatically and provide ground truth data to the keypoint detector. For artifical environments, superpoint network outperforms classical detectors

such as Harris, FAST and Shi-Tomasi detectors but underperforms in natural images. This caused the authors to further train their network on real-world images.

	MagicPoint	FAST	Harris	Shi
mAP no noise	0.979	0.405	0.678	0.686
mAP noise	0.971	0.061	0.213	0.157

- **Homographic adaptation:** The second procedure involves joint training with the descriptor head. The authors make use of homographic adaptation by which an image is distorted using reversible operations such as rotation, cropping, translation, scaling, in-plane rotation and perspective distortion. An image and its homographic pair should have the same key points and descriptors. Hence, the network is trained on pairs of these images with inverse homographic loss.

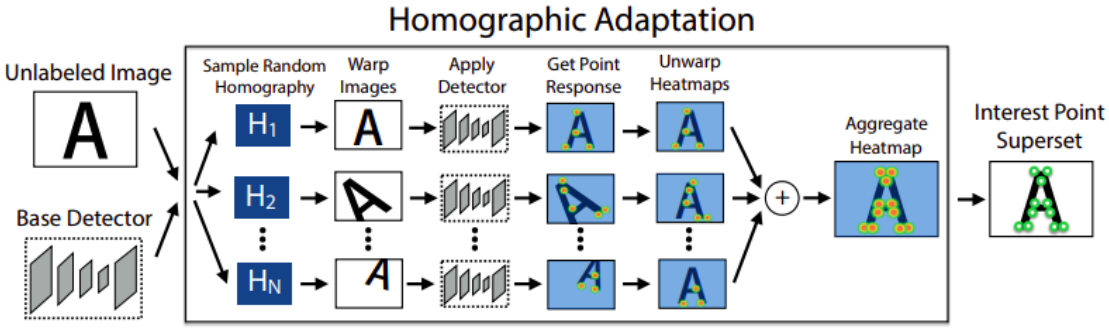


Figure 2.5: Homographic adaptation for superpoint network

2.6 Image denoising

Although the images given to us in the TUM dataset have already been pre-processed to remove outliers, the RGB-D images provided to us by the Orbbec Astra Mini S camera do not provide us with that luxury for depth cameras. Due to the inherent limitations of the structured light technology, depth images suffer from the following artefacts:

1. **Inconsistent depth:** The same object might have different distance measurements due to changes in viewpoint.

2. **Conical artefacts around the camera:** This usually happens when some of the dots in the Structured light pattern are not detected, which causes some points to

appear really close to the camera. If the camera is present in the same area for some time, the point will cluster around the camera obscuring the rest of the key points.

3. **Sharp changes in depth in uniform area:** Due to reasons described earlier, a flat surface might have some outlier points that protrude out of the surface. This causes a lot of problems while optimizing bundle adjustment since these points unfairly penalize the estimated camera trajectory during reprojection.

To rectify the issue, we propose using k-d trees, a Gaussian filter and a small convolutional network to reduce this effect. K-d trees are data structures that are built for fast retrieval and deletion. This means that we can remove outliers from the depth images without any additional costs to our processing speed. K-tree makes a tree data-structure for all pixels where the splitting of the tree is done according to some dimensionality. Outlier points thus lie far from their neighbouring pixels and are thus removed from the image and replaced with the average value of surrounding pixels. Afterwards, we pass the image through a Gaussian kernel to smoothen out any irregularities caused by the substitution of pixels.

A small convolutional network is attached before the superpoint network that acts as another layer of outlier removal. The benefit of using this network is that it reduces irregularities while keeping the edges and resolution of the depth image sharp. The network is a smaller version of DnCNN[Zha+16] network that predicts noise using a residual connection.

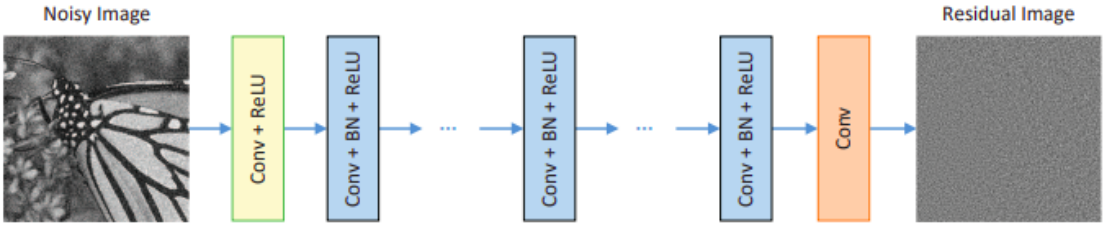


Figure 2.6: Denoising network with 4 convolutional layers

After computing keypoints using the superpoint network, we prune the points at a distance of 0 to 10 cm in front of the camera since they obscure other keypoints in the local map and cause problems in the bundle adjustment process. The backend also performs non-maximal suppression of the local map by using a search radius around each keypoint. This ensures that the map is sparse and informative with no overlapping points that might cause problems in map formation.

