Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Semantic Mapping for Enhanced Localization in Indoor Environments

Allen Isaac Jose

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr.-Ing. Sebastian Houben
Deebul Nair, M.Sc.

October 2023

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

07-10-2023

Date

Allen

Allen Isaac Jose

Abstract

Mapping the environment is an important task that needs to be achieved by Autonomous Mobile Robots(AMR) to localize itself within the environment at a later point in time. The mapping problem is itself a SLAM problem as localization needs to be performed in the uncertain map that is being generated iteratively as the ground truth location of the robot is not available. The traditional 3D SLAM methods use the geometrical structure of the environment to produce a dense or sparse map. Dense maps necessitate a large memory space of gigabytes to hold the map, which is impractical for robots with low memory capacities. Sparse maps created of keypoint features can often be occluded or not detectable due to viewpoint variations and they lack the semantic information of the scene.

In this project, we are focusing on exploring the 3D SLAM algorithms that can create sparse maps enriched with semantic object models that can better explain the scene at a low memory requirement. The two algorithms chosen for performance comparison are QuadricSLAM and OA-SLAM, both of which use ellipsoids to represent objects in the scene. The inputs to the algorithm are RGBD images and odometry(QuadricSLAM) or RGB(OA-SLAM) images. A deep-learning-based classification model is used to predict the label and bounding box of the objects which are then tracked over continuous frames and photogrammetric methods are used to enclose the object within a closed region to be represented as an ellipsoid. In the evaluation part, traditional SLAM comparison methods perform the quantitative analysis of the accuracy of camera trajectory and only perform the qualitative analysis of the generated map. This is mainly because of the lack of ground truth 6DOF pose of the objects in the scene.

For the comparative evaluation, a dataset of 10 scenes with YCB-V objects is generated using a Blender-based rendering tool. Further, a predefined trajectory is traversed by the camera to generate the dataset in BOP format(used for 6D object pose estimation benchmark). Metrics are defined to perform quantitative analysis of trajectories and objects.

The runtime performance of around 15 FPS for OA-SLAM indicates its usability in real-time applications as compared to QuadricSLAM at less than 10 FPS. Since the OA-SLAM was operating on monocular images, the scale drift problem is evident in the generated map. This problem was not observed in QuadricSLAM as the RGBD image has the depth information of the pixels. However, it should also be noted that relocalization from abrupt camera motion is not possible in QuadricSLAM as it lacks point features. Localization mode was only available in OA-SLAM which showed that the objects can aid in the localization via the PnP algorithm as they can be detected from different viewpoints and in different illumination conditions. A random set of images from the scene is provided as input and the camera pose is localized within a single frame. The generated size of the map was around 5.5 MB for OA-SLAM as it included the ORB-feature points along with the object model parameters & keyframe pose and 454 KB for QuadricSLAM with camera poses & object model parameters.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to Prof. Dr.-Ing. Sebastian Houben for his exemplary supervision of my R&D project. His commitment to scheduling monthly meetings and providing invaluable insights played a pivotal role in steering my project in the right direction. Furthermore, his prompt feedback on the report proved instrumental in helping me rectify errors and produce a valuable final report.

I would like to extend special thanks to my second supervisor, Deebul Nair, who personally assisted me in formulating an accurate problem statement from my initial undirected thoughts. Deebul's in-person guidance was invaluable in resolving coding issues when I encountered difficulties and steering me in the right direction. His tremendous support was evident in his consistent availability for in-person meetings.

I am deeply indebted to Sathwik Panchangam for his invaluable assistance in modifying his Blender-based dataset generation tool to create the dataset for SLAM. His support was not only limited to tool modification but also extended to helping me resolve challenging bugs in the code whenever I encountered them.

I would like to express my sincere gratitude to my family for their unwavering support and motivation, which has been instrumental in encouraging me to pursue my career.

Lastly, I would like to extend my heartfelt gratitude to both the university and the MAS department for granting me access to a GPU-enabled workstation, which was indispensable for the generation of SLAM datasets.

Contents

1

Introduction

1.1 Motivation

A brief overview of SLAM from the project proposal: *The number of Autonomous Mobile Robots(AMR) being deployed into the home and industrial environments is increasing day by day. This is due to the fact that they can be reliable in performing the task assigned to them redundantly in a robust manner. For this to happen, the robot should have an idea or knowledge of the structure of the environment so that it can plan its actions accordingly. This problem in the field of robotics is known as mapping. Mapping is the process of creating a virtual model of the real environment that can be used by robots for various applications such as localization, navigation, manipulation and planning. Another important requirement is the ability of the robot to be able to locate itself within the map it has created. This problem is known as localization. It has to match the observation from the sensors with the structure or objects within the available map to identify correspondences and calculate its pose with respect to the map of the environment. When the ground truth pose/location of the robot is given, the task of mapping could be done directly and similarly, when the map of the environment is provided, the task of localization can be easily achieved. But in reality, when the robot needs to explore a new environment, it has to perform both localization and mapping at the same time. This well-known problem is known as Simultaneous Localization and Mapping(SLAM). In SLAM, we need to estimate the pose and generate a map from the observations simultaneously. The main challenge is that the action model and observation model of the robots are not perfect. That is, there is uncertainty in the final pose of a robot given a command for action and similarly, there is uncertainty associated with each sensor when observing an environment. These uncertainties of sensors need to be taken care of, especially in the case of fusing the measurements from odometry-related sensors and observation-related sensors to make a map with the least errors.* In this project we would be focusing only on the indoor SLAM methods.

One of the most important questions that is asked is, for what purpose are we using the created map? Some of the most common tasks that utilize the map are localization, navigation, manipulation and task planning. In localization, the map should contain geometric and semantic structures so that the current observation can be matched against the features in the map to estimate the most likely position or pose of the robot. In the earlier SLAM algorithms, only the geometric structure of the environment is mapped. In sparse mapping, some keypoint features are extracted and stored whereas in dense mapping, the exact

shape of the scene geometry is reconstructed. This is where the next problem appears which is the storage of the map created. In the 2D realm, dense mapping is promoted as the exact geometry can be captured using 2D LIDAR and the created map is of a small size comparable to an image. In a 3D world, this would be denser and the storage and processing of the entire point cloud becomes a challenge where sparse mapping is preferred. These keypoint features can be used in localization using the bag-of-words approach. Semantic knowledge was mostly incorporated only in 3D dense maps through raycasting the per-pixel semantic segmentation performed on the images. This led to the requirement of having coarse object models that can represent the object's pose, shape and label with few parameters to add semantics in sparse maps. Cuboid [1] and ellipsoid [2] [3] are two such shapes which require only 9 parameters, 3 each for position, orientation and size.

In the navigation task, even though robots that are restricted to planar motion require a 2D occupancy grid [4] map for collision-free motion, there can be situations where an obstacle may not lie on the ground but protrude in the robot's path. Since these obstacles are not visible in 2D occupancy grid maps, the robot may assume a safe motion whereas in reality, collision happens at the top body parts of the robot. So, a 3D occupancy grid map is needed. This can be easily created from the dense map by voxelizing the point clouds. But as discussed before, the storage requirements pull us back to sparse maps. However, in sparse maps, all the objects may not be well represented by feature points and the features cannot also be detected on textureless surfaces. Here, we could utilize the coarse object model to be voxelized to create 3D occupancy grid maps. If all the objects in the environment are well-mapped, even drones can use this map to navigate safely through the environment. Normal navigation can be turned into semantic navigation by utilizing the knowledge of each object it passes by. For example, in an industrial indoor environment, the distance to some dangerous objects can be defined such that the robot ensures it keeps a safe predefined distance from such objects.

In the planning task, the semantic knowledge of at which location in the environment the object is present can help in path planning. This location can be an absolute location or a relative location. When only 1 instance of the object is present in the scene, then the path planning can directly access the absolute location. But when multiple instances of the same object are present, the knowledge of the relative position of the queried object with respect to other objects in the environment can be utilized. For example, to retrieve a glass, the command can be "retrieve the glass on the table" or "retrieve the glass in the sink". A semantic object-oriented map enables easy querying of relative positions.

In manipulation tasks, the normal object detection model may output noisy object 6DOF pose or the 2D bounding box in an unstable fashion. The object-oriented maps help reduce the uncertainty by fusing all the observations and thereby giving a stable 6DOF pose of the object along with the quadric shape(cuboid or ellipsoid) enclosing it. Such a stable object-oriented map not only helps in the robotics domain but is also usable in Augmented Reality(AR) applications. Virtual objects can be placed on top of the mapped object models and the advanced relocalization capability of sparse feature maps aided with object models can ensure stable scenes in AR.

These factors led to the need for a different mapping strategy where we can efficiently map and store the map and still perform all the basic AMR needs. Another domain that grew in parallel to SLAM

was the deep learning models which helped to provide object classification predictions with real-time performance. As a result, a new semantic SLAM strategy was developed where objects in the indoor environment can be mapped and represented using coarse models such as cuboids and ellipsoids and a label can be assigned to them. In this project, we are focusing on comparing the performance of 3D sparse object-oriented semantic SLAM methods where RGB or RGB-D images are used as input to the SLAM framework to map the environment. We are aiming to compare its runtime performance along with the accuracy of the map and the trajectory through which the camera moved to map the environment and validate on a common dataset.

1.2 Challenges and difficulties

The main challenge with the SLAM algorithm is its ability to run in real-time. This requires the tracking component which takes in the current image not to be blocked by any slower map optimization steps. For the SLAM to be able to scale to larger environments it should be able to remove redundant information in the map. This includes the keyframes, map points and objects. The SLAM should also be able to recover after losing track of the pose due to abrupt motion through relocalization. Irrespective of the camera viewpoint in the scene, global relocalization should be possible in the available map for an SLAM to be useful in localization tasks. Data association is another popular challenge in the SLAM algorithm where the 2D bounding box detections between frames should be associated with the same object or initialised as a new object. Wrong data associations can lead to duplications of the same object or merging of multiple objects. Additionally, the assumption of a static world is made which may not be always the case in the real world.

The availability of a good-quality dataset is lacking in object-oriented SLAM. This is due to the fact that the ground truth 6DOF pose and the dimensions of the object are not available. Even though the YCB-Video dataset (ref) is available, it is not intended for the SLAM operation. One reason is the low quality of the images in which feature extraction is not made possible by the noisy pixels. To create a sparse map, stable keypoints are a requirement in the map initialization process. Another reason is the absence of a loop closure trajectory. The currently available trajectory covers only a part of the scene. For coarse object models to be refined well to the shape of the object, images with significant parallax are needed.

The previous SLAM comparison methods mainly focused on performing the quantitative analysis of the trajectory and performed only the qualitative analysis of the map being generated. In our case, in order to perform the quantitative analysis of the created object models with the ground truth, new metrics have to be defined. A challenge is the difficulty in estimating the rotation error of the estimated object with respect to the ground truth object. In the case of ellipsoids, if more than 1 axis is of comparable size, then the axes of the estimated ellipsoid may interchange with each other causing the comparison with the ground truth returning very high errors. For example, the x, y, and z axes in the ground truth object may correspond to the y, z, and x axes in the estimated ellipsoid. Another challenge is how accurately can we measure the overlap of the estimated ellipsoid with the ground truth object. The ground truth shape is a cuboid and the estimated shape is an ellipsoid which requires Monte Carlo sampling to be performed

within the cuboid and checking if the point lies within the estimated ellipsoid.

In the SLAM algorithm which utilizes monocular RGB images for map generation, there is no available information on absolute depth scale. The mapping is done relative to an arbitrary scale. For the comparative evaluation, we need to perform a similarity transformation that can align the estimated map with the ground truth map. The rescaling could also be performed by comparing the ratio of the estimated object dimension and the ground truth object dimension of the YCB object which we have. Another issue with monocular RGB images is the map initialization issue which is due to the presence of a dominant planar surface in the scene. This can cause the initial poses to be unstable. In ORB-SLAM [5] based SLAM algorithms, only the keyframes(not all frames) are retained within the map that can be saved at the end which is challenging to compare with the ground truth trajectory. An alternative approach is the use of pose for the individual frames which is returned by the local tracking component of the SLAM.

1.3 Problem statement

In this project, we will perform the comparative evaluation of the two semantic object-oriented mapping algorithms such as QuadricSLAM [3] and OA-SLAM [2] that utilize ellipsoids to represent objects. A literature survey of existing SLAM methods is to be done to justify what led to the requirement of semantic object models in 3D sparse maps. A brief overview of QuadricSLAM and OA-SLAM is to be explained based on its internal framework. To compare both algorithms on a common dataset, a new dataset consisting of loop closure trajectory is to be generated using a blender-based rendering tool. The ground truth trajectory and object pose information is stored in BOP(ref) format and since it is a common benchmarking format, others can also benefit from these datasets. Metrics for quantitative evaluation of the trajectory and the object poses are to be defined to compare the mapping accuracy. We also need to check its runtime performance to validate whether it could be useful for real-time applications. Also, need to perform a system profile evaluation to check its compatibility with limited computational resources. To compare the mapping accuracy, the ground truth 2D object bounding boxes are to be passed into the framework. To validate the robustness of the object shape estimation, another experiment with noisy bounding boxes(by varying their centre point, width and height) is to be performed. Further, we need to check how fast can the algorithm localize in an already mapped environment by running another experiment with a random set of images from the scene. Finally, we need to check the amount of memory required to save the map with all the necessary data to load the map again.

2

State of the art

2.1 2D SLAM

The 2D SLAM involves creating a two-dimensional representation of the world. The robot is assumed to be moving in a planar region so that the motion is constrained to two dimensions. So the features or landmarks in this scenario are the obstacles which act as point features. In the 2D map being created, each cell is identified as occupied, free or uncertain. To estimate whether a cell is occupied or free, we may need range sensors such as a 2D LIDAR sensor or RGB-D camera from which the depth at a particular height is extracted. These constitute the observations. The motion that is performed by the robot is sensed using wheel odometry or using inertial units. Ideally, if there is no error within the sensors, the odometry data can be directly used to estimate the current pose of the robot and the range sensors can be used to map the environment. This is called dead-reckoning where the robot state is estimated from only odometry measurement. But, in the real world, drift error may accumulate over time in the odometry sensor and this will affect the mapping accuracy also. There is also an error associated with the range sensors. To fix that, the SLAM is modelled as a probabilistic model where the uncertainty in the estimated pose and uncertainty in the observations are modelled as correlated and a Maximum-a-Posterori estimate of robot pose and map are found which are consistent with the least error. Loop closure is an important term in SLAM where the robot revisits the same place it has seen before thereby correcting all the measurements taken till that time. SLAM can be seen as an exploration-exploitation scenario where new frontiers are explored which may increase the uncertainty over time and revisits a previously mapped region to exploit the information for loop closure thereby reducing the accumulated uncertainties. In a SLAM framework, usually, there is a front-end and back-end sections. In the front end, the measurements of the odometry and observations are used for tracking and are stored as a data structure, whereas in the backend, a Maximum-a-Posteriori estimate for the robot poses and the map is performed iteratively to reduce the error accumulated by the front end. As the deep learning domain emerged, people have also started adding semantic information to the 2D map by projecting the object label predicted by the classification models onto the region where the object is occupied in the map. These methods will be further described in the next section.

2.1.1 2D Map types

Metric maps

Metric maps try to exactly replicate the spatial information of the environment using range measurement sensors. In the case of two-dimensional representation, the occupancy grid [4] is a popular mapping approach where the environment space is divided into discrete grids and the occupancy of the cell is computed. It can be free, filled or uncertain. Since the ranging sensor has uncertainties in its measurements, the observations for the same grid cell over multiple views should be probabilistically updated. A Bayesian filter is used which recursively estimates the probability of the state of the cell and the previous value of the cell in the map acts as the prior in the recursive update. The inverse sensor model is assumed to have Gaussian noise. Also, the state estimation for each grid cell is performed independently thereby reducing the computational complexity even though the grid cells that are close by may be correlated if they contain geometry of the same obstacle. For each grid cell, two states are represented as O(occupied) and F(free). After all the measurements have been updated, the final state of the cell is determined using the maximum-a-posteriori decision rule [4]. That is, a cell is occupied if the probability of state O is greater than that of state F and a cell is free in vice versa condition. If both probabilities are comparable, then the state of the cell is set to uncertain. Since the sum of probability for O and F states is 1, we need to store only one probability at a time. Usually, the probability of the cell being occupied is stored. There can also be other thresholding methods that could be used based on the sensor properties. Also, since this type of map representation can incorporate uncertainty in its state estimation, rather than discrete binary representations(0 or 1), other representations can also be used based on the application. The occupancy grid map is stored in PGM (Portable Gray Map) format which is a grayscale image with values ranging from 0 to 255. The probability can be obtained by normalizing. This type of map is most commonly used in localization and navigation applications because of its ease of storage, retaining uncertainty and efficient processing for both path planning and particle filter-based localization.

Figure 2.1: Occupancy Grid Map(link)

Topological maps

A topological map is another compact form of representation which could be visualized as a graph where the nodes represent the free spaces and the edges indicate the traversability between the spaces. If the occupancy grid could be seen as pixel-level discretization, a topological map could be considered as block-level discretization. It is most commonly used in navigation applications for planning a global path from region A to region B and the graph data structure can be exploited by the search algorithms such as A^*. An example of a topological map is topomap [6] which is represented in the figure below 2.2. It is generated from a sparse feature-based 3D map using a monocular camera. Occupancy status from the 3D voxel grids in the map is used to identify the free regions and the region-growing approach is used to form the convex(closed) space and is represented as a node.

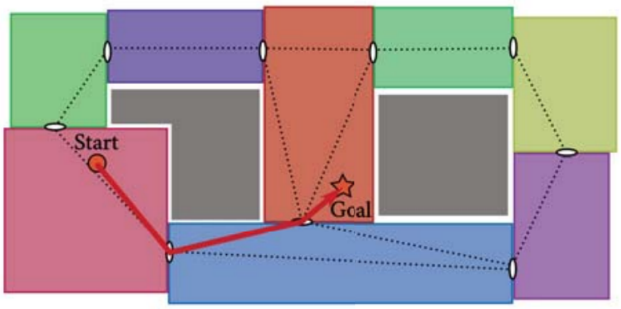

Figure 2.2: Topology Map [6]

Topological maps could also be generated from 2D occupancy grids through Voronoi decomposition, followed by forming topological regions using critical points and critical lines [7]. These polygonal regions can be represented as nodes in a topological graph for computational purposes. As topological maps do not usually exist alone, they are often used along with other map types thereby forming Hybrid maps. So more examples of topological maps can be seen in the section on hybrid maps.

Semantic maps

The field of semantic maps grew with the emergence of deep learning models. In the other map representations, we are trying to replicate the geometrical or spatial information of the environment which lacked any form of explainability. Adding the semantics of the objects in the room and the place category information of the room itself can enable better semantic navigation and planning tasks. For example, to navigate from room A to B, there can be two paths. A metric map will return path 1 as it is the shortest one, whereas, the semantic map can return path 2 if safety is a higher priority than the shortest path. Thus, semantics helps to perform reasoning in a lot of planning tasks which enables a robust coexistence of robots along with humans in indoor environments. In metric maps, each cell is uncorrelated whereas in semantic maps, we could say that the occupied cells in a certain region belong to an object X. This helps in understanding scene change detection where an object is removed or newly added to the environment.

Also, for human visualization, a semantic map is more meaningful than a floor plan map(metric map). In this type of mapping, along with a range sensor, an RGB image is also needed to perform object classification tasks. The semantics can be of two types, one is the category of the room and the other one is the classification of objects in the room.

In the 2DLaserNet [8], 2D LIDAR data is used to classify a region as a room, corridor and doorway. As compared to other deep learning models for point clouds like PointNet, in 2DLaserNet, the spatial or geometrical information of the points is also taken into consideration. In PointNet++, the neighbouring points are identified through feature extraction from the individual points, whereas in 2DLaserNet, the neighbouring points are identified based on the successive laser beams that have closeby angular distance in each laser scan. In the network of 2DLaserNet, 1D convolutional layers are employed for feature extraction followed by fully connected layers for class prediction. A sample semantic map for place recognition is given below 2.3.

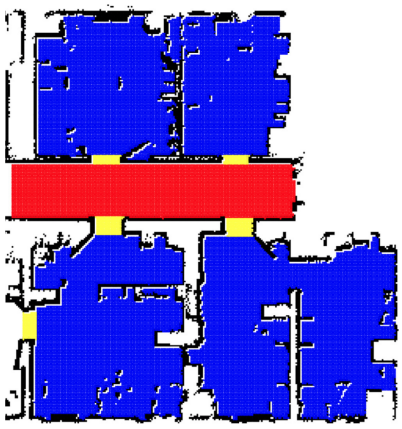

Figure 2.3: Semantic Place Category Map - room(blue), red(corridor), yellow(doorway) [8]

Another indoor place categorization approach [9] utilizes the RGB image along with the 2D laser data to predict the scene among 205 trained scenes. Places_205 [10] convolutional network is used to classify the place from the RGB images. The closed set assumption of the classes is removed here and newer classes can be learned online through one-vs-all classifiers which enable life-long learning. The semantic map that is generated has as many layers as the number of class labels and an additional layer 0 which is an occupancy grid map is also present. All the semantic layers encode the probability of the particular grid cell in the occupancy map being the respective class. The label is propagated from the RGB image into the occupancy grid map by projecting the label into the unoccupied cells that are detected by the laser beams overlapping with the current RGB image. Further, bayesian filtering is applied to fuse temporal noisy predictions and finally arrive at a maximum posterior solution. An example map is given below 2.4.

Figure 2.4: Semantic Scene Category Map [9]

To add semantics of objects into the map, an approach by Zimmerman et al. [11] utilized the prior floor plan of the environment instead of the occupancy grid map and projected the label of the objects onto the 2D map as rectangular boxes. The aim of this map is to provide semantic cues to aid in global localization. Room category could also be predicted based on the knowledge of objects within the room which can aid in better initialization of particles in particle filter-based localization. In a real-world scenario, objects like a table, or chair may change their position and orientation slightly over time and a metric map cannot understand such changes as it has no correlation between grid cells. The 3D bounding box of the object is created using the left and right corners of the 2D bounding box from the RGB image and projected to a depth that is informed by the 2D lidar points. A visibility map is created in which each unoccupied cell stores a list of objects that are visible from the cell and also the viewing bearing vector. This is later used in Monte Carlo localization when the update step is computed based on the observation. When a particle which represents the robot's pose views an object with label l, if there exists a bearing for that object in the cell in which the robot is, then the cosine similarity is performed. The proposed method of adding semantic cues into a 2D floor map improved the global localization, especially in the scenario where multiple rooms are similar in geometry.

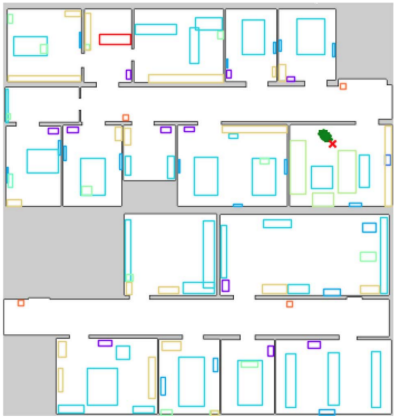

Figure 2.5: Abstract Semantic Map [11]

Hybrid maps

In a real-world scenario, there can be various operations that an AMR should perform and each map explained above may be useful in different scenarios. This led to combining different maps that can overlap with each other and can be used based on the requirement. The semantic map explained above is usually a hybrid map as the semantic label is added on top of the occupancy grid map. A larger indoor environment with multiple rooms can be represented as a topological map where nodes represent the rooms and for each room, a dedicated occupancy grid map could be stored. This enables scalability and manageability while updating a smaller part of the environment rather than updating a single map containing all the rooms.

A multi-hierarchical semantic map [12] approach was proposed where two parallel levels of maps are stored for spatial and semantic knowledge respectively as seen in the figure 2.6*a* below. Anchoring is a procedure where the symbolic knowledge or semantics are connected from the semantic hierarchy of maps to the sensed geometries of the spatial hierarchy of maps and this process is called symbol grounding. The spatial one is useful for navigation and the semantic one is for reasoning and planning. In the spatial hierarchy, the top level represents an abstract node, the middle layer consists of different regions that are connected to the abstract node and the bottom layer contains the sensory data such as the occupancy grid map and the images of the regions in the above layer. In the conceptual hierarchy, everything is derived from the top class called the Thing and in further levels below, the categories are defined and finally, the instances of such categories are generated. It is similar to abstract class, classes and object instantiation in the software paradigm. On the right figure 2.6*b*, the spatial hierarchy is much more visible where the occupancy grid map of each room is represented separately and connected via nodes.

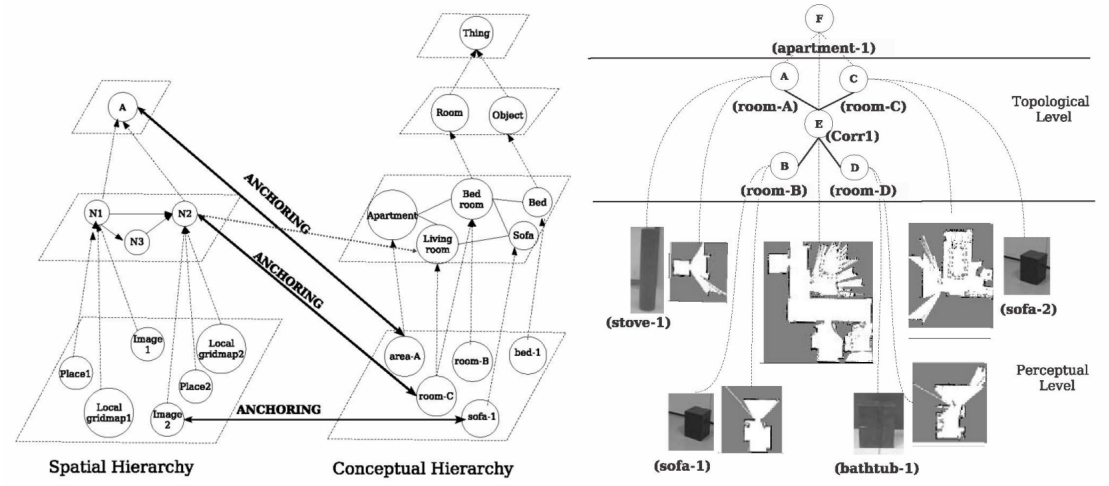

Figure 2.6: a.) Multi-hierarchical semantic map b.) Spatial Hierarchy [12]

A probabilistic way of establishing hierarchical relationships utilizing heterogeneous sensor modalities is implemented by Pronobis et al. [13] as seen in the figure below 2.7. There are three levels. The

bottom layer contains the sensed spatial properties of the environment in the form of metric maps and landmark features. In the middle layer, there exists a place layer and a categorical layer. In the place layer, each explored region is identified as *place* and unexplored regions are identified as *place_holders*. In the categorical layer, the properties of places and objects such as shape, size, model, and appearance are linked from the top knowledge base to the observed features of the environment. The top layer is the conceptual layer where the general knowledge is represented as an ontology and concepts are instantiated for each observed feature in the environment and represented in a probabilistic chain graph model. The probabilistic method allows us to arrive at a consistent result based on fusing the confidence probability for each instance of the concepts as there is uncertainty associated with each sensed information. This type of representation also helps in predicting the properties of unexplored regions marked as *place_holders*.

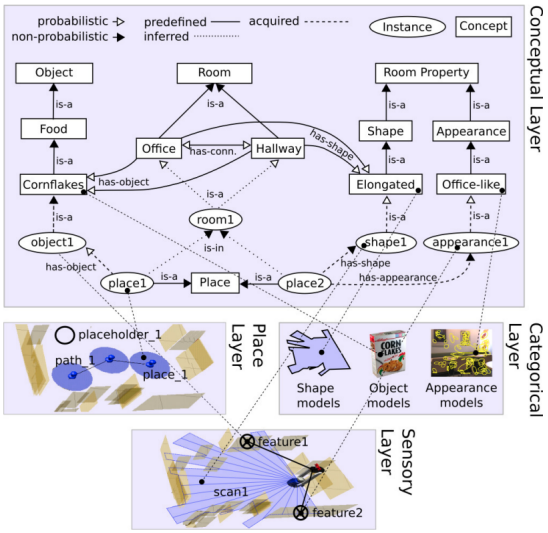

Figure 2.7: Probabilistic Hierarchical Chain Graph Model [13]

2.1.2 2D Mapping methods

Filter-based Mapping

The filter-based mapping consists of utilizing either the Kalman filter or a Particle filter approach to jointly estimate the robot poses and the observations or landmarks iteratively. Kalman filter uses the Bayes formula to estimate the posterior state for the robot pose s_t and the map m using the odometry z and observation u measurements received so far $[p\left(s_t, m \mid z^t, u^t\right)]$ [14]. For a 2D environment, the robot pose at time t (s_t) consists of the x, y and orientation parameters and the map is made of the x and y positions of landmarks. The motion model and observation model are assumed to have a Gaussian noise which is an unimodal distribution. The Gaussian distribution has a dimension of d where d is a state vector of length $2 * l + 3$. This includes the 2 dimensions of each landmark l and the 3 robot pose dimensions. The estimated state of the world(both robot pose and the map) can be obtained from the

mean μ and covariance ϵ of the posterior Gaussian distribution. There are three steps in state estimation which are prediction, Kalman gain computation and correction. In the prediction step, the μ and ϵ of the previous posterior Gaussian distribution are updated with the current motion measurement. Further, the Kalman gain is computed which determines the confidence or uncertainty based on the covariance matrix of the predicted state. The final step is the correction where the Kalman gain governs how much of the difference between the measured and expected state should be combined to form the posterior distribution. Kalman gain tries to give more importance to the data which is more confident.

However, in reality, the motion model is not linear as there is an angular parameter in the robot pose and this assumption fails. Also, since the landmark is observed at a bearing angle, there is also a non-linearity here. This led to the formation of the Extended Kalman Filter(EKF) which is most commonly used in all SLAM applications. In EKF, the non-linear motion model is approximated into a linear one through first-order Taylor series expansion. Previously, in the update step and correction step, linear functions were used. Now they are replaced by non-linear functions that try to linearise it around the estimated mean μ and using the Jacobians or slope computed at the linearization point. As said before, the dimension $d = 2 * l + 3$ may increase as more and more landmarks are added to the map and the computational complexity for matrix multiplication and inversion would also increase. Using efficient data structures and taking advantage of the sparsity of the matrixes, this computational time can be brought down. Also, the outlier map points are removed and only robust landmarks are retained within the matrix. In this type of state estimation, the full posterior is computed over the entire map and robot states which may be computationally expensive but can result in a good convergence. The main challenge in Kalman filter-based estimation is the correspondence problem where the landmarks observed by the sensors fail to associate with the known landmarks thereby creating a multi-modal distribution posterior. Distinct landmarks and strong correspondence are the requirements in EKF. Since landmarks are correlated, viewing more and more landmarks makes the map more accurate as the uncertainty is reduced. Correct data association is an important requirement for loop closure. Even a single wrong loop closure can make the whole SLAM system brittle. The number of landmarks, that can be added to the matrix is also limited to achieve real-time operation.