2.7 Point-cloud reconstruction

Point-cloud reconstruction for our implementation of visual slam involves three steps. After the frontend signals to the backend that it will send no frames from now on, the backend broadcasts the camera poses back to the frontend. The frontend receives the camera poses and then stitches together the point-cloud by loading the RGB-D data that it received from the Orbbec camera. The camera poses are used to produce a transformation between the different point-clouds. We keep a downsampling rate of 4 to increase the speed of computation and reduce duplicate points. Afterwards, we use the outlier detection in open3d library to remove outliers, before saving everything in a ply file. This ply file can be visualized in a 2D plane in the terminal via termviz(ROS 2.0 tool) or can be imported later on a later and visualized using meshlab.

For more high-quality environments, we also provide the option of passing the point-cloud through the PCN network[Yua+18]. The network predicts the completed pointcloud in chunks to render a point-cloud with fewer gaps and spaces than the one usually returned to us via the depth camera.

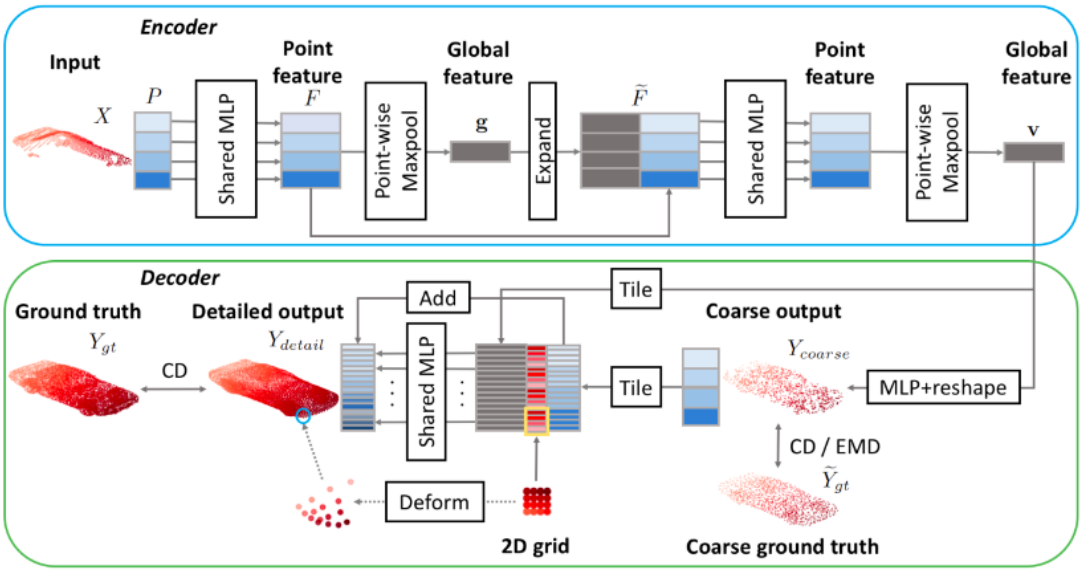


Figure 2.7: Point-cloud completion network

3 Results

3.1 Overview

We conduct our experiments on the TUM RGB-D dataset as described before. The updated dataset has around nine sequences which have an average 56s. We evaluate the RMSE(root mean squared error), translational RMSE, trans RMSE and rotational RMSE for each frame:

Sequence	RMSE	ABS trans error	trans RMSE	ROT RMSE
freiburg1 desk1	0.06630	0.05398	0.104547	3.71211
freiburg1 desk2	0.03724	0.03356	0.07105	4.01544
freiburg1 plant	0.01483	0.01360	0.02058	2.12108
freiburg1 xyz	0.00970	0.00838	0.01512	0.97378
freiburg2 desk	0.07731	0.07606	0.11385	1.46904
freiburg2 pioneer 360	0.09108	0.07556	0.21296	2.31261
freiburg2 xyz	0.01580	0.01401	0.02600	0.59611
freiburg3 long office household	0.02333	0.01895	0.04192	1.28120
freiburg3 teddy	0.11412	0.05999	0.21731	11.48721

Table 3.1: Results of ORB-SLAM on TUM dataset for 500 keypoints

The results for ORB-SLAM using our frontend and backend framework are promising. Even with 500 key points, the RMSE losses are very low. Remember that the RMSE loss of 0.01 corresponds to a distance of 1 cm, perhaps the state of the art regarding orb-slam. However, our ORB-SLAM framework performs very poorly on the teddy bear dataset. On closer inspection, it looks like the dataset has many unfiltered points in the point cloud that cause the Bundle Adjustment optimization to perform poorly in the backend. Using the statistical outlier removal reduces the error for the teddy bear dataset to 0.05. Meanwhile, for the rotational RMSE, ORB-SLAM also performs closely to the state-of-the-art SLAM algorithms such as DVO. A rotational RMSE of 1 corresponds to an error of 1 degree on average. Aside from the teddy bear dataset, all the sequences have a rotational RMSE of less than 4.5 degrees. It is interesting to see that the long office dataset(which is the largest sequence in the TUM dataset), the