Multi-modal distribution could be modelled using a mixture of Gaussians or using particles in a particle filter approach. The particle filter-based approach is more popular among the two and a modified version called Rao-Blackwellized particle filter [15] is implemented in the Gmapping algorithm [16]. In this approach, particles are used to represent a robot pose and each particle has an independent hypothesis of the map. In the Rao-Blackwellization process, the joint posterior is factorized into two components.

$$P\left(\mathbf{X}_{0:k}, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right) = P\left(\mathbf{m} \mid \mathbf{X}_{0:k}, \mathbf{Z}_{0:k}\right) P\left(\mathbf{X}_{0:k} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0\right).$$

where the second factor consists of estimating the trajectory so far $X_{0:k}$ and using that trajectory estimate to map factor m. The posterior distribution of the second factor is computed using the particle filter approach where new particles are generated based on the sample importance weighting. The first factor, which is map updation is performed based on the estimated robot pose and the history of all the observations till this time. Since each particle has its own map, updating the map for a large number of

particles is computationally expensive. Also, in the resampling process, there are chances that the best-fit particle may get rejected. To solve this, an improved sampling strategy is chosen, which computes the posterior pose distribution or the proposal distribution(the second factor) based on the current observation of the map. For each particle, a Gaussian proposal is defined and used to sample a new pose for the current state. If the observation model fails, then only the motion model is used. Further, the resampling is not performed in every iteration and an adaptive strategy is implemented. A dispersion value for the importance weights of the particles is computed which indicates the variance in the posterior distribution. If it is above a limit, then the convergence has not yet happened and new particles are resampled. If it is below a limit, it indicates the particles are clustered together in closer regions and hence resampling is avoided. One of the problems is its inefficiency in mapping larger environments as the map is continuously updated after each iteration. Also, loop closure is not explicitly present, but using a large number of particles, it will implicitly happen.

Feature-based mapping

In feature-based mapping, the high scanning frequency and accuracy of LIDAR sensors are exploited for matching consecutive scans or features for the creation of the map. Hector SLAM [17] is one such example. There is a map creation component and a navigation filter component. In the mapping part, the currently observed LIDAR scan is matched with the map created so far using Gauss-Newton-based non-linear optimization. The rigid transformation matrix from scan matching is estimated which can be used to compute the current robot pose. The parallelly running navigation filter estimates the 6DOF pose of the robot using the IMU, and gyroscope sensors through the Extended Kalman filter. The estimated 6DOF pose is projected into the 3DOF planar scene which is used for initialization in the non-linear optimization. Also, since the occupancy grid map is discrete, the gradients are computed using an approximated 2x2 box filter. To enable robust scan matching, the map is stored in multi-resolution scales of lower resolutions similar to image pyramids used in computer vision applications. The initial scan matching is performed on the lowest resolution one which is used to initialize the scan matching in the higher resolution maps. As compared to Gmapping, this approach is faster as it relies only on fast laser scans and only a single set of consistent maps is retained.

Graph-based mapping

In graph-based mapping, a factor graph is created with nodes for representing the robot pose and the landmarks and edges connecting between them. These edges are the constraints on this graph. There are edges between two robot pose nodes which indicate the odometry constraint and also the constraint between the robot pose and the landmark through a measurement constraint. These constraints are similar to the elasticity of a spring. The more certain the measurement is, the more stiffer the edge. That is, the nodes that are connected with that edge would act as anchors or are less movable. This indicates the confidence of those measurements in the optimization process. In the SLAM framework for graph SLAM, there is a front-end and a back-end. The front-end takes the current odometry measurement and the observation and adds it as constraints between nodes in the graph. This step can also be called

tracking. The back end, which is usually run on a separate thread, performs the expensive non-linear joint optimization on this graph which can be viewed as an error minimization problem or maximum likelihood estimate. This is the correction step. During this step, the initial robot pose node is fixed at the origin and other estimates can be computed with respect to this anchor node. There can also be a third thread running to check for loop closures. Data association is important for loop closure. When seeing a landmark, it should be distinct enough to be recognized as an already existing one in the graph or not. Because a wrong loop closure can propagate the error to other nodes within the graph, giving an erroneous estimate. Loop closures are important in this graph structure to connect the robot pose nodes that represent the same position and thereby a global optimization can be performed by propagating the error through the closed loop in the graph. Since the optimization process is expensive, two separate optimization strategies are employed. A small sliding window of defined length is used to optimize the latest N nodes in the graph which is helpful for local mapping. When a loop closure is detected, a pose-graph optimization is performed where only the poses are optimized and the landmarks will converge to their true position. As compared to filter and feature-based methods where the Markovian assumption is made, graph SLAM is a full SLAM where the entire history of measurement is used for optimizing the entire trajectory and the map. In EKF-based SLAM, the complexity increases(the size of the state covariance matrix increases) only when new landmarks are added. Whereas in graph SLAM, as the robot moves, new pose nodes are added to the graph which increases the complexity in a linear fashion.

Kartoslam [18] is a graph-based mapping technique that takes advantage of the sparse representation of the graph for an efficient optimization process using a method called Sparse Pose Adjustment (SPA). The non-linear optimization function used is the Levenberg-Marquardt algorithm. However, as the number of robot poses and the observation increases, the size of the matrix on which the optimization is to be performed is computationally expensive. But most of the matrix elements are 0 as most relational edges between poses and landmarks are local. So, a Cholesky decomposition solver is used to solve such a sparse representation and its efficiency is dependent on the number of loop closures. Also, the sparse matrix is represented in a memory-efficient way using compressed column storage (ccs) format where the row and column pointers of the non-zero values are saved along with the value in a tabular format. For the optimization process, the graph is represented as a linear system which is linearized at each pose node. The error value computed at the pose node, an error Jacobian at the point, information matrix(inverse of covariance matrix) are used to solve a linear equation that will yield a Δ correction value that could be added to the robot pose estimate. The observation measurements are performed using the scan matching method where the current scan is compared with the previous scan or a set of scans in the map for data association or for the creation of new landmarks.

2.1.3 2D Localization methods

The kidnapped robot problem is a famous challenge in robotics where the robot should localize itself within a map available in hand. The starting pose is unknown. This can also be called global localization. The time required to localize, the computational power required and the amount of motion required by the robot to gather enough evidence to localize are some of the properties that can be compared among

the algorithms. In the outdoor environment, the GPS can aid in identifying the most probable search area within the map. However, in the indoor environment, the robot has to rely on the features for localizing.

Filter-based localization

Particle filter localization [19] is the most commonly used Monte Carlo localization method as it can represent multi-modal distribution. A predefined number of particles having equal weights is generated in the initialization step and is uniformly distributed in the map we have in hand. A particle represents the current robot pose x, y and orientation. Once the robot moves in the real world, the same motion command is executed by the particles generated in the map along with a noise which accounts for the uncertainties. This is the prediction step. Now, the observations by the robot in the real world are compared with the observations in the map by the particles and the particles having consistent observations will get higher weights. This is the measurement step. To represent the posterior distribution, new particles are resampled based on the weights assigned in the measurement step and the particles with lower weights will eventually die out. The overall number of particles remains the same and after repeatedly performing the above steps, the particle will converge around a point which is the most likely location. The SLAM equation $p\left(s_t, m \mid z^t, u^t\right)$ changes to $p\left(s_t \mid z^t, u^t, m\right)$ for localization. The previous equation for SLAM where we jointly estimate the map and the robot pose is replaced by estimating the robot pose given the map.

In the Adaptive Monte-Carlo Localization [20], the particle depletion problem is solved to an extent by modifying the likelihood function of the observation. This is because, when performing a global localization, if the sensor has an accurate sensor model(high peak in the likelihood function), there can be chances where an incorrect observation may cause that particle to have a higher importance weight and the posterior distribution would be sampled densely around that particle. The rest of the environment would not be represented with enough particles thereby leading to a particle depletion problem where no particle has an accurate representation of the pose of the robot. To solve this, the likelihood function for observation needs to be smoothed. The likelihood function is represented as a Gaussian distribution. For each particle, a circle of uncertainty is formed using the K-nearest neighbour method where k==1. The radius of this circle is the governing factor for modifying the standard deviation of the likelihood Gaussian distribution. The bigger the circle indicates that the corresponding particle is representing a bigger state space and hence the smoothing of the likelihood function should also be higher. Further, a KL divergence is computed between the posterior distribution of unsmoothed and smoothed likelihood functions. If the divergence is greater, then the convergence has not been achieved and the number of particles is not reduced in the resampling process. As the particles converge to the ground truth position of the robot, the KL divergence value will decrease and less number of particles can be used for representing the state. Another strategy to reduce the overconfidence in the sensor is to select only 10 beams out of 360 laser beams for the observation model. The proposed approach is said to be covering at a faster rate even with a smaller number of initial particles.

Graph-based localization

AMCL is the most commonly used localization procedure in a 2D environment. However, the graph slam can be modified to perform localization only. For this, the graph that is used in the mapping stage should be available and the uncertainty in the robot poses and landmarks should be set to 0. If the graph is not available, create a graph from the map based on some keypoint robot locations where maximum unique non-redundant features of the environment can be captured as landmarks and set all the uncertainty to 0. That is, all the positions of robot poses and landmarks cannot move from their estimated value. The kidnapped robot after getting the first scan can be matched with the landmarks that are stored in the graph. This unknown pose of the robot is added as a new pose node with high uncertainty such that during optimization, the pose can be shifted to the most correct location anywhere within the map. As the robot moves and records more observations, more constraints are added and the robot can be better localized. One drawback is that it can represent only unimodal pose distribution. If there is an environment with similar rooms, a particle filter can represent multiple hypotheses using two sets of particles concentrated in each room, whereas, in the graph-based method, it would be biased to only one of the rooms. Even in the case where the environment is distinct, the data association should be good enough to match the viewed feature with a known landmark for the approach to work as expected.

2.2 3D SLAM

2D SLAM methods are most commonly used in AMR applications because of their ease of storage and perform localization using AMCL. However, the application is limited to robots moving on a planar surface which may not be the case in real-world environments which may even have stairs and slanting surfaces and the assumption of z=0 fails. Also, the features are from the observations made at a predefined height at which the 2D LIDAR is located which may not provide distinct landmarks. This led to the requirement of having 3D maps for better representation and explainability of the features or landmarks.

In 3D SLAM, as compared to 2D SLAM, the dimension of the map increases from 2 to 3. Also, the pose of the robot changes from 3DOF to 6DOF. As a result, the computational power required is higher which demands better strategies to track between frames in realtime requirements. The most commonly used sensors are 3D LIDAR, monocular and depth cameras. In this project, we are only focusing on camera-based SLAM methods called visual SLAM. The types of maps are not separately mentioned in a subsection as each SLAM method produces different types of maps. However, in broad aspects, they can be divided into two such as dense mapping and sparse mapping. In dense mapping, the exact representation of the environment is mapped using point cloud or mesh representation which requires a very high memory requirement to store the map. In sparse mapping, some essential features(points, lines, planes, objects) are extracted from the environment which is tracked and stored as a map. It can be stored with limited memory capabilities. In both mapping methods, the semantics can be added on top of it. In dense mapping, it can be in the form of assigning each pixel in the point cloud with a class label. In sparse mapping, the keypoints can be assigned a label and objects can be represented using coarse object models such as cuboids and ellipsoids with a label assigned to them. Further, the SLAM methods are divided into direct and feature-based methods. In the direct method, the tracking is performed between image

frames based on aligning the entire image space through methods like ICP whereas in the feature-based method, only features(SIFT, SURF, ORB etc.) in the current frame are tracked between multiple frames.

2.2.1 3D Mapping methods

The following subsections briefly explain some of the relevant dense and sparse mapping strategies in the order of the year in which the work was published. ORB-SLAM [5] which is a sparse feature-based mapping method is not explained in this section as it will be explained in detail in a later chapter4.2.1.

MonoSLAM

MonoSLAM [21] was one of the first real-time online 3D SLAM approaches using only RGB images from a monocular camera. MonoSLAM can be categorised into a sparse mapping method and a feature-based mapping method as features are detected and tracked over multiple frames to create a sparse map with only features. The underlying framework is the Extended Kalman Filter(EKF). To achieve a runtime performance of 30hz, the number of features that were retained in the covariance matrix is 100. So the features that need to be retained should have good robustness. This also limited the usage of the SLAM to a smaller environment with limited features. The state vector in EKF consists of the 3D position of the camera, the quaternion representation and the linear and angular velocity components. Since a monocular camera is used, an actual depth estimate is not available and as a result, an object of known size is kept in front of the camera when the SLAM is initialized so as to fix the unknown scale. The features used are 11x11 image patches and the world is assumed to be static. The 3d position of the feature is to be estimated which includes the depth. A line is drawn from the camera position at which the feature is first observed to the feature and a uniform distribution is assumed for the depth of the feature along the line. This is called partial initialization of the feature. Even though the motion model of the camera is unknown, a constant velocity model is assumed and the most probable pose of the next frame is predicted. The partially initialized feature is warped and projected onto the next frames and a template-matching search is performed. To reduce the computational complexity, active search is employed where the search is performed in the region proposed by the uncertainty ellipse of the feature. The line is also reprojected and the depth uncertainty peaks from a uniform distribution to a Gaussian distribution. After which the feature is said to be fully initialized. In the map visualization, this could be seen as an uncertainty ellipsoid being compressed into a small sphere or point with very small uncertainty. Loop closure is present as the template matching is performed. Within the generated map, pruning of less reliable features is performed where features which are not visible in at least 50% of the frames in which it is predicted to be visible are removed. The experiment in the Augmented Reality(AR) application is performed which indicates the creation of stable features which were used as anchors to attach virtual furniture to the scene. Since a constant velocity model is adopted, the tracking may get lost if a sudden jerk happens.

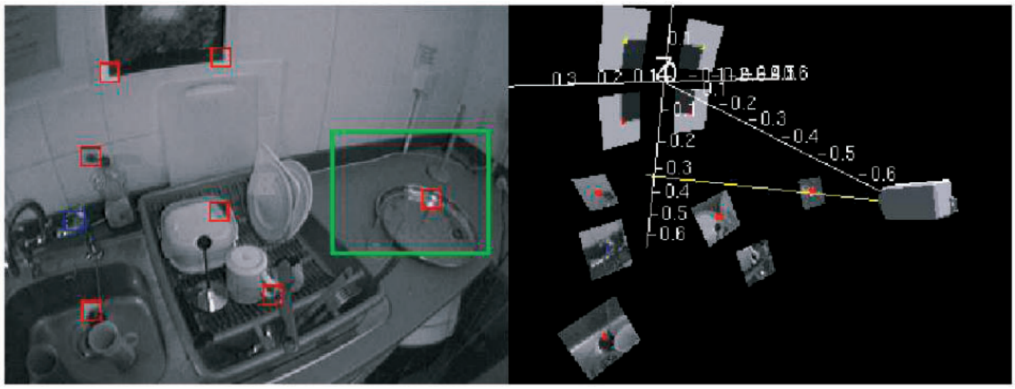

Figure 2.8: MonoSLAM [21]

PTAM

PTAM [22] stands for Parallel Tracking And Mapping which is the first method that separates the tracking front-end and the mapping back-end into two separate threads such that the expensive batch optimization in the mapping part won't block the real-time tracking process. In MonoSLAM, without map updation, it couldn't process the next frame. PTAM can be classified as a sparse mapping method and a feature-based mapping method where a much denser representation of features is possible as the EKF backend is replaced by bundle adjustment. This is the first popular algorithm to use non-linear optimization. The features that are extracted are the FAST [23] corner features. As in MonoSLAM, along with the pruning of the map features, the keyframes that have redundant information are also pruned in PTAM thereby reducing the computational complexity in the bundle adjustment process. There is local and global bundle adjustment. In the local one, the latest K keyframes are taken for real-time tracking and a bundle adjustment is performed on them to estimate the relative pose of the latest frame. In local bundle adjustment, the current frame and the past 4 keyframes, the map points that are visible in these keyframes and all the keyframes that view these mappoints are used. The error that is being reduced is the projection error. During the tracking step, a motion model is used to predict the pose of the current frame, and the feature points in the map are projected into the frame to refine the predicted pose. The feature point projection is done in two stages where initially only 50 points are used and after an initial estimate is obtained, 1000 points are used for refinement. The map is initialized using the five-point algorithm and map points are created via correspondence search and triangulation. Further, global bundle adjustment is performed on the whole map and once convergence is achieved, further data association is performed where the guided correspondence search for unassociated points is performed. Some of the problems that were faced are the trouble in extracting FAST features on featureless surfaces, losing track of the camera when an abrupt motion occurs, not being able to perform big loop closures and the errors in the map initialization which may propagate to further keyframes. Also, the points that are occluded in the real world at certain viewpoints are visible in the map created at any viewpoint because the occluded objects are not mapped.

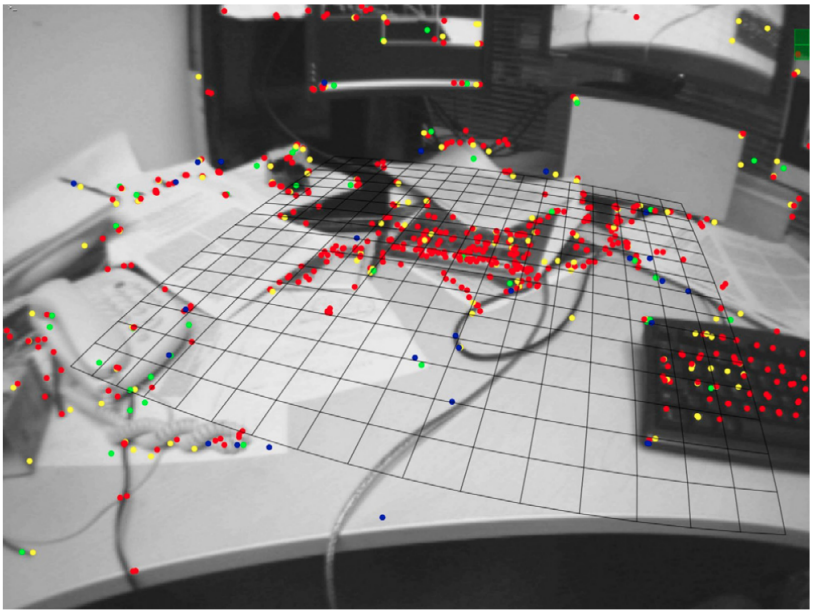

Figure 2.9: PTAM [22]

DTAM

DTAM [24] stands for Dense Tracking and Mapping which can be grouped as a direct mapping method where the map is of dense type. It utilizes every pixel of the RGB image to perform the alignment between consecutive overlapping frames. So, during the mapping stage, the motion is restricted to have trajectories that ensure overlapping RGB images. It tries to create an inverse depth map by minimising the photometric error. Since it is operating on pixel intensities, this algorithm could be used in a textureless environment where feature-based methods fail to extract features. To arrive at the least cost non-convex solution, it needs to perform iterative gradient-based operations which can be accelerated using a GPU. The Huber norm is used instead of the L1 norm to form smooth regularisation on edges where discontinuities are present. Once the smooth inverse depth map is estimated, 3D meshes can be extracted. The occluded unmapped regions can be predicted using the inverse map representation. During the localization step, a virtual camera is created over the created map and obtains the inverse depth image and the RGB image which is used to compare with the actual viewed RGB image by the robot for pose estimation. For robustness in the tracking, only the pixels having a photometric error below a certain threshold are selected. The AR experiments showed that PTAM required relocalization when the camera moved abruptly whereas DTAM was stable and didn't lose track of the camera pose. One problem existing in DTAM is that the assumption of the world is static and the illumination of the scene doesn't change during the mapping stage. Another challenge is the requirement of higher computational resources like GPU for the inverse depth map estimation.

Figure 2.10: a.) Inverse depth map reconstruction b.) DTAM colored output [24]

KinectFusion

KinectFusion [25] is a dense volumetric mapping method that utilises a Kinect RGB-D sensor. This algorithm can be grouped into dense mapping and direct method of mapping. The mapping is represented using a truncated version of the SDF (Signed Distance Field) where the empty space is represented by positive values, the surface by 0 and the occupied region by negative values. A line is drawn from the camera centre to the pixel location and the SDF values are truncated around a region. Through viewing the same surface from multiple frames, the TSDF values are fused. This TSDF method of dense volumetric representation was popularised after KinectFusion. Having a dense map also means that the memory required to store it is really high. The usage of the Kinect RGBD camera offered a better advantage over DTAM to operate in any light condition as the depth is estimated from the IR beam. The SLAM algorithm consists of 4 steps. From the input image, surface normal maps are generated which can be used to initialize the map if it is the first iteration. The next step is to estimate the pose of the camera. This is done by performing a frame-to-map ICP matching method at different scales where the current depth image and the depth map are aligned to estimate the camera 6DOF pose. The ICP can be initialized with frame-to-frame depth image alignment between consecutive frames. However, this may fail in cases where the dominant part of the scene is covered by a planar surface. The third step is to update the map with the new TSDF information from the aligned camera pose. The final step is to estimate the latest surfaces in the map being created. The experiments in AR showed that the algorithm was able to detect surfaces and perform at 30 Hz. This real-time performance requires the availability of the GPU. Kinectfusion proves to be good for object reconstruction as each object is well-distinctly mapped and can be extracted. A challenge for the Kinect camera is the scattering in the case of reflective surfaces.

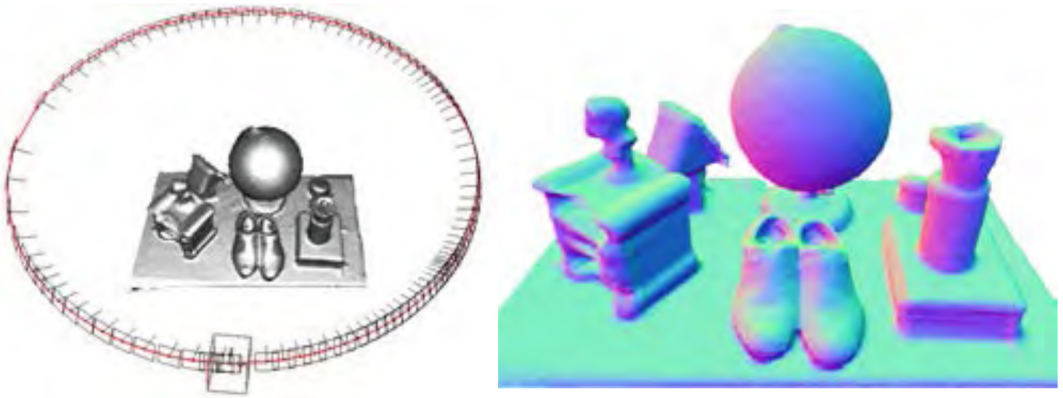

Figure 2.11: KinectFusion [24]

SLAM++

SLAM++ [26] is a first-of-kind semantic SLAM algorithm where an indoor environment enriched with semantic information of known objects can be used to aid in mapping and relocalization. This SLAM algorithm can be classified into direct and dense semantic mapping methods. The inputs to the SLAM framework are the RGBD images and prior 3D mesh models of objects. In this SLAM, there is an initial training phase, where the 3D mesh model of the object is created using KinectFusion and point-pair features(PPF) are created for each object which acts as descriptors for object detection and the object 6DOF pose estimation. These are the critical points on the object that can best define the 6DOF pose of the object. The objects that are used for training are the most commonly appearing objects in a scene and all the object-related properties are stored in a database. The backend of SLAM++ is based on graph optimization where the vertexes of the graph are represented by the camera and object poses(along with their labels). The constraints between the camera poses are estimated using ICP alignment between the current depth frame and the object-enriched map. That is, when a part of the object is viewed in an image frame, it is used to predict the 6DOF of the object which can be used to place the dense mesh model of the object into the scene and can be used to predict the next scene view. In KinectFusion, the ICP alignment is performed on the incomplete depth map. For object detection, the descriptor searching and matching method is performed in a parallel manner using the GPU and this limits the number of objects to be searched to be 5 to achieve real-time requirements. Also, in object 6DOF pose estimation, a voting mechanism is incorporated to fuse the prediction from multiple views. These objects can also aid in relocalization by forming a new short-term graph with vertexes as object positions and a normal vector in the direction of the x-axis. As more and more objects are added to this short-term graph, it could be compared with the long-term graph for camera pose estimation. This process also helps in loop closure. Another assumption is made where all objects are assumed to be on the dominant planar surface(floor) and this constraint is added as a weak unary prior for object nodes in the graph.

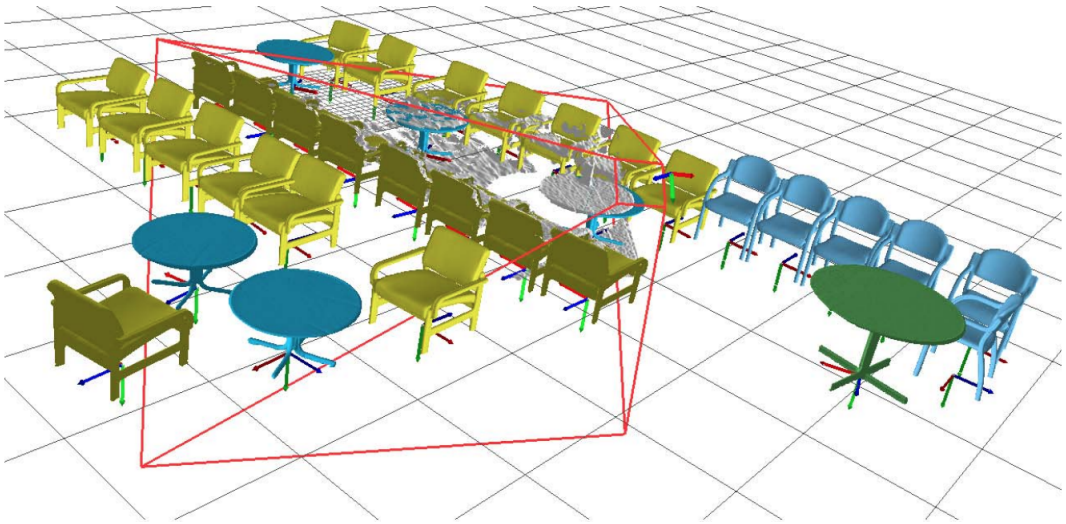

Figure 2.12: SLAM++ [26]

LSD-SLAM

LSD SLAM [27] stands for Large-Scale Direct Monocular SLAM which is a direct mapping method that produces semi-dense maps using monocular RGB images. The underlying framework is a pose graph. Each node represents a keyframe which stores the image, inverse depth map and its variance. The nodes are connected via edges which indicates the similarity transformation between the keyframes. Similarity transformation is used as a monocular camera has a scale drift problem. The error that is being reduced is the photometric intensity error. There are three main stages in this framework. Initially, we need to have a keyframe fixed that can generate an inverse depth image which is used to initialise the map with high variance. The first stage is tracking where the rigid body transformation between the current frame and the latest keyframe is performed with the previous frame pose used as initialization. The second stage is the depth map estimation where the current frame is either selected as a keyframe or not. It is selected as a keyframe if there is a significant viewpoint change and then the depth map of the previous keyframe is projected into the current keyframe and is set as the reference keyframe. If not selected, then the frame is just used to refine the probabilistic semi-dense map being represented. It is a semi-dense map, as the inverse depth map is estimated around the regions where there are high gradients such as edges and corners. Also, the noise of each depth pixel is not uniform as in previous direct methods and rather varies depending on the number of times the point is viewed which leads to lesser noise with more viewtime. The final stage is map optimization where the pose graph optimization and the loop closure are detected and performed. The highlight of this approach is its ability to run on a CPU. A disadvantage is that it cannot recover from the abrupt motions of the camera.

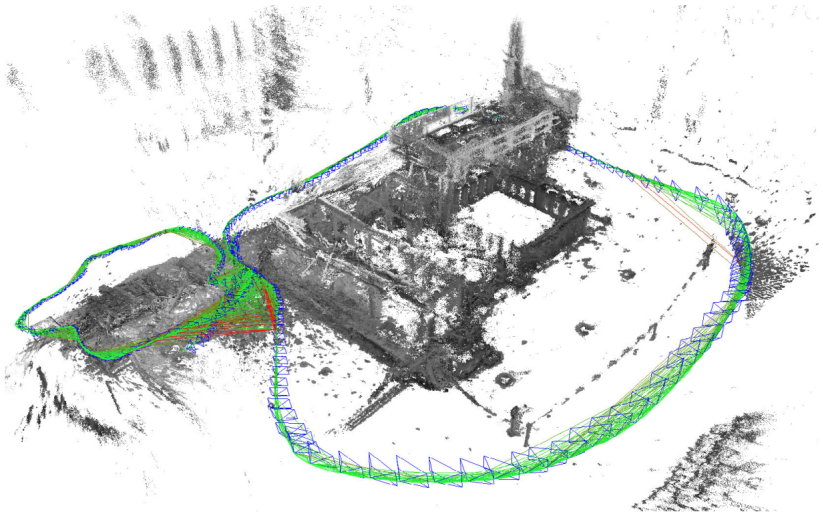

Figure 2.13: LSD SLAM [27]

RGB-D mapping

RGB-D Mapping [28] utilized the RGBD images to perform a feature-based dense map creation. The underlying backend is a pose graph whose vertex consists of the pose of the camera connected by the motion constraints. This motion constraint is obtained from the modified form of ICP called RGBD-ICP. This is due to the fact that the noisy depth measurements from the RGBD cameras are not reliable beyond a certain distance and the normal ICP method may fail to get an accurate rigid transformation between frames. Whereas the RGB image provides better visual features which are represented by SIFT features. The 3D position of the features is extracted from RGB and depth information. The visual features between 2 consecutive frames are used to initialize the ICP. Further, data association is performed using RANSAC and a distance metric in terms of visual features and point features(from depth data) is defined which is minimized using Levenberg-Marquardt non-linear optimization. The pose graph is then optimised using TORO(Tree-based netwORk Optimizer)ref which returns the maximum likelihood camera pose estimates. Loop closure is also performed on a separate thread where the ideal candidate frames are selected based on their closeness and the SIFT feature-based inlier count is estimated using RANSAC. For a frame to be a keyframe, it is selected only if the number of overlapping visual features with the latest keyframe should be below a threshold. The map is represented using a surfel map. A point cloud may contain a lot of points and it is found that creating surfels provides more efficient and accurate computation than downsampling the point cloud. A surfel represents a small patch of the point cloud with information such as 3D position, confidence value, the surface normal and colour. These parameters are updated as they are viewed from different viewpoints. Surfels below a certain confidence value are removed. The colour of the surfel is estimated best when the normal from the surfel best aligns with the camera centre viewing it. Even though the map created is accurate, this method was not able to perform in real-time as it wasn't using any GPU.

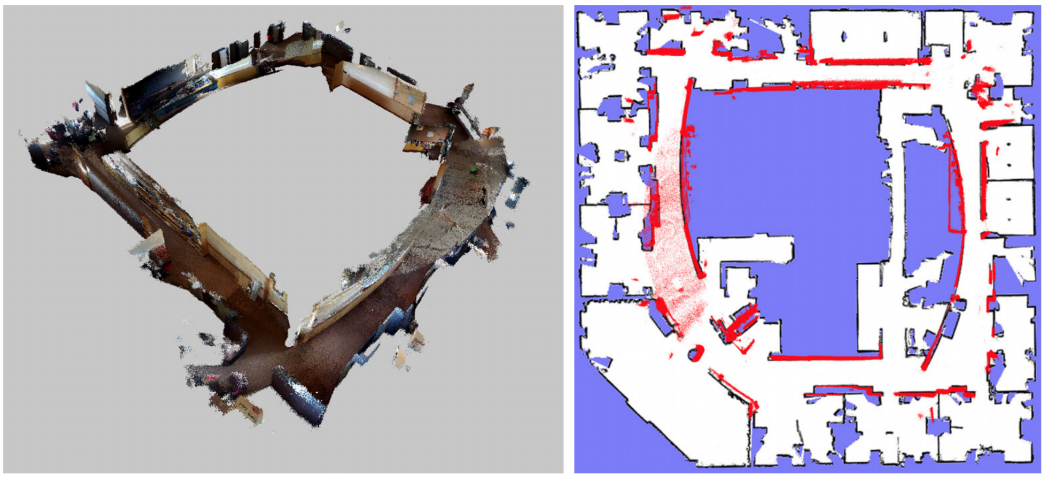

Figure 2.14: RGB-D Mapping [27]

ElasticFusion

ElasticFusion [29] is a direct dense mapping method that is able to perform in real-time. The direct methods are usually slower as they process the entire pixels which is solved here by the use of surfel representation of maps. Tracking is performed using a joint optimisation performed on both photometric(RGB) error and geometric(depth) error. This is also called frame-to-model tracking where the color information of the model can also be used to represent an error term. There is no usage of a pose graph. In the surfel representation, each surfel also encodes the timestamp of initialisation and the timestamp at which it got updated latest. Based on the latest updation timestamp, they can be divided into active and inactive parts of the map. If the timestamp is above a threshold, it could classified as an inactive region. Only the active part of the map is used for pose estimation of the camera. The RGB and the depth map are projected into the current frame for error estimation which is minimized using Gauss-Newton non-linear optimization at three different scales on an image pyramid. Further, the map is updated using a deformable graph which performs non-rigid transformation based on L1 norm distance. Each vertex in the deformation graph represents a surfel region and its relation with 4 other nearby vertexes. This allows us to perform a relative transformation of different regions of the surfel map without causing tearing of the map where updating one part of the map causes it to deviate from a global map. Deformable graphs are map-sticking methods. These are called local loop closures and the surface constraints can be passed to the inactive region for further modifications. The surfel parameters are updated as it is viewed multiple times. Also, in ElasticFusion, the position of the light sources in the room can be detected by voxelising the reflected rays and the intersection area is identified. This can help in solving the lighting conditions problems that affect the direct mapping methods. This SLAM requires the usage of GPU.

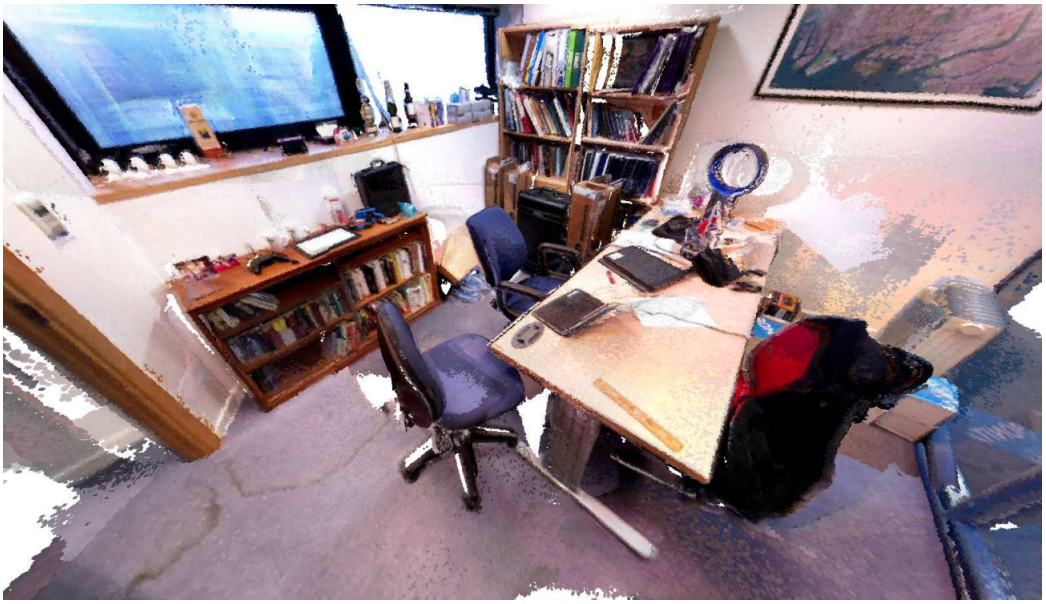

Figure 2.15: ElasticFusion [29]

SemanticFusion

SemanticFusion [30] is a semantic SLAM framework that is built on top of ElasticFusion with a class label for each pixel. It can be classified as a dense semantic map created using direct mapping. A Convolutional Neural Network having VGG architecture is modified to produce per-pixel label probabilities. As compared to SLAM++ where semantics only relate to the objects that are available in the database, in SemanticFusion the semantics of walls, and floors can also be added. The network is trained with RGBD images instead of RGB images. This type of semantic segmentation proved to be more accurate than a single frame segmentation as the probabilities of labels obtained at different viewpoints can be fused. These label probabilities are stored for each surfel element. So when a surfel element is initialised, it has a uniform probability for all the possible classes and later, through Bayesian fusion, these probabilities are updated to peak for a certain class label. When an RGBD image is provided to the CNN, the per-pixel label is generated and is projected into the surfel map with the help of the camera projection matrix which is obtained via the tracking section of the ElasticFusion which provides stable correspondences. A map regularisation is also performed to smooth the region of the map where there is noisy class label prediction as compared to its neighbouring surfels. For this, the fused label probability of a surfel and the labels of the closeby surfels along with their normals are used for regularisation. Surfels having similar normals are most likely to represent the same object. As in the case of ElasticFusion, GPU is needed and the SLAM was able to achieve a real-time performance of 25 Hz.

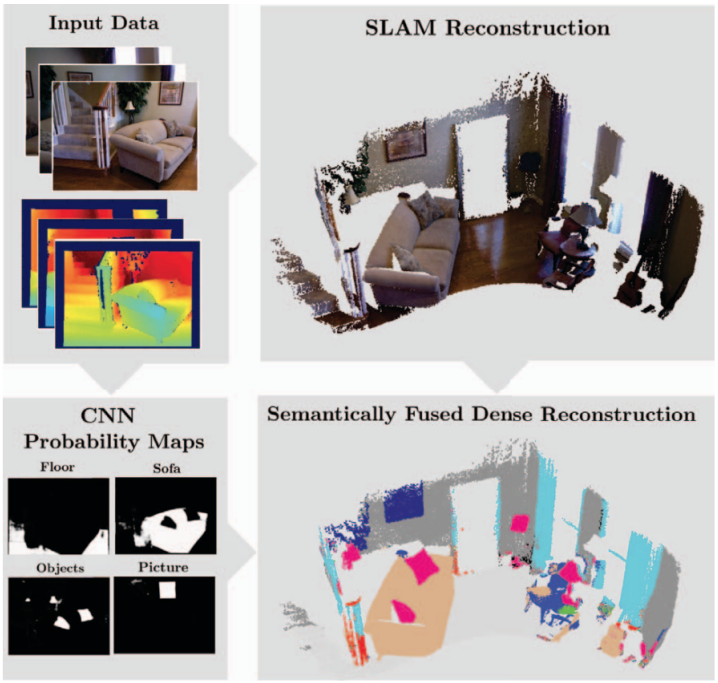

Figure 2.16: SemanticFusion [30]

RTAB mapping

Real-Time Appearance-Based mapping [31] is a complete SLAM solution which is popular in the ROS community because of its ability to customize all the framework properties such as input sensor type to output map type and ability to handle memory management. It can be classified as a feature-based dense mapping method. The SLAM framework is able to process multi-modal sensor data. It requires odometry from the robot and since it can be fed into the system externally, we can define from where this odometry is extracted. Proprioceptive sensors such as wheel odometers or IMUs can be used. Further, the input sensor can be 2D or 3D LIDAR or RGBD or stereo images. Odometry is important in RTAB mapping because they mention that the visual methods can fail to extract features when viewing planar textureless surfaces where these proprioceptive sensors can provide valuable information. In other words, visual-inertial odometry is more robust than visual odometry. At the input of the framework, there is a synchronization module which synchronizes data from different sensors based on the least time delay approach. The backend of the SLAM is a graph optimization. This framework is able to handle multisession mapping where multiple independent areas can be mapped and when a loop closure is detected, can be aligned together globally. If multisession mapping is performed, then Tree-based netwORk Optimizer(TORO) is used for non-linear optimization and in the normal single mapping method, g2o or gtsam can be used. The loop closure is detected using a visual bag of words approach which can add a constraint between 2 nodes on the graph and the error can be propagated. Loop closure should be performed carefully as a wrong loop closure can make the entire system fail. So a threshold is defined which compares the loop

closure rigid transformation and if it is above a certain limit, then the loop closure is rejected. Another parameter decides how fast should the new nodes be created. If the robot is moving fast, we should set it to high such that the consecutive frames have some overlapping for visual feature matching. If the robot is moving slowly, then it should be kept low such that redundant information is not added to the graph which consumes memory. A frame is said to be a keyframe if the number of inliers in the current frame compared to the current keyframe is less than a threshold. There are two types of memory in RTAB mapping. The short-term memory(STM) is responsible for estimating the relative transformation of the current frame with respect to the previous frame. This can be performed either using frame-to-frame matching or frame-to-map matching. In frame-to-frame matching, the visual features are extracted(which we can customize), a descriptor can be defined for matching, and then the estimated transformation can be used to predict the motion which can be validated using RANSAC inlier counting. Also, STM extracts the visual features that are added to the visual bag of words for loop closure detection. Further, a local map is created using the downsampled pointcloud of the input depth. There is another memory called long-term memory(LTM). To reduce the size of the graph matrix to be optimized, important nodes are moved to LTM. This is based on two factors such as the importance weight of the node and the view time period of the node. That is, if the node is visited multiple times, then we have robust features with respect to that node and it has a higher importance weight. Also if a node is not visited for a long time, then it is moved to LTM. There is another customizable parameter which decides how many nodes should be kept in the short-term memory. After a node is added to the graph, a local bundle adjustment is performed using the part of the graph with only nearby nodes that are viewing the same features. During global bundle adjustment, the nodes in the LTM that are present in the loop area are recovered for accurate optimization. The experiments showed that stereo camera images have better accuracy for depth mapping. However, in textureless scenes, the RGBD camera performed better. This SLAM is able to be run in real-time using a CPU.

Figure 2.17: RTAB Mapping ref

CubeSLAM

CubeSLAM [1] is a feature-based sparse mapping method where the objects are represented as cuboids from monocular RGB images. It uses the 2D bounding boxes of objects obtained from the YOLO object detector and utilizes the vanishing point property of perspective camera images to sample the cuboid corners on the 2D box. This cuboid is optimized through multiview bundle adjustment. This type of object-centric SLAM has a graph-based backend where the objects can also add constraints in optimization. Previously, the object prediction and the SLAM were decoupled problems. Here, they show that object detection and the camera pose can benefit from each other. In previous approaches like SLAM++ only the objects whose 3D model is available in the database could be used in the SLAM whereas here, the 3D coarse representation of cuboid can be created for any object that is detected by the YOLO model. A cuboid can be represented by 9 parameters where 3 each for centroid, orientation and size. A single 2D bounding box, 3 vanishing points and a single corner on one of the edges of the bounding box can be used to project and identify 7 other corners of the cuboid in 2D. Now, the PnP algorithm can be used with 4 adjacent corners to project the cuboid to a 3D world and estimate its 9 parameters. Also, if the object is assumed to be on the ground plane, the 4 bottom corners can be projected onto the 3D ground plane and the vertical corresponding corners can be estimated. These 3 vanishing points are unknown and have to be sampled and different proposals are generated. For sampling the vanishing points, the edges of the object within the 2D bounding box are detected using a classical computer vision algorithm(canny edge detection) to intersect them and the cuboidal proposal is made to align with the slope of those edges. For each cuboidal proposal, a cost function is defined for the distance, angle alignment and shape errors which are to be minimized. The framework is built on top of ORB SLAM(with loop closure turned off) [5] where the objects are also added to the graph. There are three constraints in the graph which are camera-to-point, camera-to-object and object-to-point. In the camera-to-object constraint, the error modelled is the overlapping between the reprojected 2D box of the 3D cuboid onto the current RGB image and the current 2D bounding box for the object. For the camera-to-point constraint, the error is the reprojection error as in ORB SLAM. For object-to-point, the error is the 3D distance between the point that belongs to an object to the 3D cuboid representation of the object as the points belonging to an object should lie within it. Data association is performed using the descriptors defined for the feature points belonging to an object. A dynamic SLAM is also defined by the authors if the scene includes dynamic objects which can also aid in camera pose estimation instead of treating them as outliers along with the dynamic object trajectory and velocity estimation. The graph is modified to separate points into static and dynamic points where dynamic points represent the feature points belonging to objects. As the object is assumed to be rigid, the error for these dynamic points can still be represented using reprojection error. Another motion constraint is added for the dynamic object which is assumed to move between the frames. Data association is performed using the optical flow of the dynamic points and associating them with the objects. This algorithm is able to run in real-time at 30 Hz for static scenes and 10 Hz for dynamic outdoor scenes.

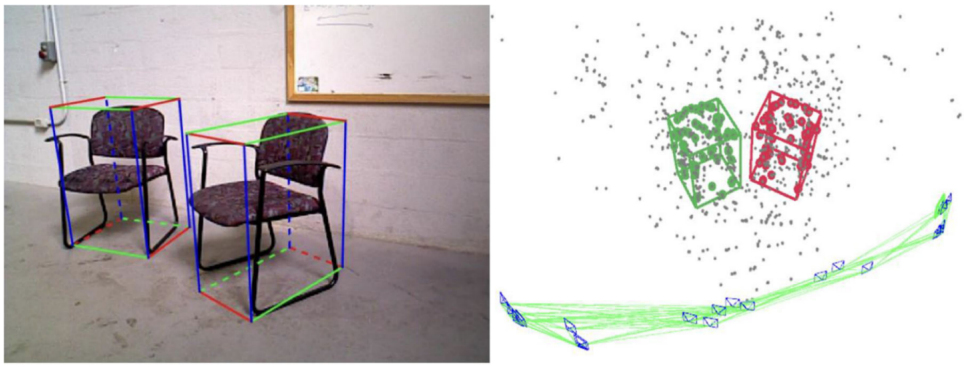

Figure 2.18: CubeSLAM [1]

Kimera

Kimera [32] is a full SLAM system that utilizes both feature-based and direct methods to generate dense mesh maps. It is a complete package including 4 components such as Kimera-VIO(Visual-Inertial Odometry), Kimera-RPGO(Robust Pose Graph Optimization), Kimera-Mesher and Kimera-Semantics. The input to the system is a stereo RGB image pair and IMU sensor data. There are 4 threads running in parallel. The first thread runs the frontend of Kimera-VIO. It takes in the current stereo images and the Shi-Tomasi corner features are detected in each image and the correspondence between the left and right images is identified. These features can be used for tracking. Further, the IMU sensor data is also taken as input and the GTSAM-based factor graph is created. The second thread runs the Kimera-VIO backend and the Kimera-Mesher. Kimera-VIO backend optimizes and publishes the latest state estimate. In the factor graph, the outlier points and the states that don't belong to the current smoothing region are removed for faster and more robust optimization. The Kimera-Mesher generates a per-frame mesh and a multi-frame mesh based on the state estimate from the Kimera VIO. The per-frame mesh is generated by projecting the RGBD feature using the estimated current camera pose. Multi-frame mesh fuses the per frame meshes till this time instant. The mesh is represented using connected vertexes which represent 3D features. New vertexes are added to the multi-frame mesh using the unmapped features from the per-frame mesh. Kimera-Mesher focuses on the generation of geometric maps for obstacle avoidance while navigating. The thread 1 and 2 are operating at faster rates followed by slower threads 3 and 4. Thread 3 performs the Kimera-RPGO. Loop closure is detected using the bag-of-words approach. Since factor graphs are sensitive towards invalid loop closures, loop closures are accepted if it is within an allowable region with a good amount of inliers via RANSAC. Also, irrelevant states are removed from the pose graph. These operations make the pose graph optimization robust and fast. Thread 4 runs the slowest global dense semantic map creation with the Kimera-Semantics. The dense map is created from the depth input using TSDF representation. Per-pixel semantic segmentation is performed on the RGB images and the label probabilities are raycasted onto the dense map. Bayesian update is also performed to fuse the label probabilities over time. The system is able to run with a CPU at 10 Hz.

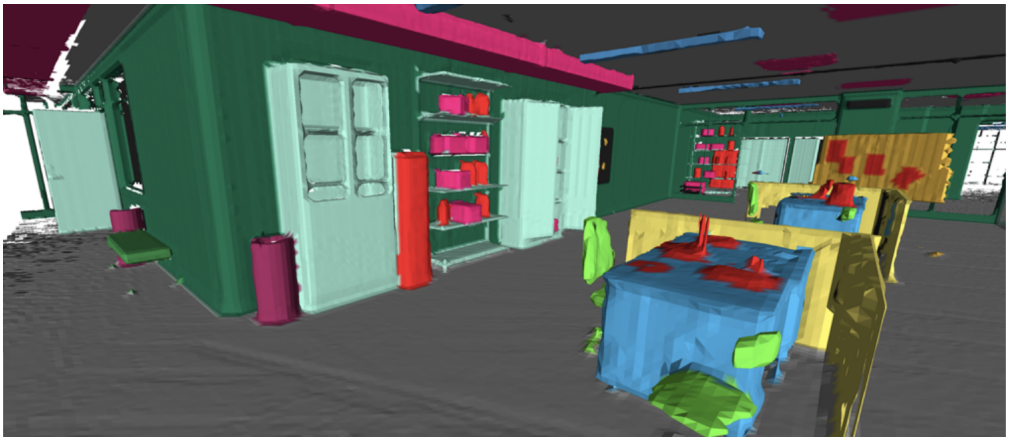

Figure 2.19: Kimera [32]

2.2.2 3D Localization methods

The global relocalization within the map being generated is a topic that is less discussed. While direct dense methods focused on creating an accurate representation of the map, relocalization within it became a tedious task as the features that are to be compared are quite high. Whereas the sparse mapping methods were able to identify the correspondences using feature descriptors. The localization methods can be broadly classified into two such as visual localization using images and feature-based localization. Only analytical methods are described here and recent deep learning-based approaches are not considered.

Feature-based localization

Feature-based localization is one of the most commonly used global localization methods in 3D SLAM. There can be two types of correspondence matching to perform the pose estimation. The first one is the 3D-3D pose estimation. In this approach, we have a known feature map with 3D points and a set of 3D feature points observed in the query image is provided. Given that the 3D points are obtained from the stereo or depth camera, a rigid transformation is to be estimated using ICP(Iterative Closest Point) after correspondence matching. Rigid transformation involves estimating the relative rotation and translation components between source and query 3D points. It is an iterative alignment process and singular value decomposition is used to estimate the rotation and translation component. If the images are from a monocular camera, then there would be a scale drift error and hence a similarity transformation is to be estimated which gives the relative pose of the query image with respect to the map. In similarity transformation, not only rotation and translation components but also a scale component is to be estimated. For this, both the 3D points have to be normalized to be made scale invariant and the ICP can be applied to find the rotation and translation components.

The second approach is the 2D-3D pose estimation. In this method, a 3D feature map is available and a 2D image with features on the image plane is provided. The correspondence matching can be done with

bag-of-words and the pose is to be estimated. The algorithm used is called PnP(Perspective-n-Point). In case the input image is RGBD, then the PnP algorithm can be directly performed as the depth is known. For monocular RGB images, triangulation needs to be performed as an initialization step in the PnP algorithm. P3P is one of the PnP algorithms where only 3 points are needed for pose estimation. It solves the problem as an ICP alignment problem assuming that the depth input of the 3 points is accurate. A 4th point can be used for verification based on the estimated pose.

In both these cases, after the pose estimation, RANSAC can be performed to check for inlier feature point count to verify whether the localization process is valid or not. Also, after pose estimation, the unmatched feature points or points for which correspondence has not been made can be associated based on a guided search method which can further refine the estimated pose.

Visual RGB-based localization

As compared to feature-based localization where the distinct points or pixels within the environment are used for pose estimation, in the RGB image-based approach, the entire image is used for pose estimation. The image-based approach tries to overcome the occlusion problem of keypoint features and the non-visibility of keypoint features in varying environmental conditions. Usually, the current image is compared against a set of template images in the database based on a brute force method.

An example method can be seen in RatSLAM [33] which is the SLAM method that is developed based on the hippocampus model of the rodents. There are 3 important components called a pose cell network, local view cells and experience map. A pose cell network is a 3-layer network of nodes with internode connections which is used to represent the current pose of the robot and its architecture is similar to a neural network. Assuming the planar motion, the robot pose is also approximated to be 3DOF such as x, y and theta. The local view cells are the component for place recognition which is our focus. It takes in the current RGB image of the scene. The image can also be a 360-degree panoramic image which can be cropped at 90 degrees each and joined together to form a single image. The image is then cropped by a horizontal line into two parts. As most of the distinct visual features are present in the upper part, the lower part containing mostly the ground plane is removed. Since the image has a higher resolution, it needs to be converted to a predefined lower-resolution template image using subsampling for efficient storage. Further, a global normalization is performed which normalizes the current image along with all the image templates that are already there in the database. This is to ensure that the entire set of images in the environment is taken under the same lighting conditions such that these templates would also be distinct in other lighting conditions. After which a local normalization on small patches in the current image is performed. This image is then converted to a grayscale image and stored in the database. As the robot moves and views more of the scenes in the environment, the new input image is compared with the template images in the database based on a similarity score called the sum of absolute differences. If the error is below a threshold, then a match is found. Otherwise, it is added as a new image template. The final component is the experience map which is a graph-like structure with nodes representing a local view cell, a pose from a pose cell network and the location of the robot with respect to a global map. The edges between the nodes represent the odometry and the time taken to travel between the nodes. An experience

map is needed to represent a global map because, in a pose cell network, the same cells can get activated when at positions in the real world that are far apart. Even though the area that can be represented by such a network is finite, the intercell connections as in a neural network help to represent an infinite area. A pose grid cell in the pose cell network and the activated local view cell together can define a unique location with respect to world coordinates. This SLAM was tested on a robot in a dynamic office environment that had been operating for a whole year and the navigation was able to be performed.

A problem with the image-based approach is the memory resource availability to store template images of a larger scene. However, in a given indoor environment this may not be an issue due to the finite defined space of the environment as compared to outdoor scenes. Now, newer deep learning approaches are able to extract high-level features rather than manually designed low-level features that can better explain the environment in the localization process. Knowledge of semantics also helps deep learning models extract robust structural features of the environment and discard dynamic objects for long-term localization. Illumination invariance helps to detect the scene as even though the appearance changes, the relative position of the features still remains the same.

2.3 Limitations of previous work

As seen in the first part, the 2D SLAM is limited to features close to the ground plane at the height of the 2D LIDAR. However, the motion of the robot may not always be planar and most of the distinctive features are visible in higher regions which indicates the need for 3D SLAM. In dense 3D mapping methods, the main challenge is the memory requirement to store large and dense maps. In robots, the memory to store such a dense map of gigabytes in size is not feasible. Also, the depth map creation and further mesh refinement from the images involves the requirement of GPU which may not be available in all the robots. This is why sparse mapping methods are preferred. EKF-based SLAM as in MonoSLAM [21] is not scalable beyond a certain region as the state matrix continues to grow with newer landmarks causing the system to be slower over time. Graph-based SLAM should be able to detect redundant information in terms of the camera pose nodes and the less robust feature nodes to maintain the overall graph faster to compute. Previously, as the robot remains stationary, newer pose nodes are continuously appended into the graph which doesn't provide any new information. In sparse mapping, performing the tracking and mapping sequentially limits the real-time operation thereby causing a lower FPS in performance. As in PTAM [22], separating the tracking and mapping on separate threads can prevent the slower mapping optimizations from blocking the tracking part. In sparse maps, the feature keypoints may not be visible at very different viewpoints and also when the lighting condition in the environment changes significantly. These feature points extraction are important to perform localization based on the bag-of-words approach. This highlights the requirement of having some higher-level object features that can be detected at any viewpoint and under any lighting conditions. Most of the above-mentioned SLAM algorithms lose tracking with abrupt motion of the camera and this is due to the lack of stable anchoring features. In algorithms like SLAM++ [26], a pertaining step is needed to create a database with the mesh shapes of the objects. And Kinectfusion [25] is to be performed to extract the accurate mesh of each object which involves a lot of manual work. Using pre-trained YOLO detection models on larger datasets and utilizing coarse

object models rather than exact object models can resolve this constraint. The semantics are added to the dense map through raycasting the per-pixel semantic segmentation performed on the images. Object classification models can generate a 2D bounding box at a faster rate which can be taken into advantage over slower semantic segmentation. A limitation in existing sparse maps is that there is no semantic information associated with each feature point. They represent some distinct geometrical features but have no idea of which type of object that feature point belongs to. In the above-mentioned algorithms, only CubeSLAM [1] was able to handle dynamic scenes whereas other mapping strategies assume a static world and discard dynamic features as outliers. In 3D localization, another limitation is the assumption of a unique solution to global relocalization. Multiple hypothesis is not retained parallelly as in the particle filter-based approach.

3

Methodology

3.1 Experimental setup

To compare the SLAM algorithms, we need a dataset with ground truth information of both the camera poses and the object poses. The popular dataset which is used in the evaluation of most SLAM algorithms is the TUM RGBD dataset [34]. It contains ground truth odometry, RGB image, depth image and accelerometer data(link). But the problem was, that there is no ground truth object pose information. This dataset is used by people to perform quantitative analysis of camera trajectory only and the map accuracy is compared through qualitative analysis. That is, the dense representation of the environment is created and the generated object models are placed on top of it to perform the manual evaluation of the overlapping. The YCB-Video dataset contains the ground truth information of objects within the scene along with ground truth camera pose, RGB image, depth image, and object bounding boxes and they are stored in BOP format (link). There were two drawbacks to this dataset. The first one was that there are no loop closures in these image datasets. If the scene is a table-top environment, then the camera covers only 180 degrees around the table which is almost half the scene only. In OA-SLAM, since it is operating with a monocular camera, the scale drift error may accumulate over time and loop closure is necessary to rectify the errors. The second problem is that there is a significant amount of noise in the RGB images. Since OA-SLAM is based on ORB-SLAM which relies on point features, the map initialization was not possible as it couldn't find robust keypoint features. To solve these drawbacks, a synthetic dataset was generated where we can customize a variety of properties such as the camera trajectory path where loop closure is ensured, background texture of the environment, the number and type of objects and the noise would also be absent. For this, the blender-based dataset generation tool developed by Sathwik [35] as a part of his RnD project is modified to generate the dataset in BOP format. Different use cases of the semantic object-oriented SLAM can be tried out here, for example, by generating cluttered scenes and checking for duplication or missing objects in the generated map. Since we are focusing mainly on the object mapping capability, in QuadricSLAM, we are providing the ground truth odometry into the SLAM system with sufficient noise and both in QuadicSLAM and OA-SLAM, we are using the ground truth labels and bounding boxes instead of YOLO object classifier.

Figure 3.1: Sample RGB image of the scene

The above image3.1 shows a sample image from one of the scenes in the synthetic dataset. 10 such scenes are created. In each scene, the number of objects placed is 5 and the objects used in each scene are randomized. The positions of the 5 objects are equidistant on a small circle that is centred around the origin and the radius of this circle is varied across each scene. The trajectory that is displaced by the camera is a circular one that is always pointing towards the region where the objects are placed. The height of the camera and the radius of the circular trajectory are also varied in each scene. Each scene consists of 1500 images where 1400 images belong to 1 full trajectory along the circle and an additional 100 more images are covered by the camera to ensure the loop closure event can be made by the SLAM. Each dataset generation took around 2.5 hours and the resolution of the images was 640x480 pixels. RGB images and depth images for the corresponding RGB images are generated which would be utilized by the QuadricSLAM [3] as it depends on the depth. This dataset is provided as input to both QuadricSLAM and OA-SLAM and the output map is generated.

To test the robustness of the SLAM frameworks towards the instability or uncertainty of the bounding boxes, the file containing the ground truth bounding boxes is corrupted with noise. This is done either by removing the bounding box for an object, changing the centre of the bounding box or by changing the width or height of the bounding box in each frame. Further, the SLAM operation was performed on this dataset.

Another experiment was designed to test the localization capability of the object-oriented maps. QuadricSLAM didn't have any inbuilt localization capability. So the experiments were performed only on the OA-SLAM. The experiment aims to prove the concept that having object representations along with feature point representations can aid in improved localization. Two mapping scenarios were defined. In the first scenario, one of the scenes from the synthetic dataset is selected and the entire images were used to create a full map of the scene. In the second scenario, half of the images from the scene are used to create a half map(half of the circular trajectory). Using the full map, the localization code was run in two modes namely localize with only points and localize with points & objects. Similarly, using the half map, both localization modes were tested. The test images consisted of both one image and two image sets which were selected from the unmapped regions of the half map. This is to prove that localization can

still be performed even if the current camera pose doesn't belong to the trajectory used for mapping.