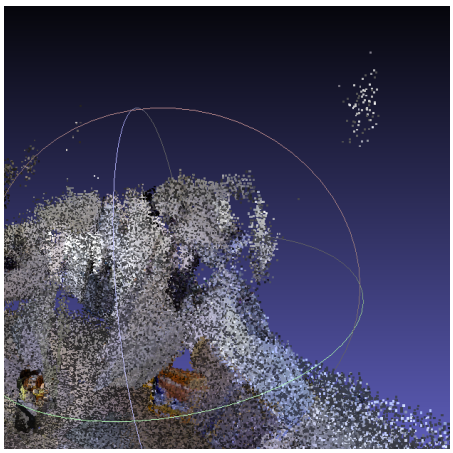


Figure 3.1: The teddy bear dataset with unfiltered points

rotational RMSE is surprisingly very low. This means that as the sequence increases, multiple loop detection calls decrease the rotational RMSE very effectively.

Sequence	RMSE	ABS trans error	trans RMSE	ROT RMSE
freiburg1 desk1	**0.01767**	**0.01576**	**0.03732**	**2.08865**
freiburg1 desk2	0.05462	0.04319	0.08816	4.34751
freiburg1 plant	**0.01284**	**0.01182**	**0.01872**	**1.19003**
freiburg1 xyz	0.00999	0.00863	0.01557	1.00299
freiburg2 desk	**0.07179**	**0.07040**	**0.10909**	**1.38548**
freiburg2 pioneer 360	0.13421	0.12230	0.55219	4.16298
freiburg2 xyz	0.01688	0.014689	0.02659	0.63092
freiburg3 long office household	**0.01280**	**0.00798**	**0.02257**	**0.72540**
freiburg3 teddy	0.12015	0.06321	0.22893	12.09657

Table 3.2: Results of Superpoint-SLAM on TUM dataset for 500 keypoints

The table 3.2 shows the performance of the superpoint slam on the TUM datasets. The superpoint SLAM performs better than the ORB-SLAM framework on 4 out of 9 sequences. It performs better than ORB-SLAM on larger sequences and sequences that have less variation in intensity. This would make more sense since the superpoint learns to predict landmarks regardless of intensity values. However, since we do not make use of the superpoint descriptor, our implementation does not outperform orb-slam on most of the sequences. Furthermore, the descriptor in the backend is optimized to make use of orb descriptors with their corresponding key points. Furthermore, among

the 500 super point key points that we use, for 10 per cent of the key points, we cannot calculate a valid descriptor which has a threshold score greater than 0.1. This ultimately causes the superpoint network to fail. Fig 3.2 shows the teddy bear dataset which has higher intensity changes as compared to the plant dataset in 3.3.

Figure 3.2: (a) RGB image of teddy bear data (b) Laplacian

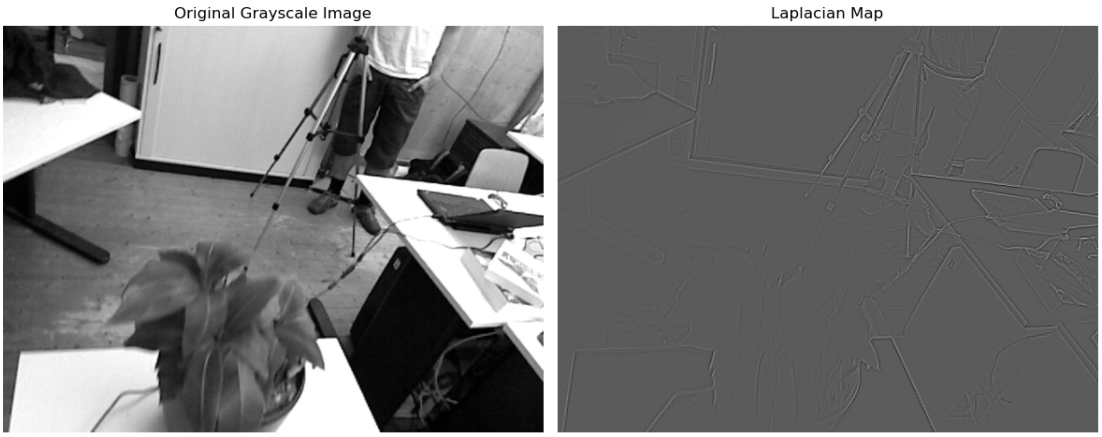


Figure 3.3: (a) RGB image of plant dataset (b) Laplacian

3.2 Ablation studies

We then used our rosedge+orbbec camera to find out the real-time performance of the visual slam. To speed up our pipeline, we employ an offline version of visual slam where we record the sequence we are interested in and then we send the keypoint frames to the main server. Unfortunately, this is the best we can do with the current setup however this might change with the 64-bit version of rosedge in the future. The rosedge then publishes frames at a rate of 2 frames per second for orb keypoints and 1 frame per minute for the superpoint network. We use the following setup for doing visual slam:

Figure 3.4: Visual slam setup

The setup performs well even in bright indoor lighting conditions as shown below. Furthermore, we also get minimal artefacts if the visual slam setup continuously moves without any sudden movements. Furthermore, the setup also works for outdoor environments and does work at night with minimal lighting. To remove the chances of motion blur distorting the visual slam pipeline, we reject rgb images which have a variance of laplacian less than 25.