3.2 Evaluation metrics

- Below mentioned are some of the evaluation metrics used to compare the **object errors**.
  - **Centroid error** - Measures the Euclidean distance between the actual object centroid and the estimated quadric centroid. For each scene, this could be individually calculated to identify which object has max and which has min deviation among all. Also, the average object centroid error of all the objects in the scene could be calculated so that it could be compared among multiple scenes. The estimated centroid can be derived from the 3 elements of the last column of the dual quadric representation matrix of size 4x4. [36]
  - **Rotation error** - Measured as the quaternion angular difference between the rotation matrixes of ground truth and estimated object pose represented in unit quaternion format. This can be done by performing dot product operation on these quaternions and then converting it to angular format in radians. The average rotation error for all the objects of a scene is also calculated.
  - **IoU of objects** - Computes how much of the ground truth object represented as a cuboid is overlapped by the estimated object represented as an ellipsoid [37]. We need to perform Monte Carlo sampling to randomly sample points(10,000) within the ground truth 3D cuboid and check how many of them lie within the estimated ellipsoid to get an idea of the overlapping percentage. Even though it may not give the exact value as an ellipsoid overlapping a cuboid perfectly has a higher volume, it can be used to compare the performance of different SLAM methods on the same scene. The overlapping volume should be as high as possible.

$$K = \frac{2V_i}{V_{gt} + V_{est}} \tag{3.1}$$

    In the above equation, V_i indicates the intersection volume and V_{gt} and V_{est} indicate the volume of the ground truth cuboid and the estimated ellipsoid respectively. When there is 100% overlap, the denominator is still higher as the ellipsoidal volume is higher.
  - **IoU of aligned objects** - In certain scenes, the IoU of objects in their actual pose may give 0% overlap but may lie in a very close vicinity. So we could align the estimated object pose to the ground truth pose and then compute the IoU. This metric can help to compare the volume of the ground truth object and the estimated object. The equation is the same as that above. The only difference is that the object is aligned before computing the intersection volume.
- Below mentioned are some of the evaluation metrics to test the **camera pose or trajectory errors**.
  - **Trajectory deviation error** - Measures the 3D position error(euclidean distance) of the estimated trajectory and the actual trajectory. A plot is plotted to show the accumulation/increase of trajectory error over each frame in the trajectory. The max and min deviation can

be obtained from the plot. Also, the root mean square error is computed using the below equation [36]:

$$\mathrm{RMSE}_{\mathrm{pos}} = \frac{1}{n}\sum_{i=1}^{n}\sqrt{\|\mathbf{x}_i^{x,y,z} - \hat{\mathbf{x}}_i^{x,y,z}\|^2} \tag{3.2}$$

- **Rotation error** - Measures the rotation angle error between each ground truth camera frame and the estimated camera frame for the entire trajectory. These errors over each frame are plotted over frame ID in a plot. Also, an average rotation error of the scene is computed from these errors.
- **Trajectory similarity measurements** - Given a set of 3D points of ground truth and estimated trajectories, we can check those trajectories' similarity using Procrustes Analysis, Fréchet Distance, and Chamfer Distance. Procrustes Analysis perform a similarity transformation on ground truth and estimated trajectory and outputs the dissimilarity value which should be low for the ideal case. Both Fréchet Distance and Chamfer Distance measure the similarity based on the closeness between the points on two sets of data. These distance metrics would be 0 in an ideal case.

- Below mentioned are some of the evaluation metrics to test the **object association errors**.
  - **Uninitiated objects error** - This error is measured in terms of integer numbers which indicate how many objects in the environment were not able to be mapped by the SLAM approach. These objects are detected by the bounding box but were not represented as object models in the map as they didn't satisfy some conditions of the respective SLAM algorithm.
  - **Fused objects error** - This error is also estimated in integers where it indicates the number of wrongly associated objects which causes two nearby objects to be identified as a single object. This occurs mainly due to the noises in the bounding boxes causing the bounding box of one object to overlap with the other object and thereby making the SLAM algorithm believe that both the bounding boxes originate from the same object.
  - **Duplication error** - Measured in integers where it counts how many copies of the objects in the real world are duplicated in the created map. This occurs when the bounding boxes are very noisy causing them to appear at two different locations over time and thereby the SLAM maps the same object as two different nearby objects in the map.

- Below mentioned are some of the metrics to compare the **performance** of different object-oriented SLAMs.
  - **Size of stored map** - To compare how much space is needed to store the map containing the objects.
  - **Computational power needed** - This can be measured based on CPU, GPU, and RAM usage system profile.

– **Computation time** - A measure to calculate the overall time from reading the data to generating the output map. This measure can also be used to estimate the FPS at which the SLAM algorithm performs.

Trajectory error is mentioned as trajectory quality and centroid & volume error are mentioned as landmark quality evaluation metrics in the QuadricSLAM paper [36]. Metrics such as centroid error, volume error, uninitiated objects and orientation error as match factor are mentioned in the master thesis by Kamil Kaminski [37].

4

Comprehensive study of selected algorithms

From the 3D semantic SLAM methods, we are selecting QuadricSLAM and OA-SLAM methods to look into the backend pipeline, understand the implementation details and identify the challenging sections within it. Also, the evaluation of such object-oriented SLAM methods is also performed.

4.1 QuadricSLAM

4.1.1 Summary

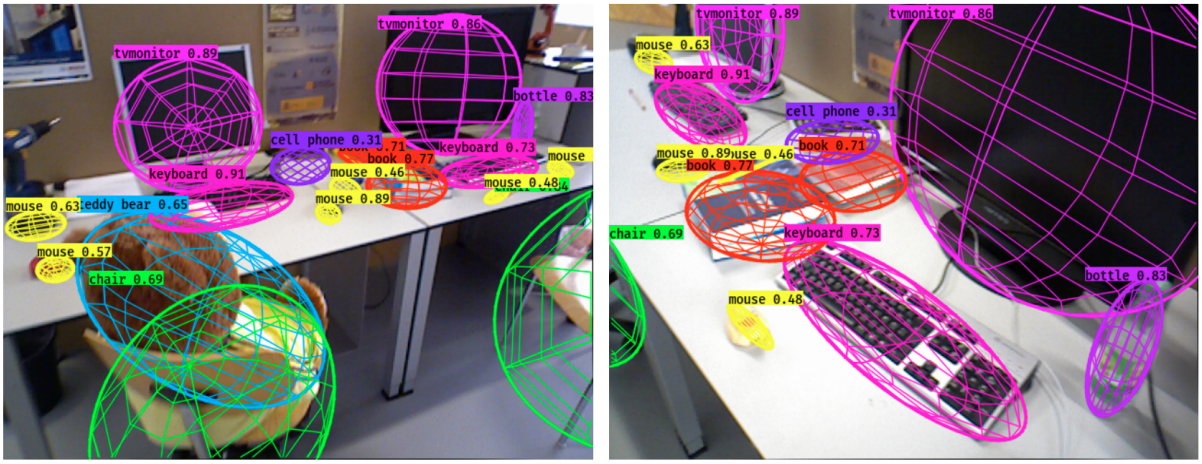

Figure 4.1: **QuadricSLAM**: Ellipsoids fitted on objects in image plane [3]

The Quadric SLAM is a semantic SLAM approach where each object within the environment is represented using dual quadrics whose parameters are estimated and optimised using multi-view geometry. The optimisation is based on the 2D bounding box generated for each object observed at different viewpoints. This object detection method also helps to add semantic labels to each object. SLAM involves performing mapping and robot pose estimation in a combined fashion. Here the robot camera poses and the quadric parameters are estimated using an underlying factor graph-based back-end. The odometry

errors in the factor graph could be reduced through the loop closure which is aided by re-observing the quadric representation of the object at different viewpoints. Thus both the odometry and the observation are aiding to minimize the errors of each other.

In the initial paper released by the authors [36], a monocular perspective camera was used to input RGB images to the front-end of the framework without any depth information. The quadric used in this application is ellipsoid because it is a closed surface. It can capture the position(3 parameters), radius or size(3 parameters) and orientation(3 parameters) of the object in the 3D world. This kind of representation can help to reduce the memory required to store the dense 3D representation of the surface of the object and at the same time serves all the basic purposes like object pose estimation, localization, semantic navigation, manipulation and planning tasks. The following section explains the mathematics behind dual quadric representation. The section is a summarized version of the content in section 3 of the Quadric SLAM paper [36].

For each view of an object, a 2D bounding box is generated with 4 corners x_1, x_2, x_3 and x_4. Each one can be represented using a 3D homogeneous vector. Further, the 4 edges of the bounding box l_1, l_2, l_3 and l_4 can be represented as a cross product of these corner points. $l_1 = x_1 X x_2, l_2 = x_2 X x_3, l_3 = x_3 X x_4$ and $l_4 = x_4 X x_1$.

The lines drawn from the camera centre to each of the four corner points can back-project the four edge lines into four 3D planes. The equation of the 3D plane can be written as $\pi_i = P^T l_i$ where i is the index of the 2D edge for which the 3D plane is generated. P is the camera projection matrix, P = K[R t] where K is the intrinsic parameter matrix of size 3x3 and R and t are the extrinsic parameter matrix of the camera of size 3x4. Therefore P is a matrix of size 3x4 and the resultant 3D plane π_i is a 4D homogeneous vector.

Thus a single view can create a constrained 3D space enveloped by four 3D planes. Since we don't have an enclosed 3D region from a single view, we may need multiple views to further constrain the object's shape and orientation.

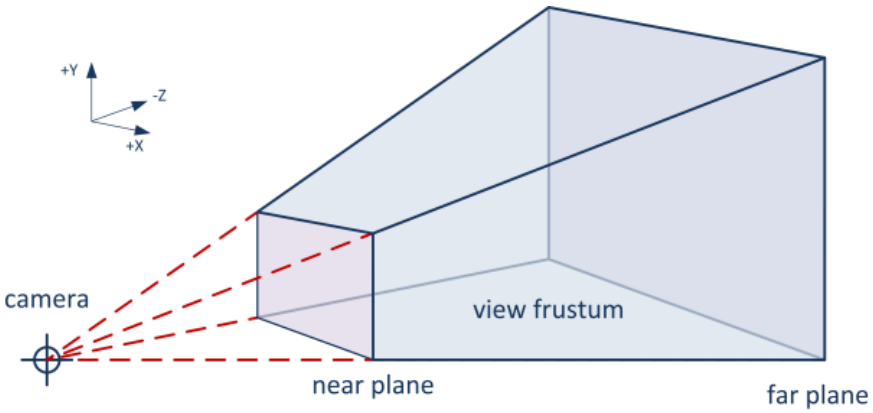

Figure 4.2: Back projection of 2D bounding box [ref]

In the above figure 4.2, we can see the bounding box which is represented as the near plane which

is back-projected to infinity. The view frustum area is the most probable area where the object can be located. To form an ellipsoid within the closed region from these 3D view frustums, we need at least 3 views which can be seen in the figure 4.3 below.

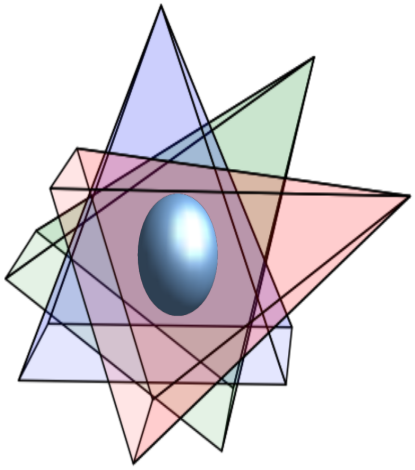

Figure 4.3: Closed region by 3 view frustums

The next important concept is what and how to represent a quadric. A quadric Q is a 3D surface that is characterized by a set of points x on the surface that satisfies the equation $x^T Q x = 0$. Q is a 4x4 symmetric matrix.

The same equation can be rewritten in terms of tangential planes that pass through each of these points which can create a closing envelope around the surface. $\pi^T Q^* \pi = 0$. Q^* is called a dual quadrics or dual representation of the quadric which is calculated by $Q^* = Q^{-1}$ if Q has an inverse or $Q^* = adjoint(Q)$.

Since Q^* is a symmetric 4x4 matrix, there are 10 unique elements. Three for centroid, three for size or radius, three for orientation and one for scaling factor which is usually kept constant at -1. Hence, only 9 parameters need to be optimized. Q^* can be represented as $Q^* = T Q_0^* T^T$ where Q_0^* is the representation of an ellipsoid centred at origin and T is the homogeneous transformation matrix of size 4x4 involving a rotation and translation parameter [R t] to transform the ellipsoid from origin.

$$Q_0^* = \begin{bmatrix} r_1^2 & 0 & 0 & 0 \\ 0 & r_2^2 & 0 & 0 \\ 0 & 0 & r_3^2 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

The parameters r_1, r_2 and r_3 govern the size or radius of the ellipsoid. The orientation and centroid parameters are incorporated as rotation and translation parameters in the homogeneous transformation matrix T. After matrix multiplication, the matrix Q^* turns into a transformation matrix where the centroid is the last column of Q^* as a homogeneous 4x1 vector. The orientation parameters pitch, yaw, and roll have to be extracted from the first 3x3 dimension of the dual quadric matrix.

The above concept of $\pi_i^T Q^* \pi_i = 0$ represents a 3D equation. However, the bounding box observations we obtained from the detectors are in 2D. We have to convert from quadrics to conics which is ellipsoids to ellipses in this case. The dual conics C^* can be written as $C^* = PQ^*P^T$ where the 3D ellipsoid Q^* is projected onto the 2D image plane using the camera projection matrix P mentioned before.

The definition of quadrics also remains the same for conics $x^T Cx = 0$ where x is now replaced with 2D points. And as tangential planes exist for points in 3D, tangential lines exist for points in 2D. Hence, in homogeneous coordinates, $\pi^T Q^* \pi = 0$ for quadrics could be replaced by $l^T C^* l = 0$ for conics where C^* is the dual conic representation and l is the set of tangential lines.

Substituting the expression for dual conics in the above equation yields, $l^T PQ^* P^T l = 0$.

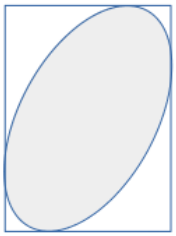

Figure 4.4: Four tangential lines to the object conic ellipse

Per detection of an object from a viewpoint, the bounding box generates 4 lines which can act as 4 constraints on the dual quadrics. For dual quadrics Q^*, there are 9 unknown parameters and atleast 3 observations from different viewpoints are needed to generate a solution for the same.

Internally, the quadricSLAM has the virtual representation of the 3D environment with the ellipsoids and when the camera pose changes to a viewpoint where the ellipsoid is visible, the 2D view of the 3D object is reprojected onto the image plane as in the figure 4.1. The concept is similar to the image 4.5 below except that in our case, the 3D object is ellipsoid and the 2D projection is ellipse.

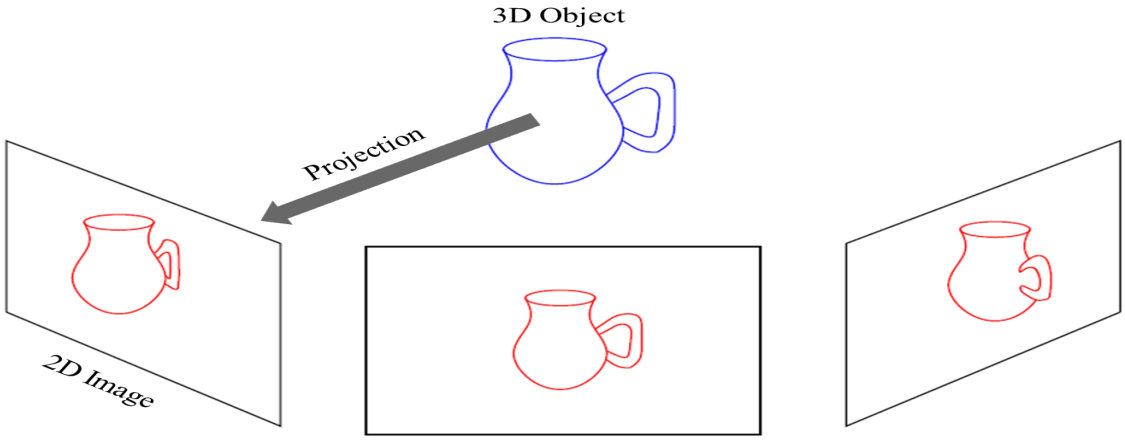

Figure 4.5: 3D virtual scene to 2D image plane projection [ref]

The following section explains about the mathematics behind using the dual quadric representation in the SLAM backend for optimization and corrections. The section is a summarized version of the content in section 4 of the Quadric SLAM paper [36].

To define the factor graph, we define certain parameters for the odometry and observations:

- A set of robot camera poses X = $\{x_i\}$ where i is the index of the pose at which a particular observation is made.
- A set of landmarks Q = $\{q_j\}$ where q_j indicates the dual quadric representation for the object j.
- A set of observations U = $\{u_i\}$ where u_i is the odometry data available via wheel odometry or visual odometry.
- A set of lines L = $\{l_ijk\}$ where k ranges from 0 to 4. l_ijk denotes the line k of the bounding box for object j being viewed at pose i.

The conditional probability distribution could be written as:

$$P(X, Q \mid U, L) \propto \underbrace{\prod_i P\left(\mathbf{x}_{i+1} \mid \mathbf{x}_i, \mathbf{u}_i\right)}_{\text{Odometry Factors}} \cdot \underbrace{\prod_{ijk} P\left(\mathbf{q}_j \mid \mathbf{x}_i, \mathbf{l}_{ijk}\right)}_{\text{Landmark Factors}} \tag{4.1}$$

This modelled representation of the factor graph tries to find the optimal solution for poses X^* and landmarks Q^* for a given set of odometry observations U and detected bounding box lines L. In other words, we are trying to find maximum a posteriori values for X and Q through non-linear optimization strategies.

The equation involving dual quadric parameter($l^T P Q^* P^T l = 0$) can be optimized through the least squares method. $\mathbf{q}_j^* = \text{argmin}_{\mathbf{q}_j} \sum_{ik} \left\| \mathbf{l}_{ijk}^\top \mathbf{P}_i \mathbf{Q}_{(\mathbf{q}_j)}^* \mathbf{P}_i^\top \mathbf{l}_{ijk} \right\|^2$. This maximizes the probability distribution for landmark factors in the above equation 4.1.

Further, maximizing the conditional probability distribution 4.1 could be simplified by minimizing the negative log of the probability. Thereby, the multiplication terms turn to addition and turn the entire SLAM problem into a least squares problem.

The factor graph is solved using the Levenberg-Marquardt optimization strategy for non-least squares problems and in this case, GTSAM [38] is the library being used in the backend for efficient computations by taking advantage of the sparse nature of the jacobian matrixes.

If depth information is available, it could also be added to the least squares problem as an extra error term that can represent additional knowledge of the problem which can be added as a factor in the factor graph.

$$X^*, Q^* = \underset{X,Q}{\text{argmin}} \underbrace{\sum_i \left\| \mathbf{e}_i^{\text{odo}} \right\|_{\Sigma_i}^2}_{\text{Odometry Factors}} + \underbrace{\sum_{ijk} \left\| \mathbf{e}_{ijk}^{\text{quadric}} \right\|_{\Lambda_{ijk}}^2}_{\text{Quadric Landmark Factors}} + \underbrace{\sum_{ij} \left\| \mathbf{z}_{ij} - \mathbf{T}_i\left(\mathbf{q}_j^{\text{t}}\right) \right\|_{\Omega_{ij}}^2}_{\text{Relative Position Factors}} \tag{4.2}$$

Each of the three factors is trying to minimize the odometry error, dual quadric parameters error and centroid of the quadric (last column of dual quadric) error respectively. In the third term, z_{ij} is the observed depth of the object j at pose i and $T_i(q_j^t)$ is transforming the centroid from the last column of the dual quadric representation from the world coordinate to the robot coordinate at pose i. So the last term is for centroid correction.

Another task is the variable initialization for robot poses x_i and dual quadrics q_i. x_i is initialized with the help of odometry data available from wheel odometry or visual odometry and q_i is initialized as an identity matrix of size 4x4.

Orientation Factor

The same authors later came up with a continuation of this work to incorporate prior knowledge of the world to better orient the quadrics within the environment using orientation constraints [39]. The prior knowledge is that we need to assign each object class to one of the categories such as Horizontal, Vertical or Unassigned. This corresponds to what is the normal pose of the object in relation to gravity. For example, a chair can be Vertical and a laptop can be Horizontal.

Further, the major axis of the quadric is aligned close to the z-axis(vertical line) or close to perpendicular to the z-axis(horizontal line) based on the prior knowledge of the normal poses of the objects. To find the major axis, eigenvalues are calculated from the dual quadric representation involving only the rotation components and an eigenvector corresponding to the largest eigenvalue is identified. Further, a cosine similarity between this vector and the z-axis is calculated. This value should be optimized towards 1(cosine similarity of 1) if the prior knowledge of the object pose is Vertical(gravity, major axis and z-axis in the same direction). The value should be optimized towards 0(cosine similarity of 0) if the prior knowledge of the object pose is Horizontal(the major axis is perpendicular to gravity and the z-axis).

To implement this in the factor graph, the problem is made into a least squares problem and added as a factor to the factor graph. To the equation (2), a fourth factor is added as orientation factor.

$$\underbrace{\sum_{j} \|g\left(l_j\right) - c_j\|^2_{\sigma_j}}_{\text{Orientation Factors}} \tag{4.3}$$

c_j is the cosine similarity value being calculated and $g(l_j)$ is the actual expected cosine similarity for the particular label l of the object j which can be 0 for Horizontal and 1 for Vertical.

One of the problems of this approach is that we need to assign each known class to one of the categories such as Horizontal, Vertical or Unassigned. And, in the example they have shown, they assigned books to be horizontal which can be a wrong assumption as books kept on shelves are in a vertical orientation. This approach of adding the fourth orientation factor is not implemented in the available code of Quadric SLAM.

Sensor model

In the final paper released by the same authors in 2019 [3], they have added a sensor model to correct the errors in the predicted 2D bounding box of partially observable objects. We need to only consider the conic/ellipse part of the object within the image plane.

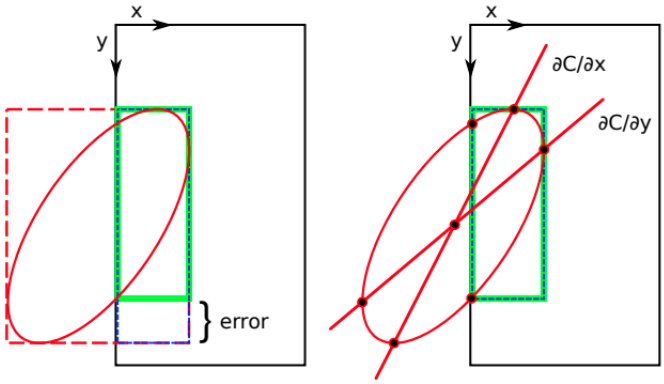

Figure 4.6: **Left**: The dotted red square is the predicted bounding box by the detector. A green square is the required bounding box. Violet is the error. **Right**: Error is removed through conic extreme x, y parameters [3]

In the above figure 4.6, the black box is the image viewed by the robot and only a part of the conic of the object is visible within the image plane. Normally, the edges of the bounding box are truncated based on the edges of the image plane. This resulted in the error component depicted by the violet colour where the object conic is absent.

To reduce this geometric error, the bounding box within the image plane is cropped using the conic extreme x and y parameters. For this, 4 points representing extreme points of the conic are selected along with the points that intersect between the conic and the 2D bounding box. All the points that lie outside the image plane are removed and from the remaining points within the image plane, the max and min values of x and y are determined which are used to create the accurate green 2D bounding box.

The results of the experiments showed that this feature significantly improved the estimated trajectory and the quadric mapping in terms of reduced trajectory and landmark quality errors.

The previously used error term in Quadric Landmark Factor was an algebraic one. That is, minimizing $\mathbf{q}_j^* = \text{argmin}_{\mathbf{q}_j} \sum_{ik} \left\| \mathbf{l}_{ijk}^\top \mathbf{P}_i \mathbf{Q}_{(\mathbf{q}_j)}^* \mathbf{P}_i^\top \mathbf{l}_{ijk} \right\|^2$. This is replaced by the geometric error proposed by the new sensor model, $\left\| \mathbf{b}_{ij} - \boldsymbol{\beta}_{(x_i,q_j)} \right\|^2_{\Lambda_{ijk}}$. Here, b_{ij} is the bounding box observed for object j at robot pose i. And $\boldsymbol{\beta}_{(x_i,q_j)} = BBOX(C_{ij})$ is the expected geometrically corrected bounding box for the conic C of the object j observed at pose i. $C = adjoint(C^*) = adjoint(P_i Q_j^* P_i^T)$ where P_i is the camera projection matrix at pose i and Q_j^* is the dual quadric representation of the object j.

The 2D object detector used in the experiments is YOLOv3.

Variable initialization

In the final paper of Quadric SLAM [3], another initialization method for quadrics is proposed. Using the dual quadric representation of quadric $\pi^T Q^* \pi = 0$, we can create a linear set of equations which can be solved using SVD if at least 3 views for the same object quadric is observed(9 unknowns and 4 constraints from each bounding box view). The last column of the decomposed matrix V gives the solution for the equation. The translation parameters can be obtained from the last column and the orientation parameters can be obtained from the first three columns of the solved Q^* matrix. The size is extracted from the eigenvalues. [3]

4.1.2 GTSAM

GTSAM [38] library is used to build the factor graph backend for QuadricSLAM and performs as incremental solvers. The robot pose and the dual quadric matrix are represented as nodes which are the latent variables that need to be estimated. The pose nodes are constrained by the odometry factors and the pose-quadric nodes are constrained by the bounding box factors.

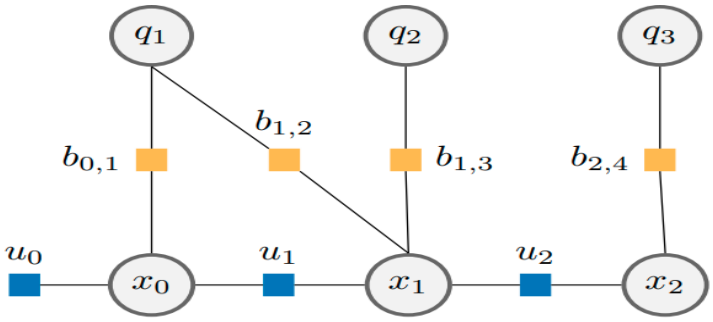

Figure 4.7: **Factor Graph of QuadricSLAM**: odometry factors in blue, quadrics bbox factors in yellow [39]

In the above figure, it can be seen that the quadric q_1 is observed from two robot poses x_0 and x_1. As the same object is observed from different poses, it adds better constraints to optimize the quadric parameters. Actions(in this case robot motion) introduce uncertainty whereas the observations (in this case the bounding boxes of detection) help to reduce the uncertainty in the pose graph.

Another important constraint in the factor graph is the loop closure constraint. Using the image or quadric features, when the robot detects that it is in a pose which was visited at a previous point in time, it can create a loop closure constraint which can help to reduce the overall error in the factor graph. This is also a field of challenge where we need to verify or make sure that the features are qualified enough for a loop closure to avoid false positive loop closure detection which can have a bigger negative effect on the factor graph.

The authors of QuadricSLAM extended the GTSAM functionalities by creating another library called gtsam_quadrics which handles quadric-related functions.

Some of the important functions used in QuadricSLAM are:

- gtsam.symbol - used to symbolically represent the robot pose and quadric variables as nodes within the factor graph.
- gtsam.noiseModel.Diagonal.Sigmas - can be used to define a noise model which is added into the factor graph when an odometry measurement or observation is used to add a constraint between 2 variables or nodes. It indicates with how much uncertainty, we are adding the measurement to the factor graph.
- gtsam_quadrics.ConstrainedDualQuadric - used to represent the quadric based on the given input parameters like rotation, translation, and radii matrixes as it has different constructors.
- gtsam.NonlinearFactorGraph - Used to initialize a factor graph onto which the robot pose and quadrics are added as latent variables along with the odometry and bounding box measurements as factors with an associated noise model.
- gtsam.Cal3_S2 - to generate calibration matrix
- gtsam.Pose3 - create a 3D pose of rotation and translation matrix based on the input provided.
- gtsam.Rot3 - create a 3D rotational matrix.
- gtsam.PriorFactorPose3 - to generate the initial robot pose with a prior noise model which is added to the factor graph.
- gtsam.BetweenFactorPose3 - to add the odometry measurement factor between two robot poses with an odometry noise model which is added to the factor graph.
- gtsam_quadrics.BoundingBoxFactor - to add the bounding box observation as a factor between a robot pose and a quadric which is added to the factor graph. Bounding box noise is also associated with the function.
- gtsam_quadrics.QuadricCamera.project - given a dual quadric matrix and robot pose, it will generate the 2D view(conic) of the quadric in the image plane from which using the function bounds(), we can obtain the 2D bounding box virtually.
- gtsam.LevenbergMarquardtOptimizer - Given the factor graph and an initial estimate for the robot poses and quadric latent variables, a non-linear optimization is applied to the graph to get the results.

It can be seen that there is a noise model attached to every odometry, observation and prior measurement factor. This creates the uncertainty region for adjusting the edges between the nodes during bundle adjustment. If the noise model is set to 0, it means the measurement is taken with 100% accuracy and then the node(in this case, the camera pose or the object pose) will be fixed or immovable.

4.1.3 Code explanation

Overall framework

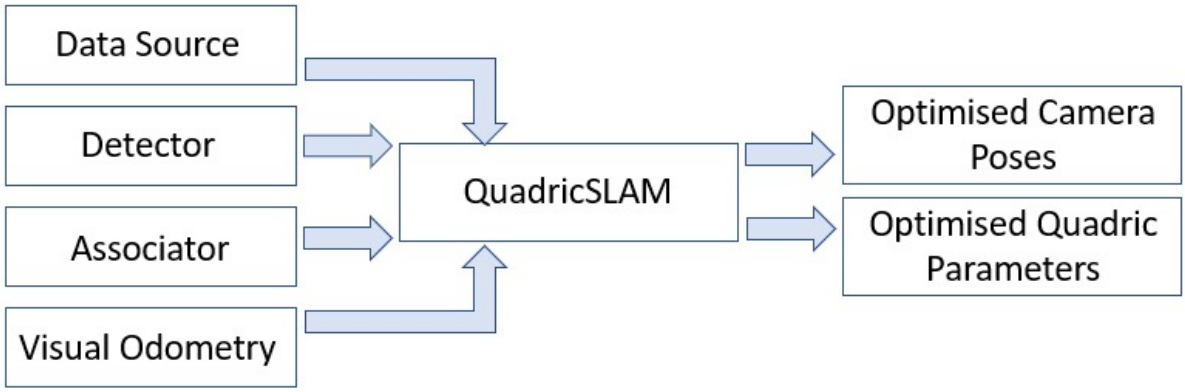

Figure 4.8: QuadricSLAM framework

The above figure depicts the general functional framework of QuadricSLAM. QuadricSLAM is implemented as a class which takes in the class objects of data source, detector, associator and visual odometry. And what we get out of QuadricSLAM is the optimised camera poses and quadric parameters which can be used to create the map. Each component is explained below.

Data source

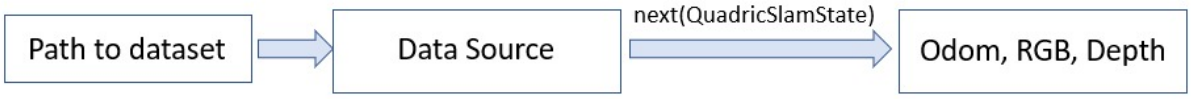

Figure 4.9: Data source

Used to either access an available dataset(acts as a data loader) or provide the live data to the QuadricSLAM algorithm. The data include odometry, RGB image and depth image. This data is retrieved by QuadricSLAM by calling the next() function of the data source. The odometry should be converted from quaternion or transformation matrix format to SE3() pose representation in spatial math. Datasource also maintains an internal counter to step through each data in time and also checks if all the images have been processed to stop the SLAM. Depth scaling factor and RGB calibration parameters are also stored here. Images are calibrated before passing into the QuadricSLAM. If using live data, then the configuration of the data source can be defined in the init() function. If using an available dataset, then the path to the dataset is to be passed into here.

Detector

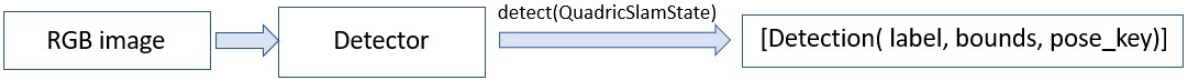

Figure 4.10: Detector

Used for detecting objects in each RGB image frame. The detect() function outputs the list with elements of object type Detection which contains the label of the object, a list of 4 values of the bounding box(x,y of top-left and bottom-right corners) and the pose_key which indicates at which pose number the current detection is made. The default detector used here is the detectron2 Faster-RCNN. A pre-trained model on certain labels is used here. Only those objects are detected. In the experiments which we are going to perform on the BOP dataset, we are going to avoid the usage of a detector and instead provide the bounding box directly available from the dataset. This will help us to evaluate the performance of QuadricSLAM given the noise from the bounding box is kept minimum.

Associator

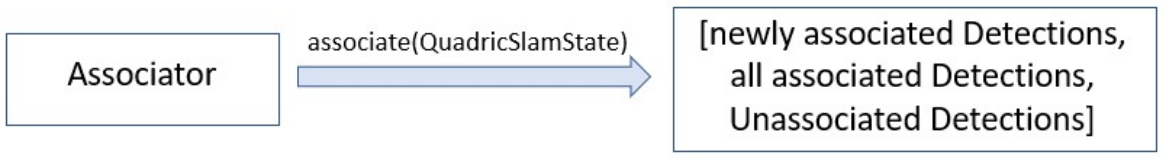

Figure 4.11: Associator

Data Associator takes the detections made in the current state and all previously unassociated detections and uses them to compare with previously associated detections to decide whether to match the unassociated detection with a previously associated detection(same quadric key to both detections) or to treat the detection as a new object(new quadric key to the detection). Basically, it decides whether a detection should be given an existing quadric key or create a new quadric key for it. The decision is based on the Intersection Over Union(IOU) value between the detection and an existing quadric. With respect to the current robot pose, all the existing quadrics within the factor graph are projected in the 2D plane as an enclosing bounding box. For each current detection bounding box, we are comparing its overlap with each bounding box generated within the map of QuadricSLAM. If the overlap is below a certain threshold, then treated as a new object with a new quadric key. Otherwise, associated with an existing object with the quadric key of the overlapping quadric. In the init() function, we can specify the threshold above which an association should be made. The associate() function outputs the list of the newly associated detections(objects for which a new quadric has been created in the current step), an

updated list of associated detections(all objects that have been associated with a quadric key), updated list of unassociated detections(objects that don't have a quadric key - usually passed as an empty list).

Visual odometry

Figure 4.12: Visual Odometry

Could be used to generate the odometry from current and previous RGBD images. Useful when no odometry information is available. Here, RgbdOdometry from the CV2 library is being used. The current and previous RGB images are converted into greyscale and provided as input along with their depth images to compute the odometry. odom() function returns the odometry value. Other visual odometry approaches suggested by the authors are ORB-SLAM2 with loop closure disabled or Kimera VIO(Visual Inertial Odometry) to get a less noisy odometry value.

Main code

There are two ways to initialize a quadric. initialise_quadric_ray_intersection() and initialise_quadric_from_depth(). In initialise_quadric_ray_intersection(), the translation and rotation of the poses at which a particular object is seen are used to project rays into 3D space and using the least squares method to find the closest converging point which is a good approximation of the quadric centroid. The initial orientation and size of the quadric is random. In the code, the size is set to (1, 1, 0.1). In initialise_quadric_from_depth(), the quadric can be initialized from a single view or image. To estimate the centroid, the z coordinate is calculated as the mean of the depth of points within the bounding box of detection. x and y coordinates are calculated as a calibrated value representing the centre point of the bounding box. The x and y radii of the quadric are calculated as a calibrated value of the height and width of the bounding box. The z radii are assumed to be a predefined object depth of 0.1. The orientation of the quadric can be estimated using the rotation, translation of the camera pose and the quadric centroid through gtsam.PinholeCameraCal3_S2.Lookat() function.

There are two modes of optimization. Optimization in a batch or in an incremental way. If it is set to true, then the optimization of the unknown variables occurs only at the end of processing all the dataset information. If false, then the optimization runs along with each incremental step. There are 3 noise models used in this application. One for the prior noise which is added along with the initial pose of the robot. Another one is for odometry measurement noise which is added to the constraint between two robot pose variables. The third one is for bounding box noise which is added for the constraint between a robot pose and the observation measurement which is a bounding box. This noise is due to the instability

of the bounding box being generated for an object which is evident for small objects.

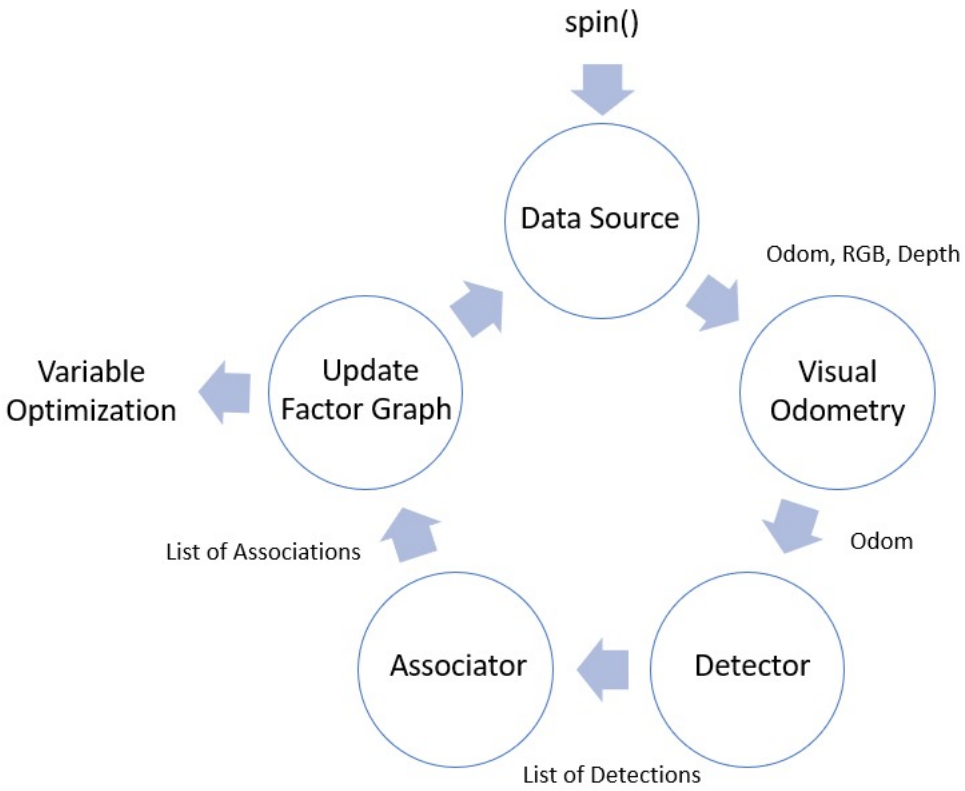

Figure 4.13: spin() function of QuadricSLAM

The above figure depicts the main operation of the QuadricSLAM which is the spin() function. The next() function of the datasource returns the current odom, RGB and depth. If there is no Odom data, a visual odometry method could be used to generate the Odom data. The detector takes in the current RGB image and outputs a list of detections with information about the label, bounds and pose_key for each detected object. detect() function is used. The associator takes these detections and uses the IOU method to identify whether it is a new object or an existing object in the graph. All the associations and unassociated detections are output by the associate() function. Based on the new pose and quadric information, the non-linear graph is updated. This cycle of process continues until the dataset is completely processed. Based on the optimization batch setting that has been set, the unknown variables are either optimized at the end of processing all the datasets or done in an incremental fashion during each cycle. The values to be recorded after running the algorithm are state.estimates - which has the initial guess of the quadrics and the poses onto which the non-linear graph performs optimisation and state.labels - which has the actual label for the quadric key which we can use for evaluation purposes. Since objects of the same class appear multiple times within the graph, we cannot just use the actual label within the graph. So they are represented by a random number and this is stored in the state.labels.

4.1.4 Key insights

Benefits

Geometric SLAM approaches utilise features like ORB [5], lines [40], and planes [41] to represent the environment. However, they lacked the semantic information for these geometrical features. These features were used only to estimate the robot's pose and create a dense point cloud of the environment. Whereas in Quadric SLAM, the knowledge of the semantic label of the geometric represented as a quadric can help in loop closure and the creation of a semantically enriched object-oriented map. As compared to the SLAM++ [26] approach where prior knowledge of CAD models is used to represent the objects when they are detected, the Quadric SLAM doesn't require priors. For example, there can be different variations of a chair and in SLAM++, it would be represented using the same CAD model of the chair which can affect the mapping accuracy. In Quadric SLAM, the position, size and orientation could be accurately estimated from multiple views of the chairs thereby producing quadrics of different parameters for each type of chair. In SemanticFusion [30], each point in the point cloud has a label and the 3D object reconstruction from the dense point cloud is performed as a post-processing step. In the proposed Quadric SLAM approach, the object representation as quadrics(mapping) is performed in an online fashion. Also, the sparse representations can help in easier storage of the map. For example, the 9 parameters of the dual quadric could be saved as an XML file.

Challenges

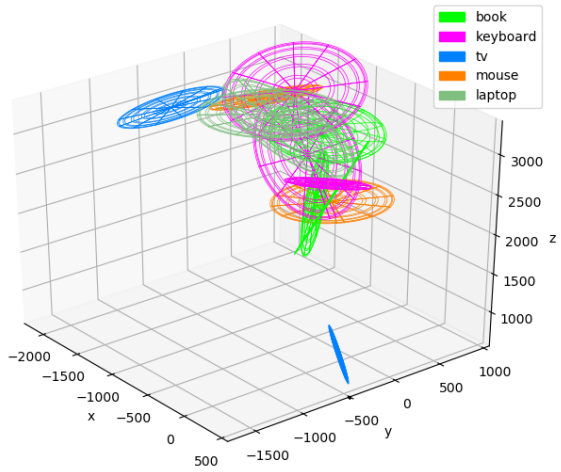

Figure 4.14: Batch optimization

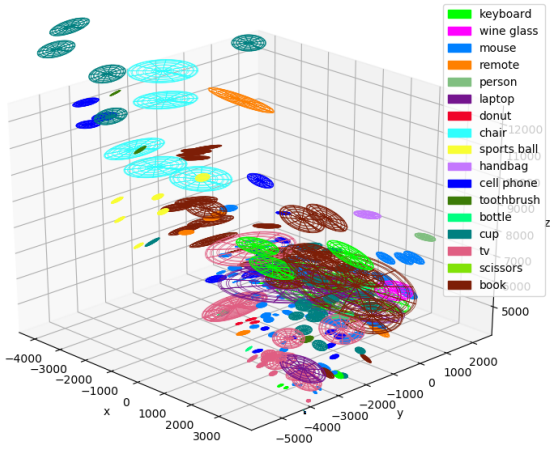

Figure 4.15: Incremental optimization

The above plots represent the quadric representation of the freiburg1_desk scene of the TUM RGBD dataset. In figure 4.14, the optimization of the factor graph is performed only at the end of reading

the entire dataset(batch optimization). The initial guess of the odometry value is the same as that of the provided ground truth odometry as we didn't use any explicit odometry estimation algorithm here. Because of that, the data association in each step of the data performed well in associating quadrics that belong together and thus generating 14 detected objects or quadrics in the end. Whereas in the incremental optimization approach (figure 4.15), at each step, based on the initialized quadric from a single detection and the ground truth odometry, the factor graph is optimized. Thus the actual ground truth odometry is pulled to a wrong position to compensate for the noisy quadric detected. This error accumulates over each time step and each detection of the same object in each data step appears at different locations in the environment. This is the data association problem in QuadricSLAM that leads to duplication. Thus 236 quadrics are generated in the end as most of the detections of the same object couldn't be associated together. There exists no method within QuadricSLAM to remove wrong or noisy detections from the factor graph and is retained till the end.

Another observation is that, since a single detection is used to generate a quadric, the uncertain predictions of the detector will generate some non-existing objects due to the wrong label prediction. In the figure 4.15, objects like person, and sports ball were not present in the environment. There should have been a counter to detect the number of times a particular label appears for a quadric and the final label should be the most occurring one. This is possible only if the data association problem is solved. Loop closure is not implemented within QuadricSLAM. This is because there is no mechanism to detect whether the current view has been previously visited or not. This limits its operation to smaller environments where precise odometry and bounding box observations available so as to have fewer accumulated errors.

There is no culling or removal of noisy object detection observations and keyframes that are redundant. For example, if the camera is static at a position, new frames are continuously pushed into the factor graph which doesn't provide any new information but rather increases the computation load of the matrix that needs to be optimized. The object is initialized even with a single observation as an ellipsoid. This can be erroneous in most cases as noisy observation(incorrect label) in a single instance could cause the object to be retained in the factor graph. Also, the depth is estimated as the average depth of pixels within the bounding box which also can cause errors in the case when a majority of pixels that don't belong to the object within the bounding box have a far depth. So the object will get initialized with an incorrect centroid and further data association would also be not possible.

4.2 OA-SLAM

4.2.1 Related works

4.2.1.1 ORB-SLAM

ORB-SLAM [5] is a feature-based monocular visual SLAM that uses ORB features to track the camera trajectory poses and map the environment by reducing the reprojection error. This error reduction process is called as bundle adjustment where the camera poses that observed the map points and the 3D projected position of the map points are jointly optimized in an iterative manner. This is a resource and

time-intensive process. Most of the approaches that used feature-based methodology have implemented bundle adjustment in succession to the tracking step which made the single iteration of the SLAM very slow which limited its real-time operation.

PTAM(Parallel Tracking And Mapping) [22] was a distinctive SLAM approach where they separated the tracking module and mapping module into two separate threads. This allowed the faster tracking component to not be blocked by the slower processes in the mapping part. Also, the bundle adjustment was separated into local and global bundle adjustments. The local one is used in the tracking thread, where the current frame and a window of 4 adjacent frames along with their neighbouring frames are used to perform efficient bundle adjustment to estimate the initial pose of the current camera frame. The global bundle adjustment is used in the mapping thread where a bigger window of frames is selected along with their map points and a slower but accurate bundle adjustment is performed. The system's implementation as separate threads made the real-time operation achievable.

Still, the system is not said to be complete as it lacked some important capabilities such as loop closure detection. Loop closure is needed to identify whether a previously mapped region is detected again and if yes, then the current observation should be aligned with that region to close the loop and minimize the accumulated error so far through full bundle adjustment. Loop closure is implicitly present in PTAM when both the loop ends are present in the global bundle adjustment window and that too in very similar viewpoints. The limitation is due to the absence of feature descriptors that can be used to localize the current frame by comparing it with the descriptors of other frames using the Bag-of-Words approach. FAST(Features from Accelerated Segment Test) [23] is a corner extraction method which is used as the keypoint in PTAM for real-time operation. PTAM was restricted to in-plane motion as FAST features are not invariant to rotation and scale. This highlights the need for a robust keypoint and descriptor having better viewpoint invariance.

Feature descriptors

Some of the popular keypoint extraction and feature descriptor methods used in feature-based SLAM methods were SIFT [42] and SURF [43]. SIFT stands for Scale-Invariant Feature Transform. The scale-invariant keypoint extraction is performed by creating image pyramids by scaling down the image and also in the same scale level, gaussian blur is applied in an increasing fashion. Further, a difference of Gaussian is performed by subtracting between the adjacent Gaussian blurred images in the same scale and the resultant features should satisfy the local extrema checking condition with its neighbouring pixels in the same scale and in the adjacent scales. To get the dominant orientation vector of the keypoint, an orientation histogram is formed using its neighbouring pixels so that we can use it to align the image patch to achieve rotation invariance. Now we need to define a feature descriptor that can act as a unique signature for this keypoint. For this, a 16x16 patch with a keypoint as the centre is selected, and in each 4x4 subblock, a discretized 8-bit orientation histogram is formed. So in total, there are 16 subblocks with 8 bins in each of them which is the 128 dimensional feature descriptor. So if a query image has a keypoint with a feature descriptor, the correspondence matching can be performed by checking the similarity distance between the feature descriptors. Illumination invariance is achieved by normalizing these feature descriptors. Even though the accuracy of the SIFT descriptor was high, the 128-dimensional

floating point vector consumes a lot of computational time for feature descriptor matching. This led to the faster SURF(Speeded-Up Robust Features) feature descriptor. They used the concept of integral image(image with pixel value equal to the sum of pixels from rows and columns until that pixel including the pixel value) to speed up the computation in constant time. The difference of Gaussian is approximated using a box filter operation whose value can be computed in constant time using an integral image with the help of the pixel values at the four corners of the filter. Using this, the image pyramid in SIFT is replaced by the scale pyramid by adjusting the kernel size of the box filter. And determinant of the Hessian matrix is computed to identify the keypoints. To make it rotation invariant, the Haar wavelet filter of size 4x4 is applied in the local neighbourhood of the keypoint which is also approximated as a box filter which takes advantage of the integral image and forms the orientation histogram. Further, the 16x16 local patch is formed with 4x4 subblocks and a haar wavelet filter is applied in the x and y direction to form 4 gradient descriptors for each subblock. This gives a 64-dimensional feature vector(4x4 blocks and 4 features in each). This is much faster than SIFT as the descriptor size is half and computation time is reduced with integral images.

Still, the SIFT and SURF feature descriptors were not fast enough to perform real-time operations. That's when ORB(Oriented FAST and Rotated BRIEF) feature descriptor [44] was introduced that uses FAST keypoints along with BRIEF descriptors to achieve 10 times the speed of existing methods. FAST (Features from Accelerated Segment Test) [23] is a corner detection algorithm that tests corner conditions on keypoints to select robust points. With respect to the keypoint candidate pixel, a Bresenham circle of radius 3 is formed which passes through surrounding 16 pixels. The intensity value of the candidate pixel is compared with the intensity value of these pixels and at least more than 50% of the continuous arc of pixels should have higher or lower intensity than the candidate pixel, then it could be selected as a keypoint. Robust points are selected through non-maxima suppression by comparing the intensity difference value(gradient) between the candidate pixel and the pixels on the arc. There can also be edge points among these which can be removed through the Harris corner detection condition. The problem with FAST is that is not rotation and scale invariant. To achieve scale invariance, an image pyramid is formed. For rotation invariance, the image moment is calculated from which the centre of mass and the dominant orientation could be found and the image patch can be oriented along that direction. For the descriptor part, the BRIEF(Binary Robust Independent Elementary Features) binary descriptor is used. The advantage of using binary descriptors is that we can compare the descriptors using Hamming distance which is the sum of XOR operations and it is computationally fast which helps ORB to achieve run-time performance. In BRIEF, an image patch around keypoint is selected and a fixed strategy is used to compare the intensity value of pairs of 2 pixels and outputs a binary value of 0 or 1. The most popular strategy is the Gaussian sampling and 256 pairs are compared to form a 256-bit feature descriptor which is much more efficient than the 128 and 64-dim floating point vectors. BRIEF is also not rotation invariant. So the pixel-selecting strategy is also rotated with respect to the dominant orientation. To make the BRIEF strategy uncorrelated and discriminative, a greedy search is performed to compare each of the tests and select the 256 pairs having the least absolute correlation.

Data structures in ORB-SLAM

Keyframe: It is used to represent the pose of the camera that generated the 2D image. It stores the camera extrinsic matrix, intrinsic matrix, and list of all the keypoints that were detected in the current frame.

Map points: It stores the 3D position(x,y,z) of the point, a unit vector which is the mean of all the rays from the viewing keyframes optic centres to the 3D point, its ORB descriptor and a min and max distance range within which the map point can be detected as an ORB feature.

Covisbility graph: A graph where nodes are the keyframes and the edges between nodes are established if they have common visible map points which are weighted by the number of such points.

Essential graph: A subset of a covisibility graph where only edges having a weight greater than 100 are retained. If a covisibility graph could be seen as a dense representation, then an essential graph could be seen as a sparse representation and is used to perform pose-graph optimization after loop closure in an efficient manner.

Visual place recognition

This is a module which is activated only when tracking is lost and needs to relocalize in the environment. The place recognition is done with the help of the bag-of-words approach. For this, a prior training phase is needed where the most commonly appearing ORB descriptors in all the scenes are identified and represented as visual words in an ORB vocabulary. When each keyframe is observed, if any of these visual words appear in the keyframes, the keyframe ID is stored within a list corresponding to the visual word. So when the camera tracking is lost, when it observes some visual words, the database can be queried with these words and the keyframe ID which appears in most of the visual word's list is returned. But the problem is, that keyframes that are close together may have similar scores as they observe the same visual scene. Further refinement is done with the help of map points matching the detected features through the guided correspondence search method which can return an accurate unique pose.

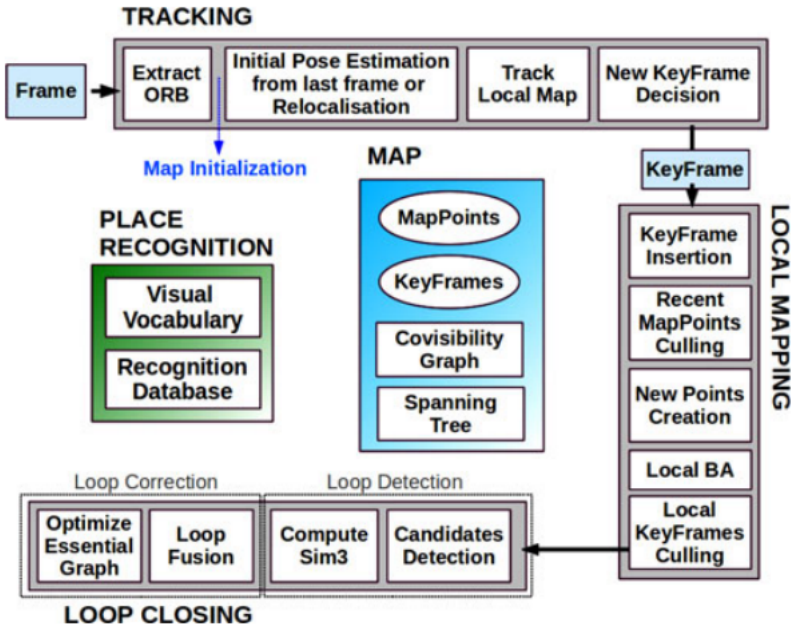

Figure 4.16: ORB-SLAM1 pipeline [5]

The above image represents the overall implementation of ORB-SLAM in which the system is running on three parallel threads: Tracking, Local Mapping and Loop Closing.

Tracking

This is the input side of the SLAM system that takes in RGB images. ORB features are extracted from these images which can be used to triangulate and form the 3D points within the map. The first step is to initialize the map. For that, two frames with significant parallax are needed. The first frame is selected as a keyframe and the second keyframe is delayed until a good viewpoint rather than initializing the map incorrectly.

The scene that is observed can be a planar or non-planar scene and the Homography matrix is to be computed in the first case or the Fundamental matrix is to be estimated if it is the second case. Since the map initialization is to be done automatically, both the models are computed in parallel and symmetric transfer errors are used to compute the score for each matrix. The best model is selected based on the ratio of these scores and the relative motion is estimated using the eight-point algorithm in the case of a Homography matrix or using the five-point algorithm in the case of a fundamental matrix. After this, a full bundle adjustment is performed with the map points and the two keyframes.

Next is the tracking part. Whenever a new keyframe is input into the system, its initial pose is to be estimated. This can be done using the motion model followed by the correspondence matching between the map points and the keypoints in the current observation. There can be cases where abrupt motion can cause the tracking to be lost. In this case, the visual place recognition database is to be invoked to get an initial estimate of the pose which is further refined using the PnP algorithm where RANSAC is

used to detect the inliers. This can help in a guided search of current keypoints with more mappoints to refine the pose.

A local map is always maintained within the tracking thread that consists of the current keyframe, a set S1 of keyframes that share common map points with the current keyframe, a set S2 of keyframes that have edges with the keyframes in S1 in the covisbility graph and also all the map points visible in all the above keyframes. This is to make the tracking thread computationally efficient by not taking in irrelevant scenes of the environment for tracking. There are still some ORB features in the current frames that are not matched with the map points. So the current map points in this local map are projected into the 2D image frame of the keyframe and consider the map points that lie within the image bound. If the viewing ray from the optical centre of the camera to the map point is within 60 degrees of the mean unit vector of the map point and the keyframe is within the scale visibility min-max range of that map point, then the unmatched features can be matched with these map points if they are closeby.

Next step is to insert the keyframe into the global map. For this, certain conditions have to be met. It should track more than 50 existing map points and not more than 90% of map points of the reference keyframe(the keyframe that has the most overlap with the current keyframe). This is to make sure that the keyframes with redundant information are culled out as the information gained from such keyframes is less and would lay as a computational burden.

Local mapping

This is the second thread of ORB-SLAM. The covisiblity graph is maintained here along with refining the map by adding new information and by removing non-robust and redundant information. The first step is the keyframe insertion. These are the keyframes that are suggested by the tracking thread. They are inserted into the covisibility graph and the edge connections are established with keyframes having common map points. Also, the bag-of-words descriptors of this keyframe are stored in the relocalization database.

The next step is the culling of newly added map points. The newly added keyframe may contribute to expanding the map by adding new map points. To check the robustness of these map points, they are to be visible in at least more than 25% of the keyframes(at least 3 keyframes is the minimum criteria) in which it is predicted to be visible(conditions are the viewing angle should be within 60 degrees of the mean unit vector and the viewing scale distance should be within the min-max range). If not, these map points are culled out as they may be noisy observations.

The map can be enriched with more map points. As the map is refined over time, more unmatched ORB features of different keyframes may align through triangulation and can be used to create new map points. The conditions of map points are also to be satisfied.

After which a local bundle adjustment is performed in the local mapping thread. It consists of the newly added keyframe, a set S1 of keyframes that share common map points with the current keyframe, a set S2 of keyframes that have edges with the keyframes in S1 in the covisbility graph and also all the map points visible in all the above keyframes. The keyframes in S2 have other neighbours who are also added in the bundle adjustment but their pose is not refined(they act as anchors).

The final step is the culling of keyframes which again checks for redundant keyframes. This is because, after the local bundle adjustment, the poses of keyframes are changing. The condition for culling is the same as that mentioned in the last part of the Tracking thread.

Loop closing

This is a thread that is continuously checking for loop closure candidates and checking if loop closure is possible or not. In monocular camera-based SLAM, loop closure is important as the scale drift error may accumulate over time. For this, the bag of words similarity score is computed between the current keyframe and its neighbouring keyframes. This acts as a minimum threshold. The loop closure candidates are queried in the visual place recognition database with the current keyframe and if they have a score above this threshold, they are considered as good candidates. Also, the database may return neighbouring keyframes as good candidates which should be removed.

Since there is no depth information, a similarity transformation is to be done to identify the 7 DOFs (3 for rotation and translation each and 1 for scale) that can align the current keyframe with the loop closure candidate. For the correspondence matching, we already have the descriptors and RANSAC can be used to identify the number of inliers. From all the loop closure candidates, the one having the highest number of inliers and if this inlier count is above a threshold percentage, then loop closure is said to be established.

Once loop closure is valid, we can connect both ends of the loop in the covisibility graph and new edges can be formed. The pose of the current keyframe is transformed with the similarity matrix. The matching inlier mappoints are fused together. The similarity transformation is propagated through the neighbours of the current keyframe. More map points may align together now which also can be fused.

The final step is the Essential graph optimization which is basically a pose graph optimization using an essential graph which is a subset of the covisibility graph. So only the keyframe poses are refined and the map points are readjusted if any one of their viewing keyframe pose is updated.

Takeaway from ORB-SLAM1

As per the results mentioned by the authors, the single iteration of the tracking thread took around 31 ms which made the SLAM operate at around 30 FPS which is a good value for the real-time demands. The mapping thread took 384 ms and the most exhaustive process within it was the bundle adjustment which took 296 ms. Since the tracking and mapping thread was separated, there was no blocking issue that would affect the real-time performance.

The process of culling the redundant keyframes and non-robust map points helps in reducing the size of the overall covisibility graph and map and thereby an informative efficient bundle adjustment could be performed.