The figure above demonstrates as the number of keypoints increases, we do not necessarily get an increase in performance, since the RANSAC optimization would have to cater to more outliers rather than inliers within the data. Our script also contains an intrinsic calibration procedure to provide the user with a good visualization of the camera motion. Fig 3.8 shows the effects of outlier removal on the 3D point cloud generated afterwards.

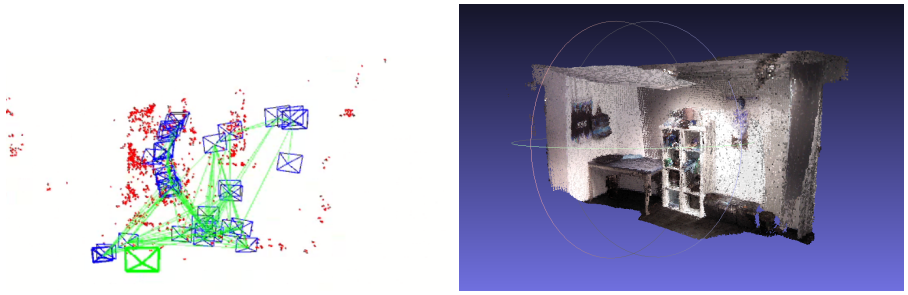


Figure 3.5: (a) Trajectory (b) corresponding 3D model

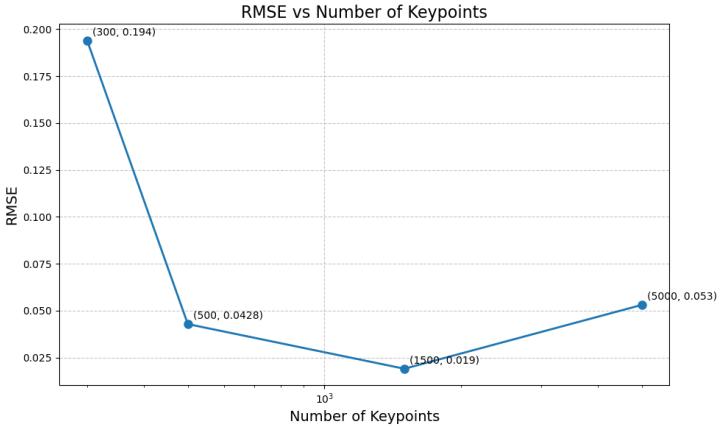


Figure 3.6: keypoints vs RMSE

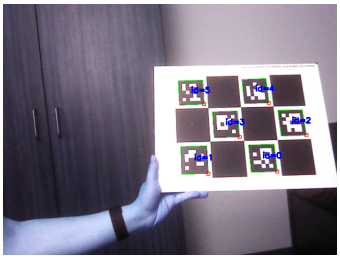


Figure 3.7: Intrinsic calibration

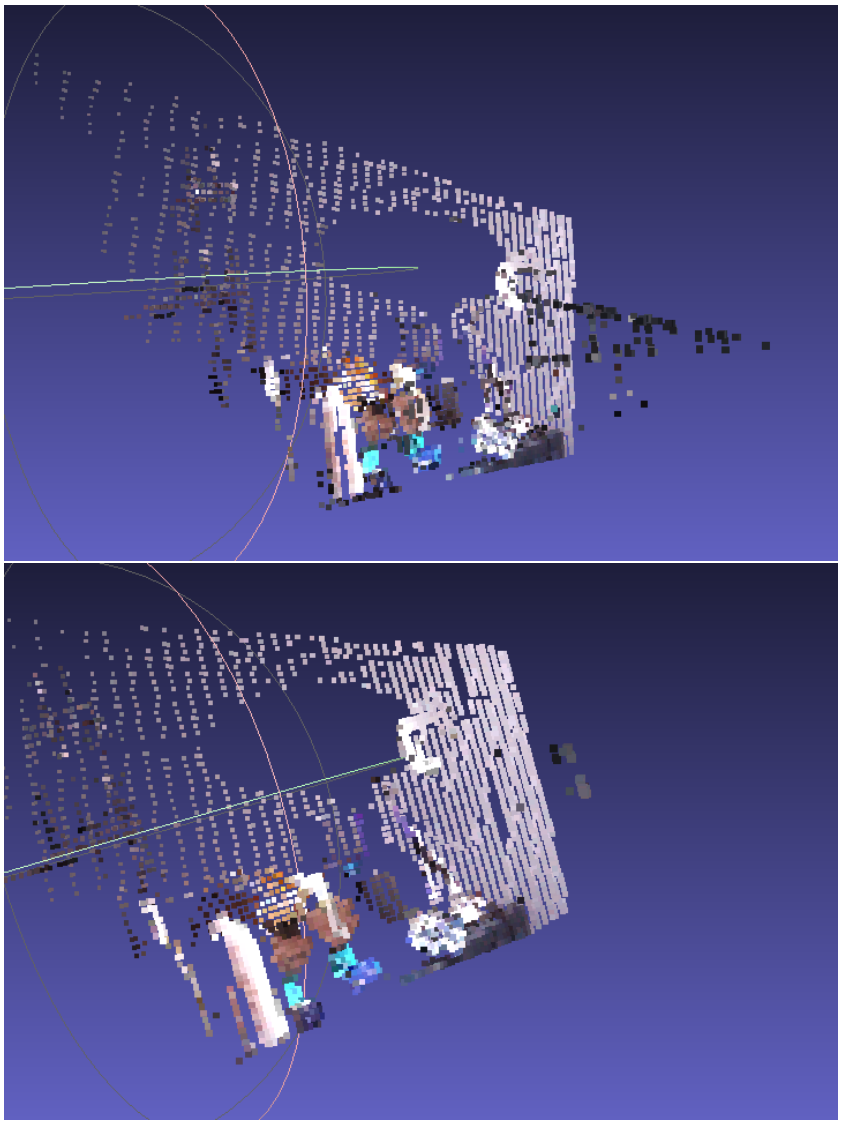