As compared to previous approaches like PTAM and LSD-SLAM which initialized the map from the first two frames, ORB-SLAM delayed the map initialization process until two frames of significant parallax were observed. This helps to prevent the further accumulation of errors.

The use of ORB feature descriptors helped in more accurate and faster detection of keypoints with

rotation, translation, and scale invariance. As a result, the same ORB descriptor was able to be used for relocalization if tracking was lost even though the viewpoint changed. This was not possible in the case of PTAM which used FAST features and limited its operation to in-plane motion.

Bundle adjustment operation was performed only in local regions. This is the most expensive operation. When loop closure is performed, instead of full bundle adjustment of the entire map, only pose-graph optimization is performed to refine the keyframe poses. The map points were adjusted from these refined keyframe poses only. This was a way to perform faster optimization. And even for the pose-graph optimization, the entire covisibility graph is not selected, but the more informative subset called the essential graph is used which reduces the size of the matrix to be optimized within the computation.

Loop closure in ORB-SLAM helped to reduce the accumulated scale drift error. Also, it helped to remove redundant keyframes observing the same scene. In PTAM, if the same scene is observed again, the keyframes are continuously pushed into the graph and the size of the graph increases with this redundant information. It was not able to detect the loop closure scene.

One of the edge cases in ORB-SLAM was when the same scene was revisited in the opposite direction in which it was mapped. Most of the ORB features were not able to be detected as the viewpoint has changed extremely. The keypoints were occluded in the current view.

Another problem in ORB-SLAM1 was due to the use of the monocular camera. The true scale is unknown. The map that is generated is with respect to some relative scale that the optimization software module suggested for faster computation. This is one of the reasons for scale drift error and also this limited the direct evaluation of the generated map and the ground truth map. We have to perform a similarity transformation to align the generated map with the ground truth map.

4.2.1.2 ORB-SLAM2

ORB-SLAM2 was mainly used to address the issues with the monocular camera RGB images that cause scale drift and also the requirement for two frames with significant parallax to initialize the map. Now, the SLAM can operate on RGB-D images or Stereo images. The changes are briefly explained below. The figure 4.16 and figure 4.17 depicts the overview of the changes in the form of block diagrams.

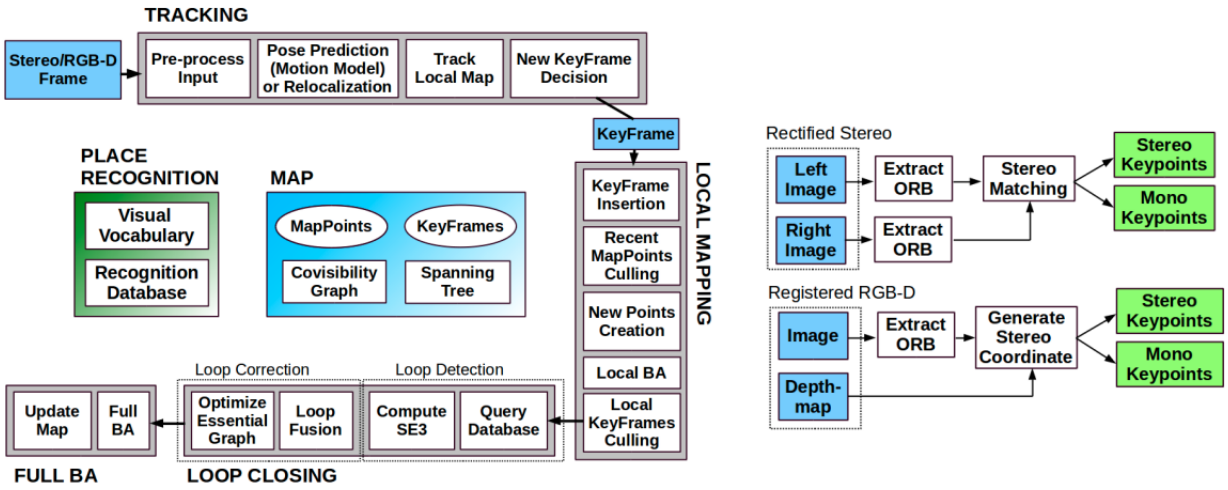

Figure 4.17: ORB-SLAM2 pipeline [45]

In the input part of the tracking thread, both the RGB-D and stereo images can be provided as input. They are preprocessed and represented as stereo keypoints and mono keypoints so that the rest of the system isn't affected by the type of input that is provided among RGB-D and stereo. If the input is stereo images, then ORB features are extracted from the left and right images, and correspondence matching is performed to connect the features in the left and right images through horizontal epipolar lines. The stereo keypoint representation consists of 3 parameters, the u_l, v of the left image and the u_r of the right image where u and v are the horizontal and vertical pixel coordinates respectively. The v is the same for both the left and right images which enables us to form horizontal epipolar lines. In case the input is an RGB-D frame, to represent them in stereo keypoint format, the RGB image is treated as a left image from which u_l, v can be estimated and a virtual right image coordinates u_r is formed using the depth and camera intrinsic parameters. Along with stereo keypoints, there are mono keypoints which represent the unmatched features in the stereo image or in case the depth value is invalid in the case of RGB-D image. For these keypoints, we just store the u, v value in the left image in the case of stereo images or in the RGB image as in the case of RGB-D setup. These keypoints are similar to those which are formed in ORB-SLAM1 input which may need multiple keyframes to find the correspondences. This preprocessing step is represented in the right image of figure 4.17. Because of the availability of the stereo keypoints, the map initialization can be performed from the first keyframe itself which is located at the origin.

The local mapping thread is similar to that of ORB-SLAM1. No structural changes have been made. A new condition is added for selecting a frame to be a keyframe. The frame should have created at least 70 new close stereo keypoints and should track at least 100 close stereo keypoints from the previous keyframes. Close stereo keypoints refer to the stereo points that lie close to the camera which have significant disparity between left and right images. Whereas far stereo points lie far away causing the disparity to be small and translational motion cannot be estimated well. They could be only useful for rotational motion estimation.

In loop closure detection in the loop closing thread, for ORB-SLAM1, we had to compute the similarity transformation for alignment because the true scale was unknown. It had three rotations, three translations and a scale parameter. Now, in ORB-SLAM2, since the true depth information is available, rigid body transformation is to be performed which has only rotation and translation parameters. This saves the computational cost of estimating the scale.

A new thread called Full bundle adjustment was created. Previously, in ORB-SLAM1, after the loop closure detection, we performed only pose-graph optimization which optimized only keyframe poses and map points were refined based on the keyframe pose changes. In ORB-SLAM2, a full bundle adjustment is performed which is a resource-intensive process but turns out to increase the accuracy of the map as compared to ORB-SLAM1. Because of this, it is running on a separate thread to avoid blocking the loop-closing thread. During Full BA, if any new loop closures are detected, the Full BA is halted, and the latest map with updated new loop closures is used to restart the Full BA. After Full BA, the global map is updated with optimized keyframe poses and map points and the new keyframes and map points generated by the local mapping thread are added to this global map which is refined using the optimized keyframe poses.

Additionally, a new localization mode was introduced along with the default SLAM mode where the local mapping and loop closing threads are turned off. The visual place recognition module is invoked to get the initial pose after which the tracking thread will perform the tracking based on the correspondence between map points and the current ORB observations.

4.2.2 OA-SLAM summary

OA-SLAM [2] stands for Object Aided SLAM which utilizes the objects as a key feature for aiding mapping and camera relocalization. It takes advantage of the benefits of both point features and object coarse model representation. The object representation as ellipsoids is based on the concept that is discussed within the QuadricSLAM section 4.1 above. The results of QuadricSLAM showed that the joint optimization of the camera poses and the object poses resulted in decreased accuracy of the camera poses or the trajectory. The main reason was that the joint optimization step in QuadricSLAM takes the current estimated pose of quadric or the ellipsoid for incremental optimization which already has a lot of uncertainty associated with that. The pose of a quadric would be useful only when it is stable which can be achieved through significant viewpoint changes around the object. This uncertainty causes the camera trajectory to deviate significantly from the ground truth path causing further unstable observations. This is why the incremental optimization provided unsatisfactory results and batch optimization provided better results because, in batch mode, the entire observations of the objects were fed together which provided useful information. This highlights the importance of point features for precise localization which is inspired by ORB-SLAM2. A point represented as a single pixel is a more stable anchor than an ellipse(2D view of the ellipsoid) which is represented by a set of pixels. However, the downfall of point features in ORB-SLAM2 was that they cannot be observed beyond the viewable distance and also when the point feature is occluded especially when viewing the scene in the opposite direction. That's where OA-SLAM utilizes the benefits of both.

The underlying architecture is similar to that of ORB-SLAM and additional object modelling modules are added. The point features are used for tracking and when there is not a sufficient number of point features, then the OA-SLAM relies on objects for localization. This is because keypoints can be viewed from certain viewpoints whereas the object models can be viewed from the top, bottom, sides and even from far away. Even though the object is partially occluded, the visible region can still provide valuable information for overlap checking. Also, present-day object classification models can provide 2D bounding boxes and labels at a really fast rate which makes the approach feasible for real-time applications such as augmented reality.

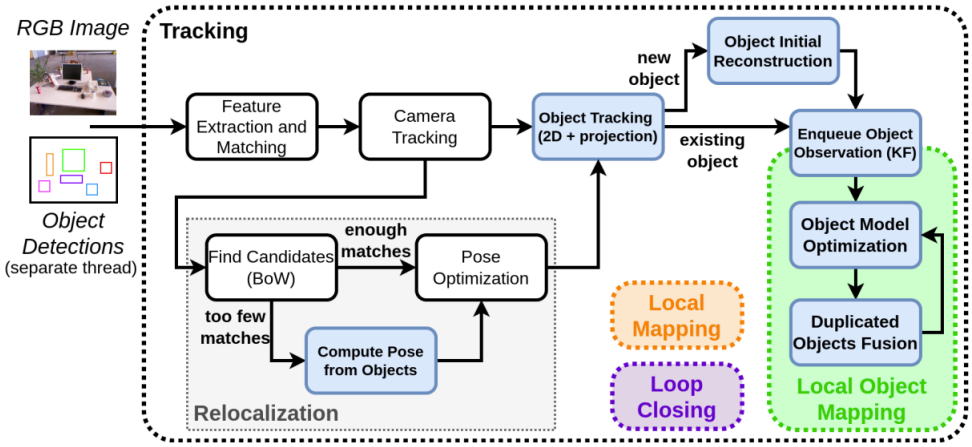

Figure 4.18: OA-SLAM pipeline [2]

The above figure4.18 represents the overall pipeline of OA-SLAM. Even though they mentioned that the SLAM was based on ORB-SLAM2, the architecture is very similar to ORB-SLAM1 where monocular RGB images are given as input. Along with the image, the 2D bounding box from the YOLO-based object classifier is passed in which is running continuously on a separate thread. For our experiments, we used the ground truth bounding box information available from the dataset and didn't use YOLO. The blue box indicates the newly added modules. In the tracking thread, along with keypoint initialization and tracking, object model initialization and tracking are also added. The local Mapping thread is unaffected, instead, a new local object mapping thread is added. The local mapping thread only handles keyframes and keypoints and the object model refinement operations are handled within the Local object mapping thread. Also, the Loop closing thread is unaffected. The relocalization module is updated to use object-based landmark information when sufficient keypoint feature descriptors are not detected for the camera pose estimation. Each of the topics will be explained in detail below.

Object model representation: A dual quadric representation is used to model the object. It stores 9 parameters of which 3 each are for position, orientation and scale information. These parameters can be used to represent an ellipsoid. The mathematical concept is explained in the previous section 4.1 on QuadricSLAM. Dual quadrics metrics represent the 3D shape and the 2D shape, in our case the

ellipse(conic), can be reprojected onto the 2D image frames using the camera projection matrix(consists of intrinsic and extrinsic parameters). Such a representation is difficult in the case of cuboidal models as it has vertices which can be matched with several points on the 2D box which is computationally not efficient. In the case of ellipsoid, the representation of the dual quadric equation has a solution and the error term can be modelled to be added into the optimization framework.

Object detection: YOLO is the object classification model the authors have used. Along with the bounding box coordinates, the detection score is also input into the SLAM frontend. Only the detections having confidence above 50% are selected for further optimization framework.

Data assosciation: Data association was an important challenge that the QuadricSLAM tried to neglect. In OA-SLAM, there are two types of tracking of objects defined such as short-term tracking and long-term tracking. At least 3 views with significant parallax are required to initialize the ellipsoid as explained in the section 4.1 QuadricSLAM. Till that time, the objects are tracked between frames using the overlapping between the bounding boxes. This is called short-term tracking. Once the 3D model is initialized, then it can be reprojected into the current frame and the overlap checking can be performed. This is called long-term tracking. Short-term tracking is susceptible to abrupt movement causing it to lose track of it. The association problem is solved using the Hungarian algorithm defined below.

$$\begin{gathered}\mathbf{S}^t = \left[s_{ij}^t\right]_{N\times M} \\ s_{ij}^t = \max(\text{IoU}\left(\mathrm{D}_i^t,\ \mathrm{B}_j^{t-k}\right), \quad \underbrace{\text{IoU}\left(\mathrm{D}_i^t, \text{box}\left(\text{proj}\left(\mathrm{O}_j^t\right)\right)\right)}_{\text{or 0 if reconstruction not yet available}} \quad),\end{gathered} \tag{4.4}$$

There are N bounding boxes detected in the current image. There are M objects tracked in the map till the current timestamp t. The score s is computed for each bounding box and each object combination. It is the maximum value among the overlap IoU between the latest available 2D bounding box of the object B_j and the current bounding box D or the overlap IoU between the 2D projection bounding box of the 3D object O_j and the current bounding box D. The bounding box is assigned to the object having the highest score.

If the maximum score is below a threshold, then point-based object tracking is used. The above method is useful when there is no texture causing the keypoint not to be initialized. If there is some occlusion, then the IoU may return a lower score. So the keypoints are also linked to the objects. If the keypoint in the image falls inside the bounding box, then they are linked. Similarly in the global map, if a map point falls within an ellipsoid, then they are linked. So when a new detection bounding box appears, a minimum threshold of at least 10 points is required to check for data association. This highlights the advantage of using point features for data association in objects.

Initial object reconstruction: The objects are to be initialized as soon as possible to enable long-term tracking through reprojection. For this, at least 3 frames of sufficient parallax(greater than 10 degrees) are needed. The initial object is defined as a sphere with the centroid defined by the triangulated

points of the detection bounding boxes. Using the average bounding box dimensions and camera intrinsic properties, the radius of the sphere is also estimated. The initial orientation is an identity matrix. This sphere is refined to ellipsoid through subsequent reprojection optimization. The object model is said to be reliable if it has been refined through at least 40 frames with the object's projection and the bounding box having at least 30% overlap after which it is added to the global map.

Local object mapping: Local Object Mapping is a new thread that deals with the refinement of the initialized object models through an error function. The error metric which is used here is the Wasserstein distance which can be used to estimate the dissimilarity between the conic or the ellipse inside the current bounding box and the ellipse that is projected by the 3D dual quadric representation. This is an incremental optimization function that takes all the weighted Wasserstein distance of all the frames till the current frame. The weight is the detection confidence of the bounding box. This error is to be minimized by optimizing the dual quadric parameters. Since Wasserstein distance is used for probability distributions, the ellipse is represented as a 2D Gaussian distribution. The next operation within this thread is the object fusion to fuse duplicate objects together. There are three conditions for two objects to be fused. The objects are represented as cuboids using the ellipsoid parameters and if there is more than 20% overlap between the cuboids, then they are fused. Also, if the centroid of one ellipsoid is within the other one or if both the ellipsoids share at least t number of common map points within it, then they are fused. This refinement of objects using reprojection error and fusion of objects is the same as the Local Mapping thread of ORB-SLAM2 where it is done for points instead of object models.

Object-based relocalization: We discussed that the point features may not always be reliable as they cannot be viewed beyond a certain scale or could be occluded because of very extreme viewpoints. Whereas, objects are much more viewpoint-invariant and could even be viewed from afar. For relocalization in case of abrupt camera motions, if the number of point features is below a limit, then the object-based relocalization is triggered. Perspective-n-Point (PnP) method is used to estimate the initial pose using the 2D-3D object point correspondences. Further, this can help other map points to perform guided searches to find their correspondences for further refining the pose. But the problem is that we don't know which objects in the current image are linked to the objects in the map(correspondence). There can be multiple copies of the same object within the environment. For that, three pairs of objects having the same label in each pair are taken in each iteration and a P3P algorithm is performed which will return 4 poses. For each pose, the 3D objects are projected into the current frame and the cost which is the sum of $1 - IoU$ for all the detections is calculated. The one having the lowest cost is the most probable one. Further, guided correspondence of the keypoints in the image and the map points are performed. Since it is an iterative process, the pose with the lowest score and the highest keypoint-mappoint correspondence(above 30) is the relocalized pose.

4.2.3 Key insights

In the current SLAM, they haven't used the objects in the bundle adjustment process. In another version, they tried adding objects into the bundle adjustment which showed less accurate results which indicates to not use them. This is due to the uncertainty in the unrefined, less accurate representation of the actual objects in a smaller number of image frames. This explains the camera pose estimation inaccuracies in QuadricSLAM where joint optimization of the camera poses and the object poses are performed. A method to initialize, track, refine and fuse coarse object models is implemented. An object-based relocalization is established which solves the viewpoint variance problem in ORB-SLAM. This enabled the camera tracking to be robust to be used in AR applications. The authors also mention that as the camera moves towards the objects, the number of keypoints and visible objects decreases which causes the tracking to be lost. To fix that, part modelling could be implemented where instead of YOLO outputting a bounding box for object classes, it should produce a bounding box for each part of the object and thereby each object can be represented by n number of ellipsoids where n is the number of parts. A drawback was the absence of object culling just like MapPoint culling where outliers are removed. In the case of objects, it could be due to inaccuracy in bounding boxes causing it to generate incorrect labels or due to the dynamic objects present in the environment.

4.3 Structure aware SLAM using quadrics and planes

4.3.1 Summary

As a continuation of quadricSLAM, Sünderhauf et. al. also looked into extracting planes from the environment which can act as an extra constraint combined with the feature points within the factor graph [46]. This concept of infinite planes was first introduced into factor graphs as a least-squares optimization problem in the paper by Kaess [41]. Later, Taguchi et. al. presented a SLAM method where a combination of planes and points can be used to register 3D data since a single plane can be used to replace a lot of inlier 3D points which can help in faster computations and compact representations [47]. In this work, planes are also considered as an important feature which is the most common feature occurring in indoor environments. A plane can represent a big region of the environment, especially in cases where there are very few objects within the environment to be mapped and used as anchors. These planes can be used to create three additional constraints. The constraint between the points lying on the plane and the plane parameters, the constraint between the object and the supporting plane parameters and the constraint between two planes.

The following section explains the mathematics behind quadric and plane representations along with how the constraints can be added to the factor graph as a non-linear least squares problem. The section is a summarized version of the content in section 3 of the "Structure Aware SLAM using Quadrics and Planes" paper [46].

From the previous section 4.1 on QuadricSLAM, we have seen that the closed-form equation of quadric with tangential planes is given by $\pi^T Q^* \pi = 0$ where Q^* is called dual quadric representation. There are 9 unique elements within this symmetric matrix that need to be estimated through error minimization. So the error vector generated is 9 dimensions.

$$\mathbf{Q}^* = \mathbf{T}_Q \mathbf{Q}_c^* \mathbf{T}_Q^T = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} a^2 & 0 & 0 & 0 \\ 0 & b^2 & 0 & 0 \\ 0 & 0 & c^2 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \mathbf{R}^T & \mathbf{0} \\ \mathbf{t}^T & 1 \end{bmatrix} \tag{4.5}$$

Since we are fixing the quadric to be ellipsoid in this application, the dual quadric representation Q^* could be written as 4.5. T_Q is the transformation matrix that transforms the ellipsoid from its origin by rotating and translating it. Q_c^* is the ellipsoid located at the origin with radius a,b and c. The last element of Q_c^* is made -1 to represent the dual quadric matrix as an ellipsoidal equation.

$$\mathbf{Q}^* = \mathbf{T}_Q \mathbf{Q}_c^* \mathbf{T}_Q^T = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{L}\mathbf{L}^T & \mathbf{0} \\ \mathbf{0} & -1 \end{bmatrix} \begin{bmatrix} \mathbf{R}^T & \mathbf{0} \\ \mathbf{t}^T & 1 \end{bmatrix} \quad \text{where} \quad \mathbf{L} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \tag{4.6}$$

Further, to make sure that the first 3 diagonal elements of the Q_c^* matrix are positive eigenvalues during the optimization process in factor graphs, the equation 4.5 is updated to 4.6 so that it guarantees positive values.

Now updating Q^* means updating T and L where T is the transformation matrix of 6 parameters(3 rotations and 3 translations) and L is a diagonal matrix with 3 parameters(a, b, c scale parameters). Thus the 9 dimension error can be split into 6-dim and 3-dim errors for efficient computation.

$$\mathbf{Q}^* \oplus \Delta\mathbf{Q}^* = (\mathbf{T}, \mathbf{L}) \oplus (\Delta\mathbf{T}, \Delta\mathbf{L}) = (\mathbf{T} \cdot \Delta\mathbf{T}, \mathbf{L} + \Delta\mathbf{L}) \tag{4.7}$$

The updation of quadric parameters in Q^* can be simplified as in equation 4.7. The Δ values indicate the updated value. In the case of L, the updation operation is addition based on the first 3 values of the 9 dim error vector and in the case of T, it needs to perform the transformation operation for the updation based on the last 6 values of the 9 dim error vector. This representation of Q^* in terms of T and L also helps to add initial prior information about the properties of the object into the factor graph. The size info can be initialized in the L matrix and the rotation and translation info can be initialized in the T matrix.

Next, we have to represent the planes in mathematical form. Inspired by the work on infinite planes by Kaees [41], an infinite plane is represented by normalised homogeneous coordinates $\pi = \begin{bmatrix} a & b & c & d \end{bmatrix}^T$ to have a compact representation. n is the normal vector $\mathbf{n} = \begin{bmatrix} a & b & c \end{bmatrix}^T$ and d is the distance to the origin. This normal vector can be optimised using the rotation matrix algebra.

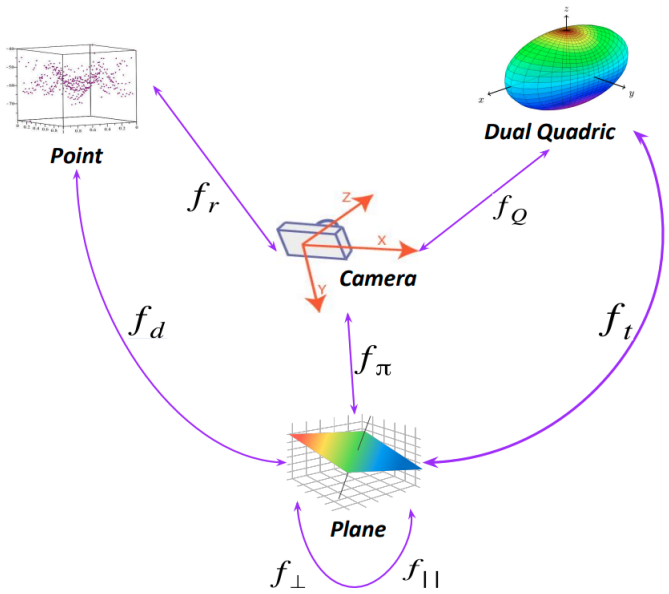

Figure 4.19: Factor graph containing points, planes and objects [46]

The above figure 4.19 shows the factor graph containing the nodes/ variables representing camera pose, 3D points, quadric parameters, plane parameters that need to be estimated and the connecting edges which represent 7 different types of factor constraints($f_r, f_Q, f_\pi, f_d, f_t, f_\parallel, f_\perp$).

1. Observation of points(f_r): The 3D points observed by the camera at a particular pose are to be registered within the factor graph. The equation tries to minimize the reprojection error.

$$f_r\left(\mathbf{x}_w, \mathbf{T}_c^w\right) = \left\|\mathbf{u}_c - \Pi\left(\mathbf{x}_w, \mathbf{T}_c^w\right)\right\|_{\mathbf{\Sigma}_r} \tag{4.8}$$

In the above equation 4.8, x_w represents the 3d point in world coordinates, T_c^w represents the transformation matrix of the pose of the camera w.r.t the world that can map the observed point in the current pose c to the world coordinate system. The error is calculated between the pixel location of the point u_c in the current pose c and the reprojected point from the world coordinate into the camera frame using the function $\Pi()$. The error function is the Mahalanobis norm $\mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x}$ where Σ is the uncertainty matrix associated with the factor. Mahalanobis norm takes the covariance structure of the data into consideration while reducing the reprojection error where the distance to the distribution is utilized instead of the distance to a single point as in the Euclidean norm.

2. Observation of objects(f_Q): The objects in this case are the ellipsoids. So need to reduce the reprojection error between the conic of the viewed object and the reprojected conic from the ellipsoid in the world coordinates at a particular camera pose c. More details on the conic equation formation in mentioned in the above section 4.1 on QuadricSLAM.

$$f_Q\left(\mathbf{Q}^*, \mathbf{T_c^w}\right) = \left\|\mathbf{C}^* - \mathbf{C_{obs}^*}\right\|_\mathbf{F} = \sqrt{\mathbf{Tr}\left(\left(\mathbf{C}^* - \mathbf{C}_{\text{obs}}^*\right)\left(\mathbf{C}^* - \mathbf{C}_{\text{obs}}^*\right)^\mathbf{T}\right)} \tag{4.9}$$

In the above equation 4.9, the norm being used is the Frobenius norm. It is measuring the magnitude of the matrix or in other words the root of summed squares of the elements of the matrix. Can be used to check for residuals. Since we are trying to reduce the reprojection error, ideally the value should be 0 or close to 0.

3. Observation of planes(f_π): The distance between observed plane π_{obs} at a particular camera pose c and the plane parameters in the world coordinate π projected into the camera frame is minimized. Since the normal vector n is used to describe the plane, the distance measured is based on the rotational distance in rotation space between the normal vector for π and π_{obs} which is explained in the work by Kaess [41].

$$f_\pi\left(\pi, \mathbf{T_c^w}\right) = \left\|d\left(\mathbf{T}_c^{w-T}\pi, \pi_{obs}\right)\right\|_\Sigma^2 \tag{4.10}$$

4. Point-plane constraints(f_d): If there is a certainty that a point lies on a plane, then the constraint between the point and the plane can be established as minimizing the orthogonal distance between the normal vector n representing the plane and a vector formed between the point x and a random point x_o on the plane.

$$f_d(x, \pi) = \left\|\mathbf{n}^T\left(\mathbf{x} - \mathbf{x}_o\right)\right\|_\sigma^2 \tag{4.11}$$

In the above equation 4.11, the dot product between the two vectors is taken. The normal vector n is perpendicular to the plane surface and if the point x is on the plane surface, then the vector connecting x and x_o would be perpendicular to the normal vector and the dot product would be 0.

5. Supporting plane constraints(f_d): In the real world, all the objects on the floor or on the walls are supported by a plane. For example, a clock is supported by a wall plane and a computer is supported by a table plane. This knowledge is exploited to create a constraint between the object and the supporting plane. The same equation which was used to define dual quadrics, i.e. a quadric is enclosed by tangential planes is used here also.

$$f_t\left(\pi, \mathbf{Q}^*\right) = \left\|\pi^T\mathbf{Q}^*\pi\right\|_\sigma^2 \tag{4.12}$$

6. Plane-plane constraints($f_\parallel, f_\perp$): Based on the Manhattan assumption, 2 planes can either be perpendicular or parallel.

$$\begin{aligned} f_\parallel\left(\pi_1, \pi_2\right) &= \left\|\left|\mathbf{n}_1^\top\mathbf{n}_2\right| - 1\right\|_\sigma^2 \quad \text{for parallel planes} \\ f_\perp\left(\pi_1, \pi_2\right) &= \left\|\mathbf{n}_1^\top\mathbf{n}_2\right\|_\sigma^2 \quad \text{for perpendicular planes} \end{aligned} \tag{4.13}$$

In the above equation 4.13, the dot product between the normal vectors of plane 1 and 2 is taken. In the case of parallel planes, the dot product would be 1 and the error term would be 1-1=0 and in the case of perpendicular planes, the dot product would be 0.

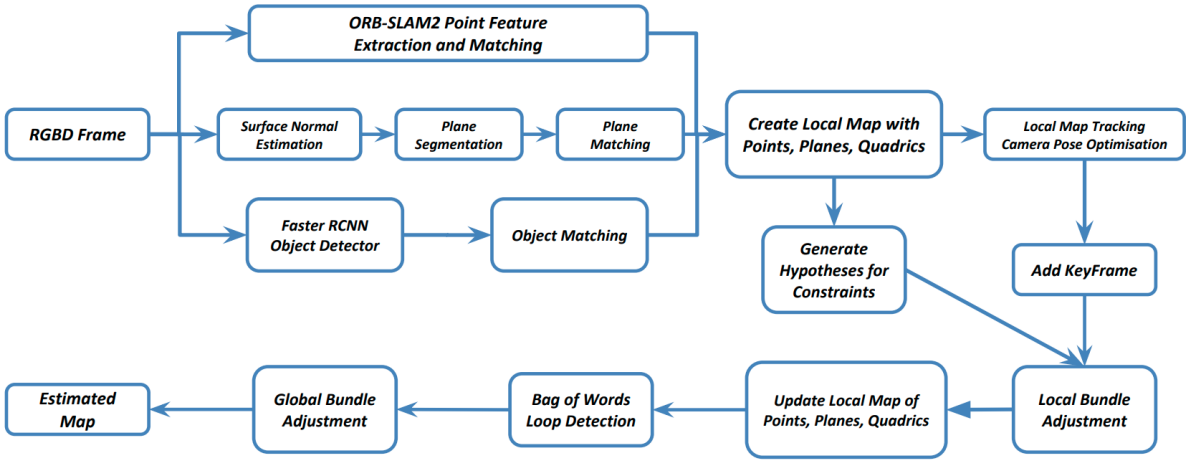

Figure 4.20: SLAM pipeline [46]

The above figure 4.20 represents the internal pipeline of the slam. This architecture is based on ORB-SLAM except for the fact that the plane and object detection are added in parallel to the point detection at the input side of the pipeline. The optimization step happens in an incremental fashion whenever a new keyframe is added along with its observations (basically an input image and the observed objects, planes and points). Loop closures are detected based on ORB feature matching using the bag of words method as in ORB-SLAM. The input is an RGBD image instead of an RGB image in ORB SLAM for the purpose of plane detection and object quadric initialization. The authors also plan to remove the use of RGBD images and use monocular RGB images in future work to perform depth estimation and semantic segmentation on single images.