Figure 3.8: Statistical outlier removal before and after

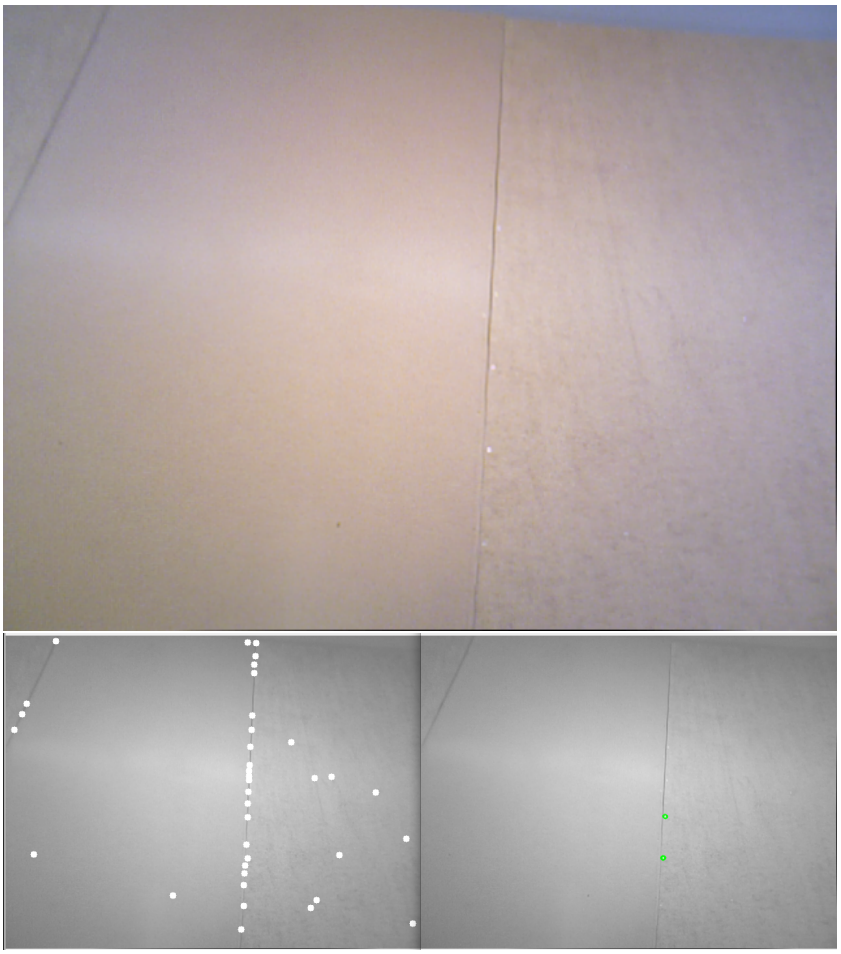


Figure 3.9: (a)Superpoint (b)ORB

3.3 Visual Results

The following contains the 3D-generated images for each sequence in the TUM dataset. The point cloud is displayed using meshlab with default lighting conditions without outlier removal. After all the point-clouds are stitched together on the ros-edge, the user then imports the ply file using scp. In the future, rosedge will support viewing the ply files natively. The 3D-generated images are stored in a ply file using the following format:

```
property float x, y, z
property uchar red, green, blue, alpha
property float nx, ny, nz
```