1. Observation of points: This is the same as in the ORB-SLAM. Unique ORB features are identified, tracked and matched(data-association) between images. The depth initialization for these points can be made available from the depth channel if an RGBD image is provided as input or can estimate the depth between consecutive RGB images.

2. Observation of objects: A pre-trained Faster-RCNN is used to generate the bounding boxes through object detection. The conic is fitted into the bounding box as explained in the previous section 4.1 on QuadricSLAM. To deal with noisy detections, the object is initialized only if it has a confidence of above 95% for a class label. Also to solve the data association problem if multiple copies of the same object are present, a nearest neighbour approach is used to find whether they are 2 separate objects or inconsistent detection of the same object.

3. Observation of planes: Usually planes are detected using RANSAC which not be sufficient to meet the runtime requirements in SLAM to perform online operations. Also, we need a finite boundary for the planes instead of an infinite plane representation.

Trevor et.al [48] utilizes the availability of organized point clouds(2D image with depth channel as compared to LIDAR point clouds) to perform segmentation which can reduce the time taken for

neighbourhood searching step in unorganized point clouds. Each pixel is assigned a label and 2 pixels can have the same label if they are similar. In the first pass, the first row and first column of the image point cloud are assigned labels based on their similarity with immediate neighbours. After that, for each other pixel, its top and left neighbours are compared for similarity. If similarities are detected, then in the second pass of the algorithm, the labels are merged. To detect planar surfaces from these point clouds, for each point belonging to labels having a large surface(corresponding to floor, wall, ceiling), a surface normal is computed to represent the plane equation. The fourth plane component d is also computed using the dot product of the surface normal and the point's coordinate. Then the angular distance(dot product) between the surface normals and the L1 norm of the d distance parameters of the 2 points as the similarity factor to identify points belonging to a connected plane based on certain thresholds. Since smooth curved surfaces are also detected through this method, a maximum allowable curvature threshold is also set. To identify large surfaces, the minimum inlier point threshold is set. Later a plane refinement step is also performed to get a less noisy boundary by taking in unsegmented boundary pixels also. The authors also mention that the colour could be also used as a similarity measure by comparing in the Euclidean space.

For data association in the factor graph, all planes are considered as the number of planes in an environment is assumed to be sparse and also the viewpoint change of the planes between the subsequent image frames is assumed to be less. The data association can be performed by comparing the normal vectors and distance parameters of the plane equations.

4. Point-plane constraints: After performing the plane detection, the inlier points are detected by comparing the distance of the point from the camera with a threshold value. This is because the farther the point is from the camera, the depth will have higher uncertainty. If it satisfies the inlier condition, then it is added to the factor graph as a point-plane constraint.

5. Supporting plane constraints: The supporting infinite plane for the object is identified by comparing the orthogonal distance of the centroid of the object quadric with the plane. If it is less than a threshold defined by $max(20cm, a, b, c)$, then the object is supported by the plane. a, b, c is the radius of the ellipsoid and hence the threshold is dependent on the size of the quadric.

6. Plane-plane constraints: For establishing plane-to-plane constraints, every plane is taken into consideration(due to the sparsity of the planes) and a parallel or perpendicular constraint is established if the angle difference between the normals of the plane is within a certain window range. The uncertainty for this constraint is set to high so that the factor graph won't force them to be always perpendicular or parallel but act as a good prior for relative orientation between the planes.

4.3.2 Key insights

The implementation of the above approach couldn't be verified as the code is not open-sourced. So we didn't perform any evaluation of this SLAM concept. But still, it is worth mentioning that QuadricSLAM, OA-SLAM and Structure-Aware SLAM lie in the same path. As in QuadricSLAM, a factor graph-based approach is followed here. Points, Planes and Objects are added to the map and constraints are established with a camera and also with each other. As compared to OA-SLAM there is no constraint between points

and objects. The underlying framework is based on ORB-SLAM which enables robust odometry through tracking point features for camera pose optimization. Still, the culling or removal of noisy observations of objects and planes is not mentioned. This is vital to reduce the size of the map and also to make the map optimization robust to outliers. This also includes the absence of fusion of planes and objects that are duplicated. In the evaluation part, a qualitative comparison is performed as we don't have the ground truth mapping data available for the TUM-RGBD dataset. In quantitative comparison, only RMSE Absolute Trajectory Error (ATE) is estimated. The qualitative comparison is performed because the real world cannot be exactly and semantically mapped for the ground truth. So the best solution is to create a virtual environment in simulators like Gazebo or frameworks like BenchBot(link).

5

Evaluation and results

5.1 Experiment description

We will be comparing the evaluation metrics mentioned in section3.2 on 10 datasets generated in BOP YCB-V format. Each dataset will be performed on QuadricSLAM in batch optimization mode, QuadricSLAM in incremental optimization mode, and OA-SLAM which is by default in incremental optimization mode. We use the batch optimization mode in QuadricSLAM because the incremental mode is highly unstable with object observations and may create many duplicates of the same object instead of fusing them together. This is due to the fact that the ISAM2 which is used for incremental optimization is not able to solve the system of equations as it is underdetermined. So, the Levenberg–Marquardt optimizer which is used in batch optimization is utilized to solve the equations in incremental mode at the expense of increased computation time for each optimization step.

In the BOP YCBV dataset format, there are RGB and depth images. Also, 3 JSON files are present. scene_gt.json contains the object poses in terms of rotation and translation matrix for each object in a particular image. scene_camera.json contains the intrinsic and extrinsic parameters of the camera in each image. scene_gt_info.json contains the bounding box parameters for each object in each image. The bounding box parameters are in the format (x, y, width, height) where x,y is the top left corner. More details about the dataset format are given in the project website(url). The dataset is generated using a Blender-based setup. Each dataset contains 1500 images and the trajectory is a circular one. The resolution of the image is 640x480. To perform the comparative evaluation of the SLAM algorithms in general, we would be providing the ground truth 2D bounding boxes of the objects into the framework. However, for one of the datasets, we would be injecting noise into the bounding box to replicate the object detector uncertainties. In the next section 5.2"Results", the evaluation metrics for all the datasets on all the SLAM modes will be mentioned. In this section, one of the scene evaluations is explained in detail to get an understanding of the experimental setup.

Figure 5.1: Sample image from Scene 000009 of generated synthetic dataset

In the above image 5.1, the scene consists of 5 YCB objects namely, 051_large_clamp, 005_tomato_soup_can, 011_banana, 036_wood_block, 061_foam_brick.

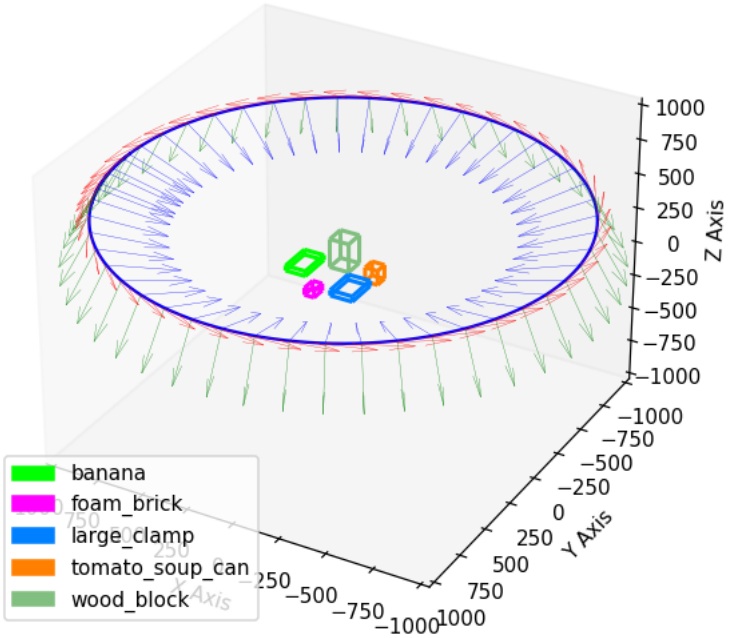

Figure 5.2: Ground truth plotted from JSON file

The above figure 5.2 represents the ground truth trajectory which can be seen as the thick blue circular line which is centred at the origin at a height of around 500 mm. In all our plots, we would be representing the dimension in millimetres. We can see there are 3 axes pointing from the circular ring. The red, green and blue arrows indicate the x, y and z axis of the camera. We can see that the blue arrow is pointing towards the objects. This is the z-axis. This is because when we represent an RGBD image, the 2D image dimensions are represented by x and y and the depth is represented by the z-axis. The objects are

distributed around the origin at a height close to the surface and they are represented using cuboids. The ground truth information of object dimensions is available from the YCB model data of the objects from which dimensions are used to plot the cuboid.

OA-SLAM OUTPUT

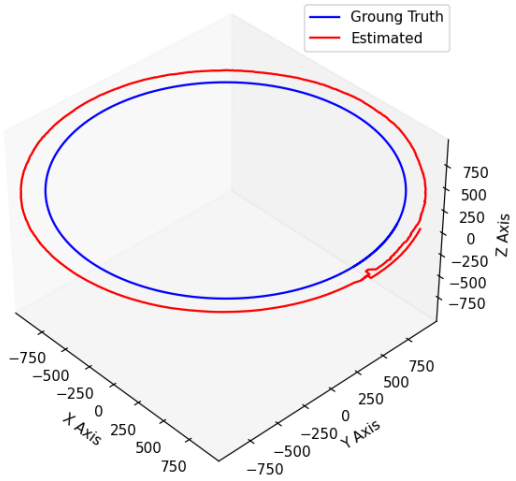

Figure 5.3: Estimated Trajectory

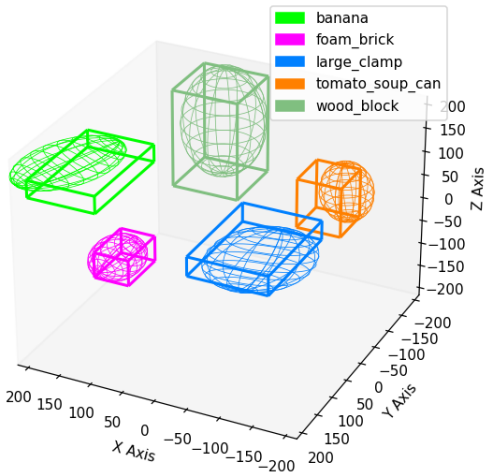

Figure 5.4: Estimated Objects

The output figure of OA-SLAM is split into two to get a better view of the mapped scene. The problem with OA-SLAM is that it uses a monocular camera and it creates a map on a relative scale. So, for quantitative analysis, we need to align them. So, the ground truth size of the object and the mapped size of the object is divided to find the scaling factor. Further, the initial pose of the estimated trajectory is aligned with the initial pose of the ground truth trajectory and the corresponding transformation is applied to the objects. There was a slight misalignment between the trajectories because of the scaling operation which was rectified using ICP to find the rigid transformation that best fit the estimated scene with the ground truth scene. The resultant figure is plotted above.

In figure5.3, the blue circle indicates the ground truth trajectory and the red circle indicates the estimated trajectory. We can see there is a scaling error existing that causes the estimated trajectory to be bigger than the ground truth trajectory in radius. However, the shape of the trajectory is well estimated. Also towards the right side of the trajectory, we can see the overlapping 100 extra trajectory frames we added for loop closure which is also well aligned. This indicates the importance of loop closure in monocular visual SLAM where scale drift error accumulates over time.

In figure5.4, we can see that the objects are well overlapped. The cuboid represents the ground truth object model and the ellipsoid is the estimated object model. Since all the objects are distributed around a circle, the estimated ellipsoids would have been well aligned with the ground truth cuboid if all the ellipsoids were pulled towards the origin or in other words, towards the centre of the circle. This would have been possible if the OA-SLAM had been well aware of the depth scale of the scene which would also

have made the estimated trajectory align with the ground truth trajectory.

Also, it is worth mentioning that the repeatability of the OA-SLAM is comparatively low as compared to QuadricSLAM in scenarios with low textures or distinct points within the environment. Since OA-SLAM has no depth information, it relies on the feature keypoints to estimate the relative scale. And this relative scale is very dependent on the map initialization process. Two factors affect the map initialization process. The first one is the dominant planar scene which exists in our datasets causing a scale ambiguity. The second one is the low-texture scene causing different less robust keypoints to be detected in the first frame each time we run the algorithm. This random point selection is dependent on our computer system properties.

QuadricSLAM OUTPUT - BATCH OPTIMIZATION

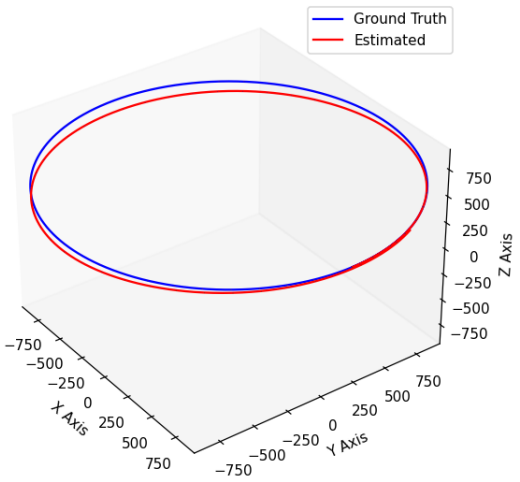

Figure 5.5: Estimated Trajectory

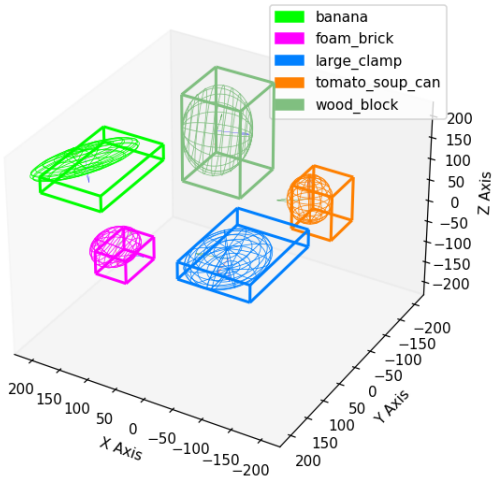

Figure 5.6: Estimated Objects

In figure5.5, we can see that the trajectory shape of the ground truth and the estimated ones are almost similar. This is because we have the depth image along with the RGB image which gave accurate scale information. And also the ground truth odometry is passed to the system with some noise. As compared to OA-SLAM, it doesn't need to perform visual odometry computation. So, even though the loop closure is not present in QuadricSLAM, the end position of the estimated trajectory aligns with the starting position. However, a small deviation can be seen on the half side of the circle that is opposite to the starting point. This is because, since we added 100 extra frames for loop closure, more constraints in the starting area cause the camera poses in those regions to have more fixed anchors as compared to the other side. So to compensate for the errors in the observations, it pulled the least constraint regions out of their true path as a result of joint optimization.

In figure5.6, we can see that the object shape and alignment are not as perfect as the OA-SLAM. Bigger objects are better mapped than the smaller objects. For example, the banana which is represented by light green is very flattened and well outside the ground truth cuboid. This is due to the fact that

bigger objects are well represented in the depth image as compared to the smaller objects.

QuadricSLAM OUTPUT - INCREMENTAL OPTIMIZATION

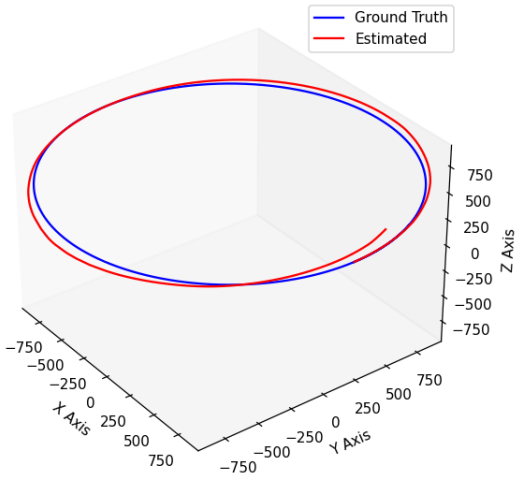

Figure 5.7: Estimated Trajectory

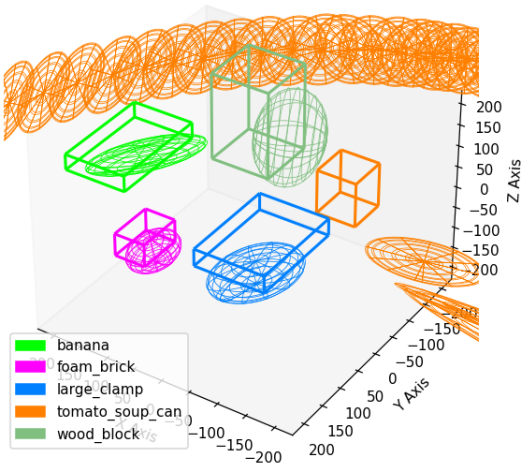

Figure 5.8: Estimated Objects

In figure5.7, it is visible that the trajectory is much worse than that of batch mode. In incremental mode, the optimized value so far is used as the initial best guess in the optimization process. If we compare the trajectory with the ground truth, it can be seen that initially, it went over the GT and later went below the GT and finally the loop ends do not even match. One of the error-causing factors in this case is the tomato_soup_can whose observations were not able to be associated together causing most of the detected bounding boxes for that object in each frame to be initialized as a new object. This is evident from the figure 5.8. This is the data association problem. However, the other 4 objects are a bit closer to their true position with a satisfactory overlap with the ground truth cuboid. These experiments with QuadricSLAM indicate the necessity of having point features for mapping the environment with better accuracy and not relying on the objects alone. The uncertainties in object detection may affect the accurate camera trajectory as they are jointly optimised.

EVALUATION

1. System profile comparison

	OA-SLAM	QuadricSLAM - batch	QuadricSLAM - incremental
Average CPU Utilization	30.59%	16.01%	58.97%
Min CPU Utilization	10.86%	10.61%	10.24%
Max CPU Utilization	35.96%	75.85%	71.43%
Average Memory Utilization	645.09 MB	209.43 MB	261.52 MB
Min Memory Utilization	128.53 MB	205.05 MB	211.66 MB
Max Memory Utilization	744.55 MB	234.16 MB	328.04 MB
Overall Time Taken	107.13 s	49.06 s	532.58 s

The test was performed on an Intel i5-8300H with 4 cores and 8 threads. The average CPU utilization is highest for the QuadricSLAM in incremental form. This is because we were using Levenberg-Marquardt optimization instead of iSAM2 for incremental updates. iSAM2 helps in updating the estimates based on the current observations without the need to perform the full graph optimization thereby useful in incremental mode. However, since the system is underdetermined, we had to use an LM optimizer which optimizes the entire graph in each iteration and thereby needs higher CPU utilization in each iteration. This is also the reason why the time taken for the complete processing of data took 532.58 seconds. For OA-SLAM the average CPU utilization was consistent at around 30% throughout its operation due to its implementation properties in different threads. In QuadricSLAM in batch mode, the CPU utilization was really low because the optimization is performed only once and that too at the end of processing all the input data. This is when the CPU utilization spiked at 75.85%.

A similar comparison can be made in terms of the memory utilization between OA-SLAM and QuadricSLAM. In OA-SLAM, along with the object representations, it needs to detect the ORB features in all the frames, maintain a dictionary to perform bag-of-words operation and maintain mappoint representations causing it to have a higher memory requirement as compared to QuadricSLAM. Whereas in QuadricSLAM, they only need depth images and just need the bounding box information of the RGB image. Also, in QuadricSLAM, they only focus on the current image frame and neglect all the previous ones for any operations. This is also a reason why QuadricSLAM has lower memory requirements. The codebase of OA-SLAM is bigger than that of QuadricSLAM and the code is also loaded into the memory while we run the program. In QuadricSLAM, both modes have comparable values. Another observable property is that as the number of input images or keyframes increases, the memory requirements also increase as the size of the map increases.

The overall time taken is not exactly comparable as OA-SLAM is written in C++ and QuadricSLAM is written in Python. But still, a rough analysis shows that OA-SLAM processed 1500 images in 107 seconds which means it can operate at 14 FPS. The quadricSLAM in batch mode is really fast as the expensive optimization step is performed only once. But in the online operation case, the incremental mode in QuadricSLAM is to be compared and this operation took 532 seconds which is roughly 3 FPS. Initially, the incremental mode had higher FPS as the graph to be optimized was smaller, but as more and more nodes were added to the graph, the optimization step became expensive. So, in total, given the accuracy and the real-time requirements, OA-SLAM seems to perform better.

2. Object mapping comparison

Figure 5.9: Object Centroid Error as Combined Plot

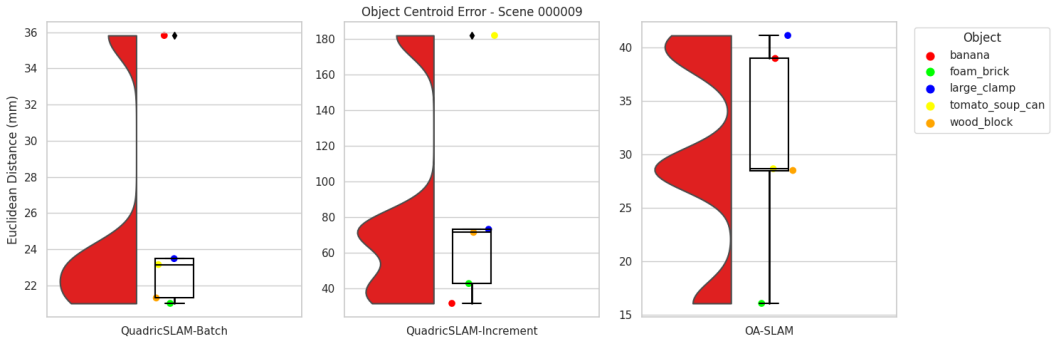

Figure 5.10: Object Centroid Error as Individual Plots

The above plot 5.9 gives a comparative view of the centroid error of the estimated object poses for each SLAM. Since the QuadricSLAM-Incremental version has very high errors, the error for its batch mode and OA-SLAM is not visible well. So, a separate plot for each SLAM is plotted in figure 5.10. By looking at figure 5.9, the huge error in the estimated tomato_soup_can is visible for QuadircSLAM in incremental mode. This is due to the data association problem causing the object not to be associated together and not optimized. This was already visible in the figure 5.8 where multiple copies of tomato_soup_can can be seen. The other 4 objects in the incremental mode of QuadricSLAM are comparable within an error range of 30 and 80 mm. When we compare OA-SLAM and QuadricSLAM in batch mode, the QuadricSLAM has a lower error and this is due to the knowledge of the exact depth at which the object is present.

When we look at the object-wise comparison in the second plot 5.10, the objects that are placed in an upright position, ie. foam_brick, wood_block, tomato_soup_can are having lower errors as compared to

horizontally placed objects like banana and large_clamp. This is because the 2D bounding box is always well-occupied or fits the object when viewed from the top whereas, there can be a lot of non-object related empty space in the 2D bounding box of horizontally placed objects. Banana, which is the smallest object showed higher error in both QuadricSLAM-batch mode and OA-SLAM.

When we computed the **average object centroid error** of each SLAM, for OA-SLAM, its **30.68 mm**, for QuadricSLAM in batch mode **24.95 mm** and for the incremental mode, its **80.25 mm**. This shows the better object centroid estimation capability of QuadricSLAM in batch mode.

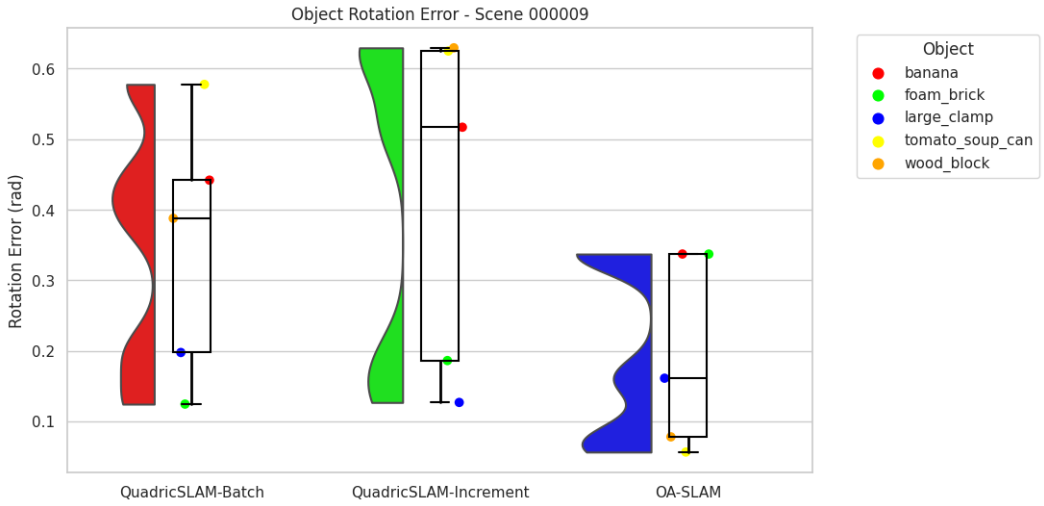

Figure 5.11: Object Rotation Error as Combined Plot

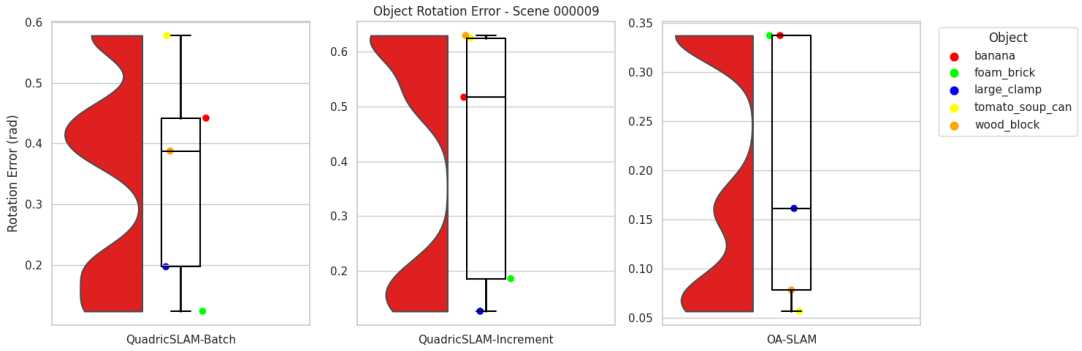

Figure 5.12: Object Rotation Error as Individual Plots

Overall, OA-SLAM has the least rotation error for the objects. The maximum error it encountered is around 0.34 rad as compared to error values around 0.6 for QuadricSLAM. In all three SLAMs, banana showed to have a higher rotation error. This can be due to its non-symmetric shape. The average rotation error is **0.19 rad, 0.35 rad, 0.42 rad** for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. This shows the OA-SLAM's capability to capture the object's pose accurately.

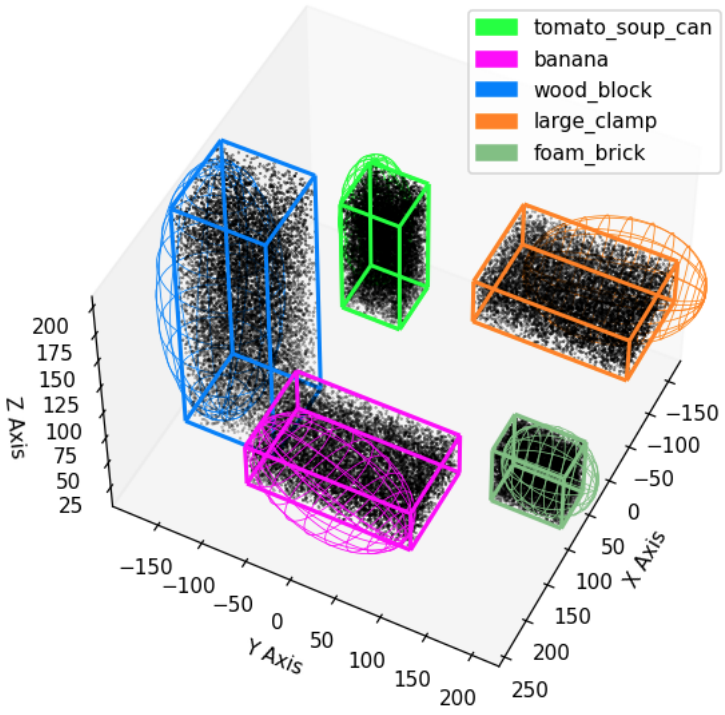

Figure 5.13: Object IoU Comparison Method

The above figure5.13 represents how we generate 10,000 samples within a ground truth cuboid to be compared with the estimated ellipsoid for overlap checking.