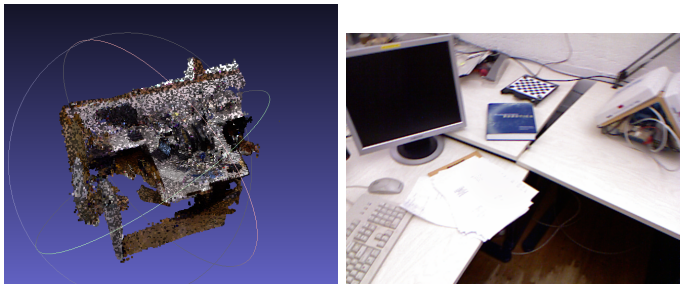


Figure 3.10: freiburg1 desk1

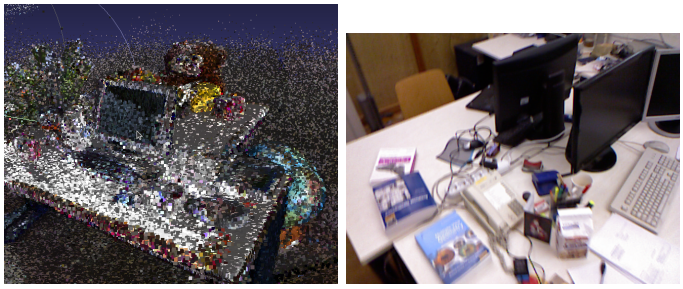


Figure 3.11: freiburg1 desk2

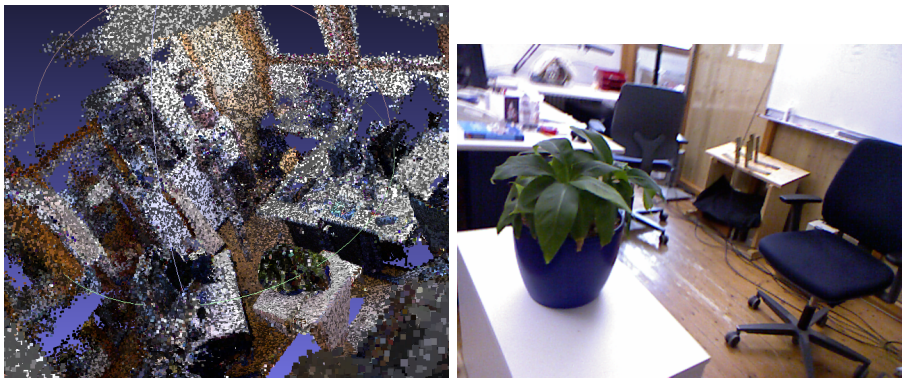


Figure 3.12: freiburg1 plant

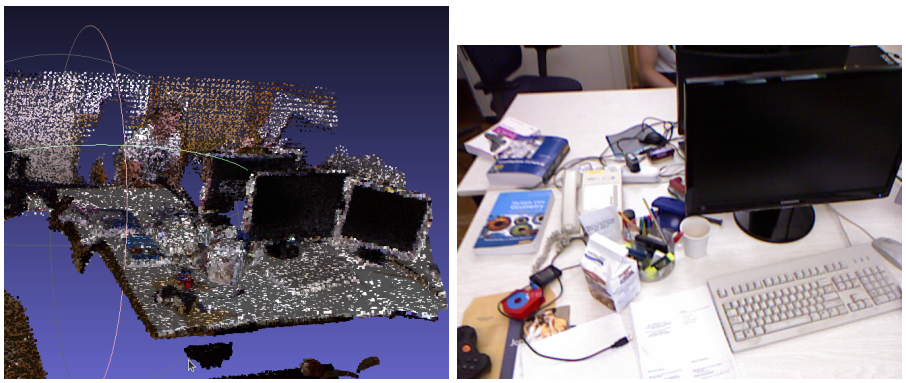


Figure 3.13: freiburg2 desk

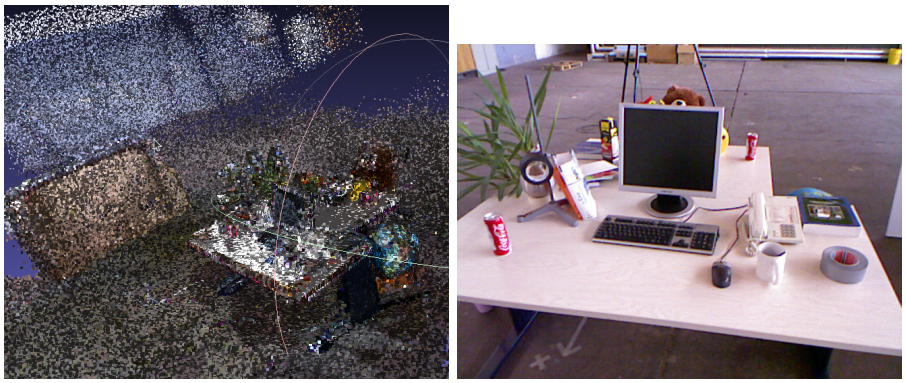


Figure 3.14: freiburg2 desk

Figure 3.15: freiburg2 pioneer 360

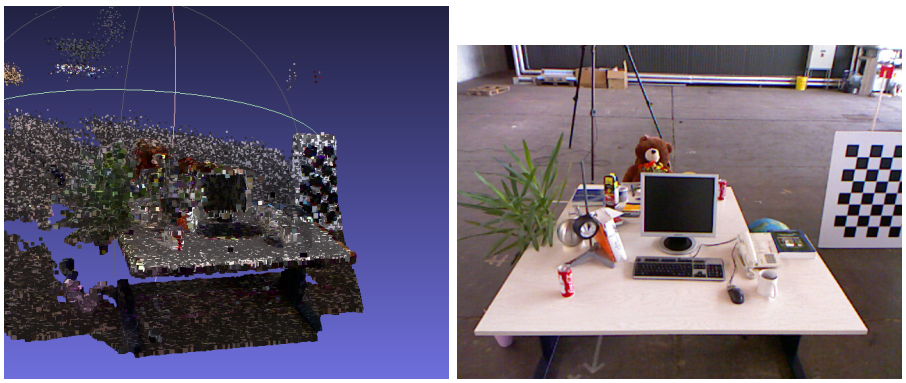


Figure 3.16: freiburg2 xyz

Figure 3.17: freiburg3 long office household

3.4 Outlook and future research

This master's thesis has been a monumental effort in improving our understanding of running visual SLAM on edge-devices. To our knowledge, this is the first real use of visual SLAM on edge devices which involves sharing the computational workload among the edge device and the main server. However, future work would involve a lot of other improvements before visual slam can be a viable algorithm on run-on-edge devices.

1. **Incorporating superpoint descriptors and light glue matcher:** The superpoint keypoint marginally works with orb descriptors and we lose a lot of valuable information about the keypoint itself by not using the native descriptor. In future iterations, users can use the light-glue matcher[LSP23] in the backend instead of the ORB matcher to do keypoint matching. The light glue network uses transformers to perform context-based matching at scale, which is very beneficial for large industrial environments.

2. **Improving the use of TPU on edge device:** As described before, the usage of TPU for real-time image analysis is not up to the mark. By switching to a 64-bit system, the TPU can run on a separate thread while the keypoint publisher works in the background.

3. **Run-time statistical outlier removal:** The statistical outlier removal algorithm is very computationally expensive. Research on reducing the computational time required for outlier removal could greatly improve the accuracy and run-time performance on visual slam.

4. **Improve hardware reliability:** The USB interface of edge-ros is very unpredictable and cuts off communication. Future work could revolve around the use of failure detection and recovery methods that would allow visual slam after connection to the main-server has been restored.

5. **Use wifi-modem with edge device:** Employ the use of of wifi-modem rather than a USB connection to make the product a better market fit for warehouse applications.

Research on these conditions will make it easier to one day make visual slam a staple on edge devices. Furthermore, it would provide Olive Robotics a viable product that makes full use of the limited computing power offered by their excellent hardware.

List of Figures

List of Tables

Bibliography

[] *GitHub - NVIDIA-ISAAC-ROS/isaac_ros_visual_slam: Visual SLAM/odometry package based on NVIDIA-accelerated cuVSLAM — github.com*. https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_visual_slam. [Accessed 09-06-2024].

[Ben+22] A. J. Ben Ali, M. Kouroshli, S. Semenova, Z. S. Hashemifar, S. Y. Ko, and K. Dantu. "Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping." In: *ACM Trans. Embed. Comput. Syst.* 22.1 (Oct. 2022). ISSN: 1539-9087. DOI: 10.1145/3561972.

[CIG23] Y. Chen, H. Inaltekin, and M. Gorlatova. *AdaptSLAM: Edge-Assisted Adaptive SLAM with Resource Constraints via Uncertainty Minimization*. 2023. arXiv: 2301.04620 [eess.SY].

[Dai+16] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration." In: *CoRR* abs/1604.01093 (2016). arXiv: 1604.01093.

[Dav+07] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. "MonoSLAM: Real-Time Single Camera SLAM." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (2007), pp. 1052–1067. DOI: 10.1109/TPAMI.2007.1049.

[DB06] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I." In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.

[DMR18] D. DeTone, T. Malisiewicz, and A. Rabinovich. *SuperPoint: Self-Supervised Interest Point Detection and Description*. 2018. arXiv: 1712.07629 [cs.CV].

[EKC18] J. Engel, V. Koltun, and D. Cremers. "Direct Sparse Odometry." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.3 (2018), pp. 611–625. DOI: 10.1109/TPAMI.2017.2658577.

[End+12] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. "An evaluation of the RGB-D SLAM system." In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 1691–1696. DOI: 10.1109/ICRA.2012.6225199.

[ESC15] J. Engel, J. Stückler, and D. Cremers. "Large-scale direct SLAM with stereo cameras." In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 1935–1942. DOI: 10.1109/IROS.2015.7353631.

[FZ98] A. W. Fitzgibbon and A. Zisserman. "Automatic camera recovery for closed or open image sequences." In: *Computer Vision — ECCV'98*. Ed. by H. Burkhardt and B. Neumann. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 311–326. ISBN: 978-3-540-69354-3.

[Gei+13] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. "Vision meets robotics: The KITTI dataset." In: *The International Journal of Robotics Research* 32 (2013), pp. 1231–1237.

[GK99] J.-S. Gutmann and K. Konolige. "Incremental mapping of large cyclic environments." In: *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No.99EX375)*. 1999, pp. 318–325. DOI: 10.1109/CIRA.1999.810068.

[Iza+11] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera." In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. UIST '11. Santa Barbara, California, USA: Association for Computing Machinery, 2011, pp. 559–568. ISBN: 9781450307161. DOI: 10.1145/2047196.2047270.

[KM07] G. Klein and D. Murray. "Parallel Tracking and Mapping for Small AR Workspaces." In: *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. 2007, pp. 225–234. DOI: 10.1109/ISMAR.2007.4538852.

[Küm+11] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. "G2o: A general framework for graph optimization." In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3607–3613. DOI: 10.1109/ICRA.2011.5979949.

[LD91] J. J. Leonard and H. F. Durrant-Whyte. "Simultaneous map building and localization for an autonomous mobile robot." In: *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91* (1991), 1442–1447 vol.3.

[Leu+13] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart. "Keyframe-based visual-inertial slam using nonlinear optimization." In: *Proceedings of Robotis Science and Systems (RSS) 2013* (2013).

[LMC99] S. Lacroix, A. Mallet, and R. Chatila. "Rover Self Localization in Planetary-Like Environments." In: 440 (July 1999), p. 433.

[LSP23] P. Lindenberger, P.-E. Sarlin, and M. Pollefeys. *LightGlue: Local Feature Matching at Light Speed*. 2023. arXiv: 2306.13643 [cs.CV].

[MT16] R. Mur-Artal and J. D. Tardós. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras." In: *CoRR* abs/1610.06475 (2016). arXiv: 1610.06475.

[NF12] P. K. Nathan Silberman Derek Hoiem and R. Fergus. "Indoor Segmentation and Support Inference from RGBD Images." In: *ECCV*. 2012.

[Nie+13] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. "Real-time 3D reconstruction at scale using voxel hashing." In: *ACM Trans. Graph.* 32.6 (Nov. 2013). ISSN: 0730-0301. DOI: 10.1145/2508363.2508374.

[NLD11] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. "DTAM: Dense tracking and mapping in real-time." In: *2011 International Conference on Computer Vision*. 2011, pp. 2320–2327. DOI: 10.1109/ICCV.2011.6126513.

[NNB04] D. Nister, O. Naroditsky, and J. Bergen. "Visual odometry." In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 1. 2004, pp. I–I. DOI: 10.1109/CVPR.2004.1315094.

[PC05] M. Pupilli and A. Calway. "Real-Time Camera Tracking Using a Particle Filter." In: *British Machine Vision Conference*. 2005.

[Ray+23] L. S. S. Ray, B. Zhou, L. Krupp, S. Suh, and P. Lukowicz. *SynthCal: A Synthetic Benchmarking Pipeline to Compare Camera Calibration Algorithms*. 2023. arXiv: 2307.01013 [cs.CV].

[RCM14] L. Riazuelo, J. Civera, and J. Montiel. "C2TAM: A Cloud framework for cooperative tracking and mapping." In: *Robotics and Autonomous Systems* 62.4 (2014), pp. 401–413. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2013.11.007.

[Sal+13] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. "SLAM++: Simultaneous Localisation and Mapping at the Level of Objects." In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 1352–1359. DOI: 10.1109/CVPR.2013.178.

[Stu+12] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. "A benchmark for the evaluation of RGB-D SLAM systems." In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 573–580. DOI: 10.1109/IROS.2012.6385773.

[Sun+21] Q. Sun, J. Yuan, X. Zhang, and F. Duan. "Plane-Edge-SLAM: Seamless Fusion of Planes and Edges for SLAM in Indoor Environments." In: *IEEE Transactions on Automation Science and Engineering* 18.4 (2021), pp. 2061–2075. DOI: 10.1109/TASE.2020.3032831.

[Tat+17] K. Tateno, F. Tombari, I. Laina, and N. Navab. "CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6565–6574. DOI: 10.1109/CVPR.2017.695.

[TBF05] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005. ISBN: 0262201623 9780262201629.

[Whe+16] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger. "ElasticFusion: Real-time dense SLAM and light source estimation." In: *The International Journal of Robotics Research* 35.14 (2016), pp. 1697–1716. DOI: 10.1177/0278364916669237. eprint: https://doi.org/10.1177/0278364916669237.

[Wu+18] Y. Wu, Z. Li, P. Shivakumara, and T. Lu. "Em-SLAM: a Fast and Robust Monocular SLAM Method for Embedded Systems." In: *2018 24th International Conference on Pattern Recognition (ICPR)* (2018), pp. 1882–1887.

[Yua+18] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. "PCN: Point Completion Network." In: *CoRR* abs/1808.00671 (2018). arXiv: 1808.00671.

[Zha+16] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising." In: *CoRR* abs/1608.03981 (2016). arXiv: 1608.03981.