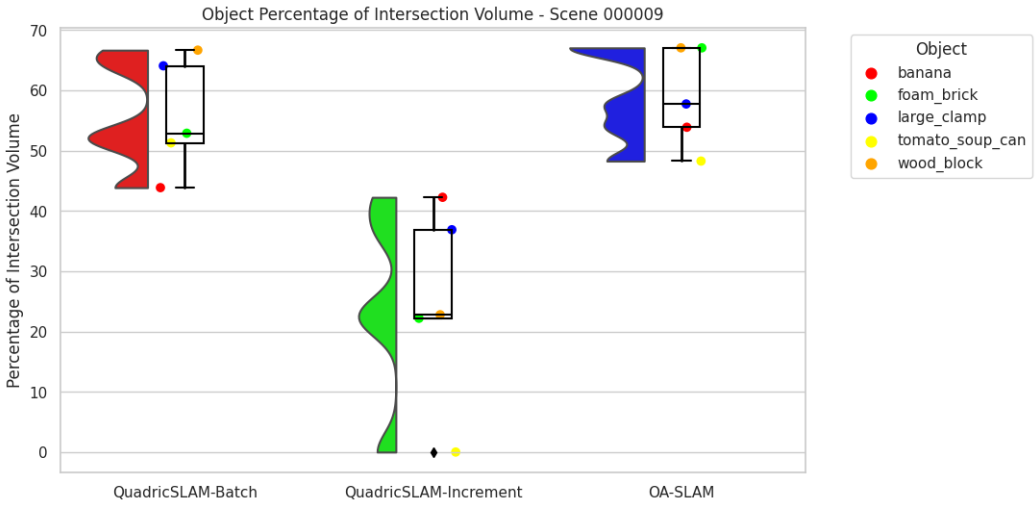

Figure 5.14: Object Overlap Percentage as Combined Plot

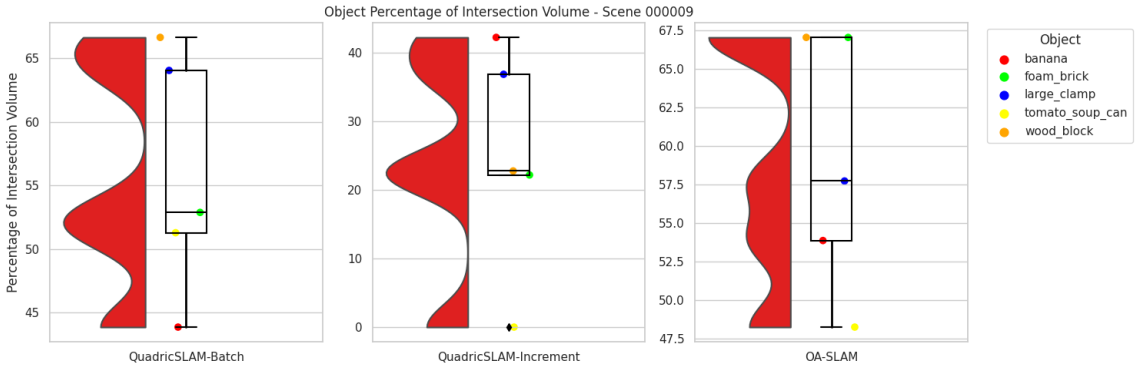

Figure 5.15: Object Overlap Percentage as Individual Plots

The above figures show that the OA-SLAM in general has a high overlap with the ground truth objects. QuadricSLAM in batch mode also exhibited a similar overlapping percentage. In the case of OA-SLAM, even though the centroid error is higher than that of QuadricSLAM, the exact capture of the shape helped in providing a good overlapping. In the case of QuadricSLAM, the centroid error was lesser, so even though the shape is not accurate, the object was well inside the ground truth to give a good overlapping result. The incremental mode of QuadricSLAM, even though has a lower overlap percentage is still having satisfactory results. The 0 overlap for tomato soup can is due to the data association problem. Another notable property is that the bigger objects were having better overlap because of their well-defined symmetric shape.

58.79%, **55.74%** and **24.82%** are the average overlap percentage for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively.

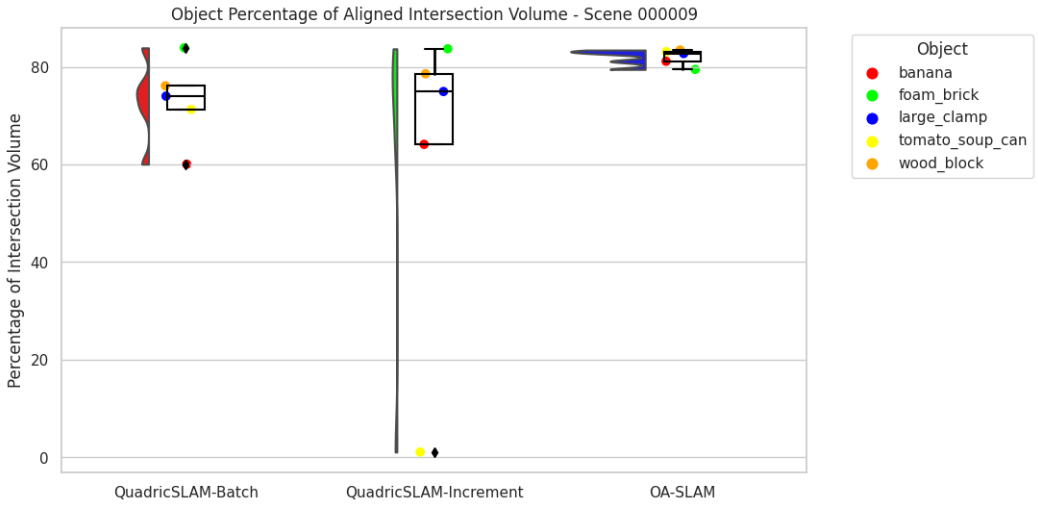

Figure 5.16: Object Aligned Overlap Percentage as Combined Plot

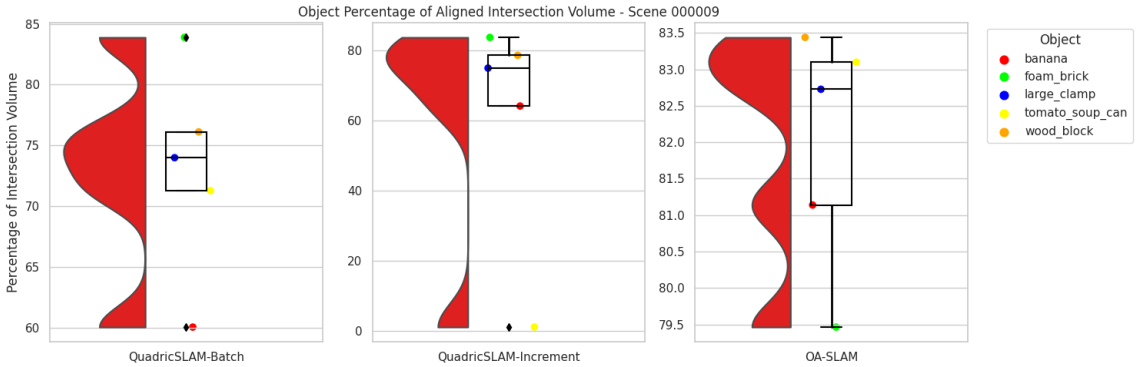

Figure 5.17: Object Aligned Overlap Percentage as Individual Plots

We can also align the objects with their ground truth pose and position and then compare the amount of overlap. This is performed to check if they have been aligned properly, would whether the intersection volume is high or not. Basically, we are comparing the ground truth volume and the estimated volume. The results show that the OA-SLAM had the best shape modelling. This is followed by QuadricSLAM in incremental mode which outperformed QuadricSLAM in batch mode by a small extent. The overall error of the increment mode of QuadricSLAM was affected by the unoptimized tomato soup can. Banana was still the shape that had the lowest shape similarity with the ground truth among all. This can be due to the curved shape of the object.

81.97%, **73.06%** and **60.48%** are the average overlap aligned percentages for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively.

3. Camera trajectory comparison

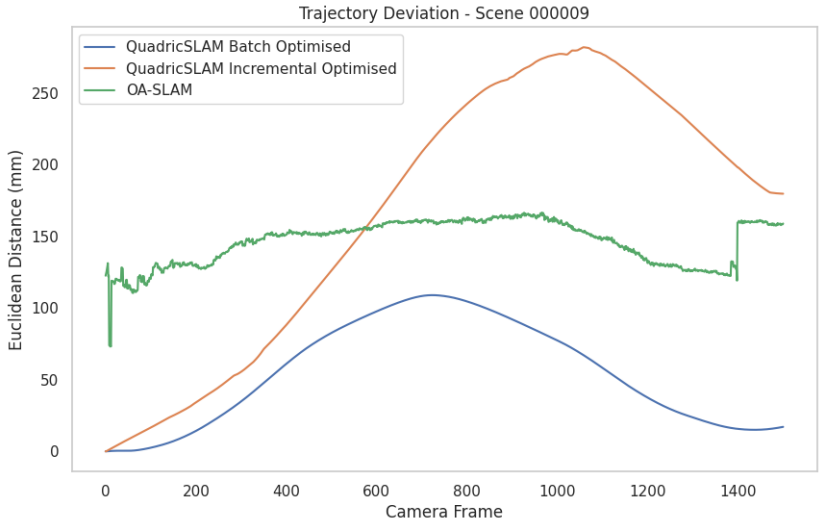

Figure 5.18: Trajectory Error Comparison

In the case of trajectory error comparison, we can see that the QuadricSLAM in batch-optimized mode is having the lowest errors. We can see that the error increases over time and peaks at around **108.87 mm**. This happens on the opposite side of the starting frame as it is the least constrained side. The QuadricSLAM in incremental mode has the highest error which is peaked at **281.64 mm**. In the case of OA-SLAM, the error starts at **73.07 mm** and peaks only till **166.45 mm**. This is a constant bias error due to the scale error of monocular camera-based SLAM. If we consider subtracting the least error **73.07 mm** from its error plot, then the error would only be peaking at around **93 mm** for OA-SLAM which is better than QuadricSLAM in batch mode. In the factor graph, similar viewpoints(loop closure views) at the starting and ending points make it a strong anchor and the viewpoints on the other side of the circular trajectory have a weaker anchor. So, when optimization is performed, the other side has a tendency to bend down to compensate for the object pose errors. This is why the errors are more on the side opposite to the starting or ending point.

146.88 mm, **65.45 mm** and **191.60 mm** are the root mean square error for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively.

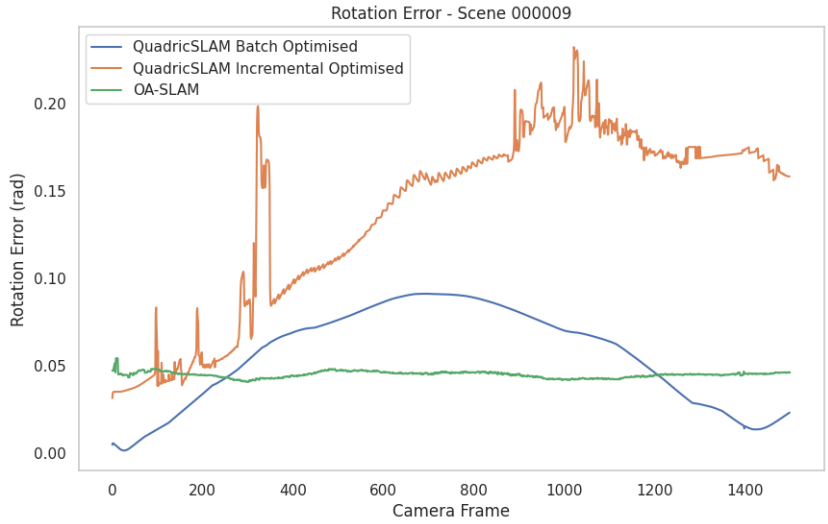

Figure 5.19: Rotation Error Comparison

The rotation error also follows a similar pattern as in the case of trajectory error. **0.04 rad**, **0.06 rad** and **0.14 rad** are the average trajectory rotation error for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. **0.05 rad**, **0.09 rad** and **0.23 rad** are the peak trajectory rotation error for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. OA-SLAM had the least error and that too which is a constant one.

Procrustes analysis - The estimated disparity value are $\mathbf{9.19}e^{-5}$, $\mathbf{1.38}e^{-5}$ and $\mathbf{1.67}e^{-3}$ for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. The lowest is for QuadricSLAM-batch. This value is the sum of the squared error between both the trajectories in normalised form and so has no unit. Procrustes analysis tries to align two given trajectory points and minimize the distance between

them. The disparity or distortion after aligning is returned as the disparity value which indicates a better similarity between the trajectories if the disparity value is low.

Fréchet distance - The estimated Fréchet Distance are **166.02 mm**, **107.60 mm** and **128.96 mm** for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. Fréchet Distance takes the shape of the trajectory curve along with its temporal feature to identify the length of the minimum leash distance that can connect two similar points of each trajectory. It is similar to dynamic time warping. A lower Fréchet Distance indicates more correlation or similarity between the two trajectory curves and a higher Fréchet Distance indicates dissimilarity. This value is similar to the maximum trajectory deviation value. The highest is for the OA-SLAM as its entire trajectory is scaled up as compared to the ground truth. For QuadricSLAM, it is comparatively lower as they follow the ground truth trajectory very closely.

Chamfer distance - The estimated Chamfer Distance are **285.53 mm**, **105.03 mm** and **157.29 mm** for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. Chamfer Distance is a symmetric matrix as it computes the Euclidean distance of a point in one trajectory to the neighbouring point in the other trajectory and vice versa. The Chamfer Distance indicates the alignment distance required to make the trajectories similar. Lower Chamfer Distance indicates better similarity. The reason for OA-SLAM having the higher value is the same as explained in the Fréchet Distance.

4. Other comparison parameters

Size of map is another parameter for comparing the storage of maps. The QuadricSLAM in batch mode takes **454.4 KB** and in incremental mode takes **625.7 KB** size. The increased size of the incremental mode is due to many duplicated objects present. This map contains the camera trajectory poses and the object quadric parameters. For OA-SLAM, the size is **5.5 MB** and this is due to the storage of camera poses, keyframes, the features detected in each keyframe, the map points, the object map points and the object quadric parameters. Camera poses are not required for map representation and are only needed for quantitative evaluation with ground truth. Using the map of OA-SLAM, we can also perform relocalization with sufficient point features. Still, the overall comparison with other dense mapping strategies shows that this type of map representation is very efficient in several orders of magnitudes.

Uninitiated objects is not observed in any of the SLAMs as we are providing the ground truth bounding box. Whereas the **Duplication error** exists only in the quadricSLAM in the incremental mode for tomato_soup_can. Because there is no proper strategy for associating the same objects in different observations.

5.2 Results

5.2.1 Results on 10 scenes

The properties of each scene such as camera position and the YCB objects used are mentioned in the appendix B.

Object error metrics

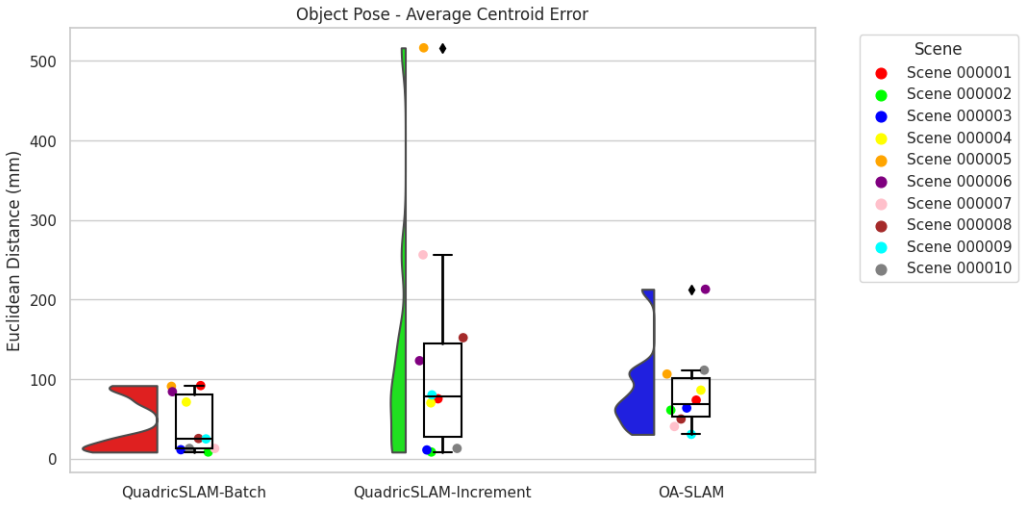

Figure 5.20: Object Centroid Error as Combined Plot

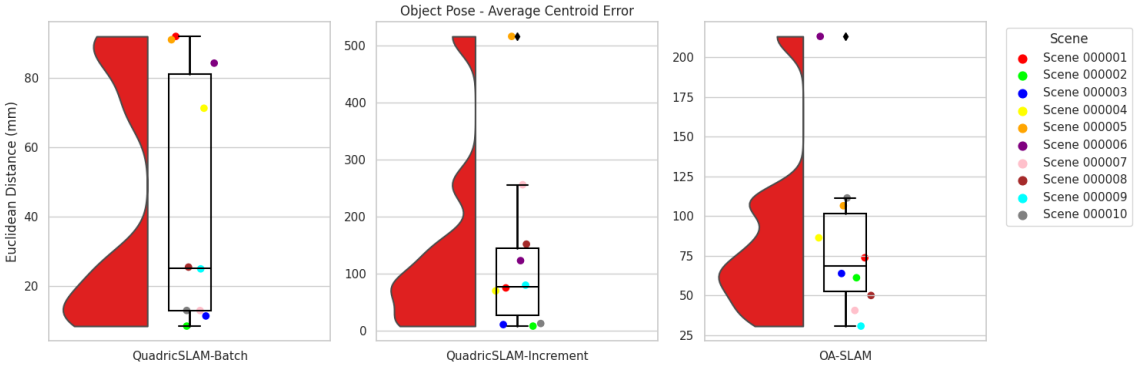

Figure 5.21: Object Centroid Error as Individual Plots

The above plots show the average centroid error for objects in all 10 scenes. QuadircSLAM in batch mode has the least error as it has the exact depth information from the depth image. OA-SLAM also has a similar error and is still maxed at around 100 mm. For QuadricSLAM in incremental mode, one or more objects may be unassociated and optimized and thereby its error affects the overall average error for the scene.

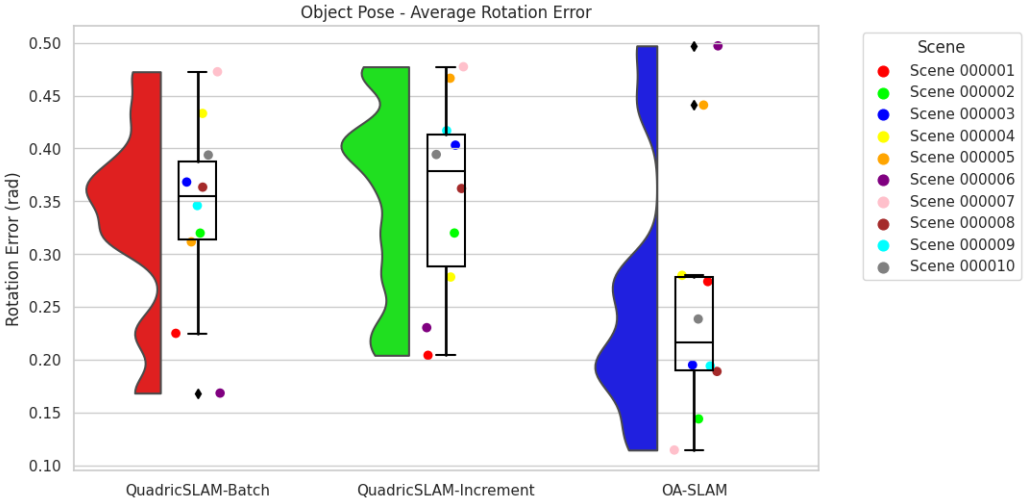

Figure 5.22: Object Rotation Error as Combined Plot

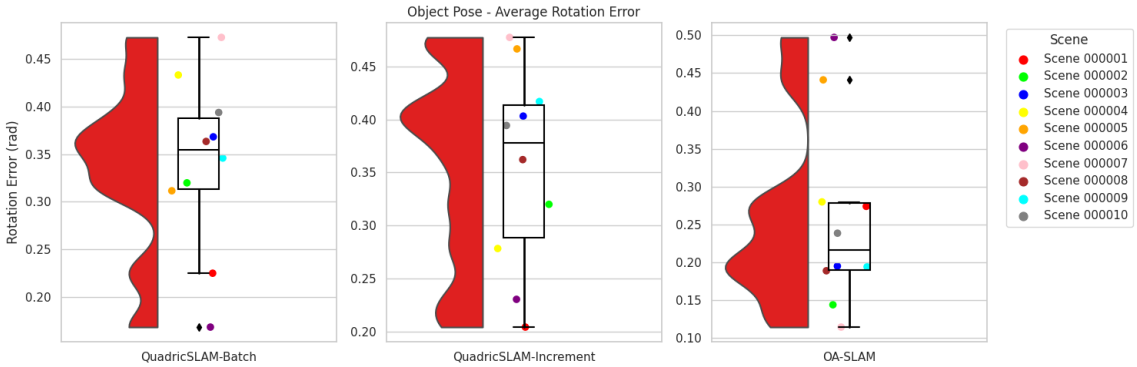

Figure 5.23: Object Rotation Error as Individual Plots

The above figure is plotted for the average rotation error for the objects. OA-SLAM was more successful in estimating the orientation of the objects and having an average error of around 0.22 rad. QuadricSLAM in batch mode had lower errors as compared to its incremental mode. The average error for both of them lies between 0.35 and 0.4 radians.

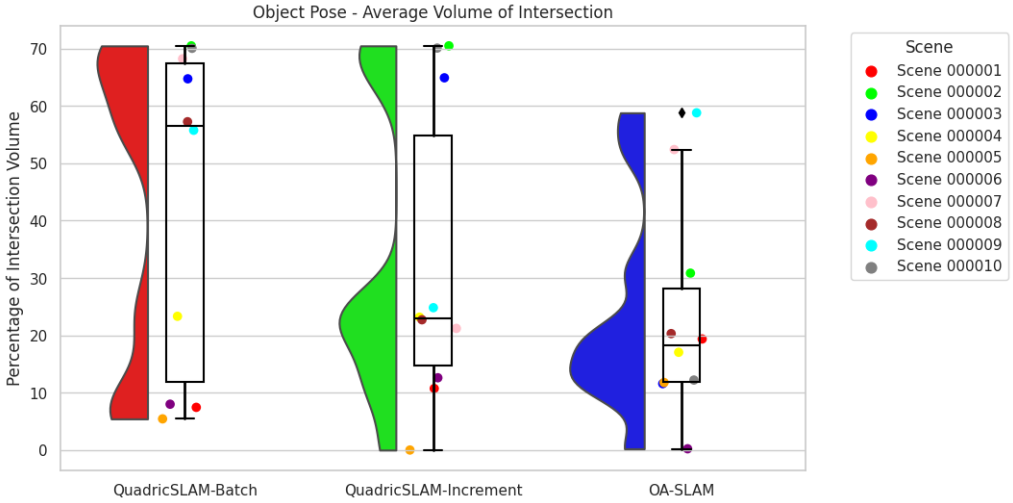

Figure 5.24: Object Overlap Percentage as Combined Plot

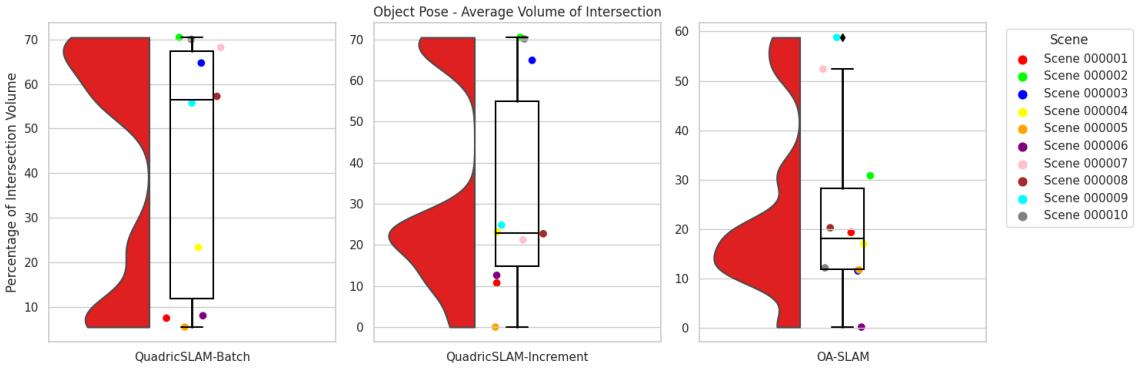

Figure 5.25: Object Overlap Percentage as Individual Plots

In the case of object overlap percentage, QuadricSLAM in batch mode had the highest overlapping at around 55% which is followed by its incremental mode at 23%. OA-SLAM is the lowest at around 19%. This lower value of OA-SLAM is due to the alignment issue when we perform scaling. For some scenes like 000001 and 000005, OA-SLAM had better performance. For QuadricSLAM in increment mode, it was not able to map any of the 5 objects properly resulting in a 0% overlap for scene 000005.

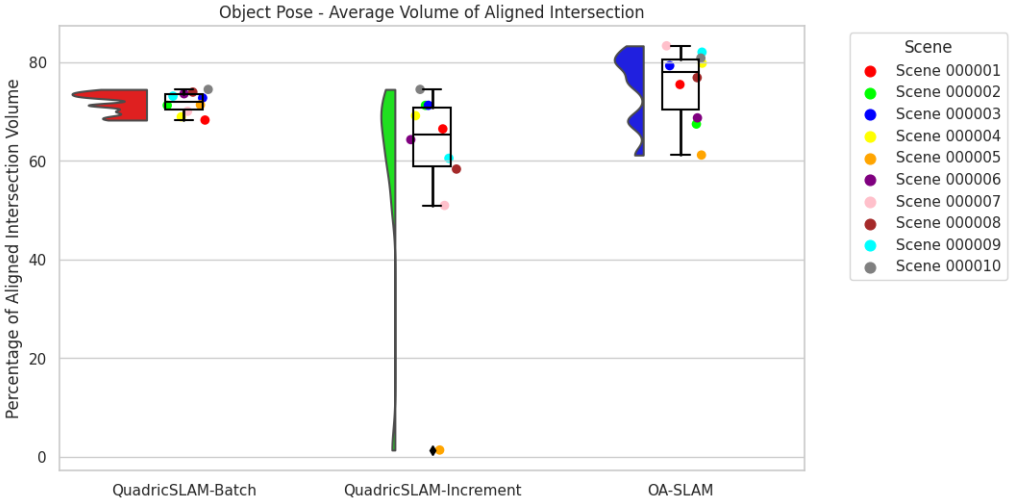

Figure 5.26: Object Aligned Overlap Percentage as Combined Plot

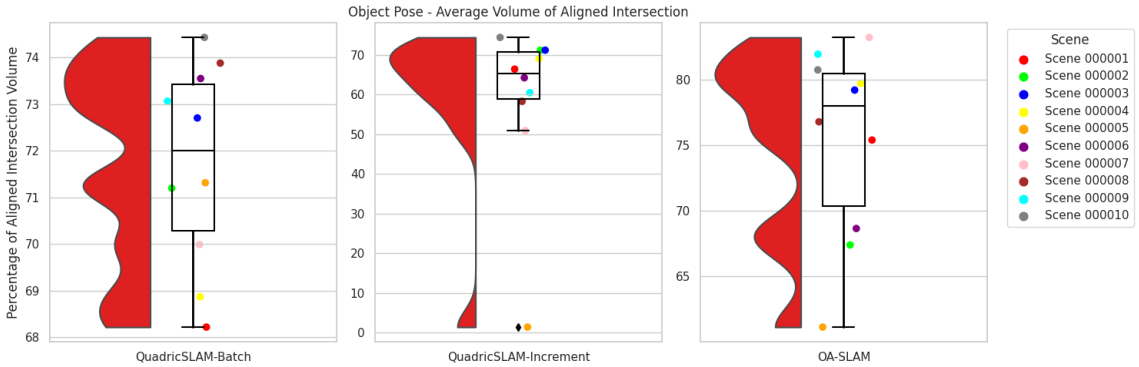

Figure 5.27: Object Aligned Overlap Percentage as Individual Plots

Due to the alignment issue in OA-SLAM, we are computing the overlap percentage after the exact alignment of each object. Now the tables have turned and the OA-SLAM reconstructed the object size most accurately at an average value of 78%. This is followed by QuadricSLAM in batch mode at 70% and increment mode at 65%.

Trajectory error metrics

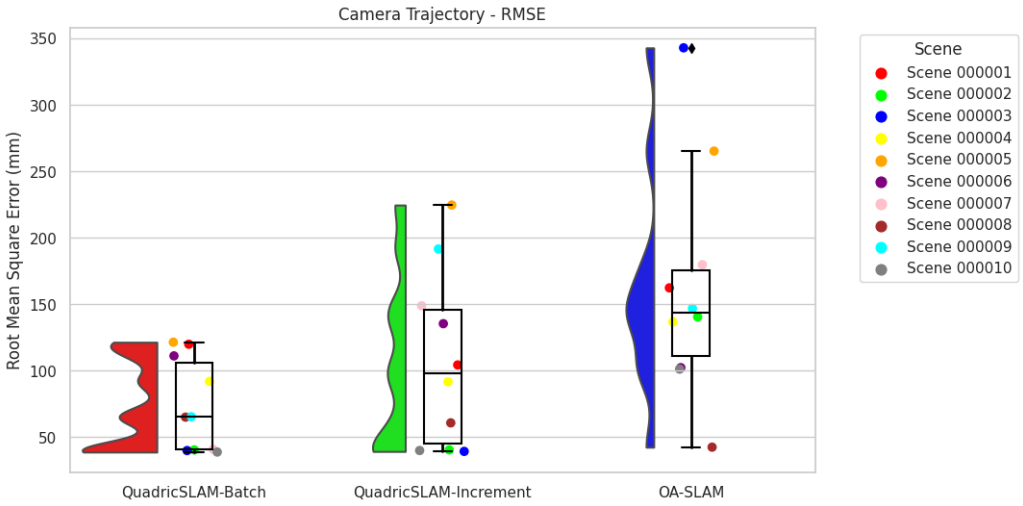

Figure 5.28: Trajectory RMSE Error Comparison as Combined Plot

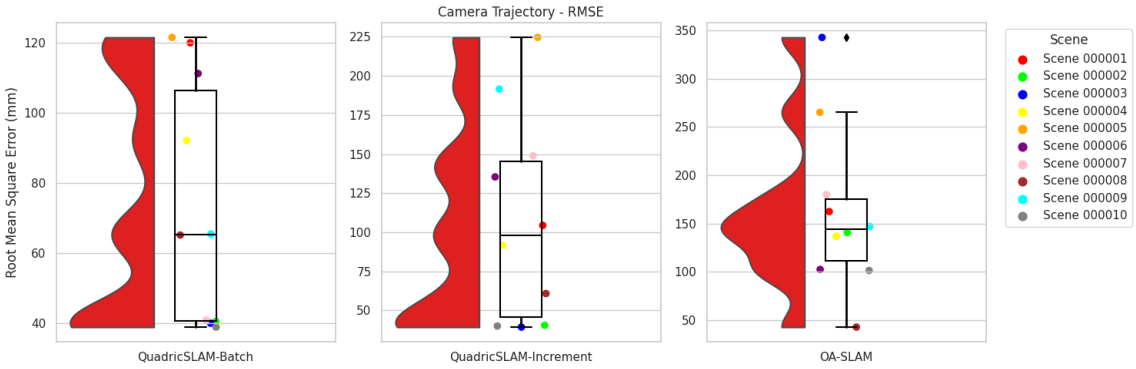

Figure 5.29: Trajectory RMSE Error Comparison as Individual Plots

The above figure indicates that the root mean square error for the OA-SLAM is the highest. This is due to the constant scaling error bias of the trajectory. For QuadricSLAM in both modes, the errors rise from 0 to max and come down causing a net RMSE to be a lower value.

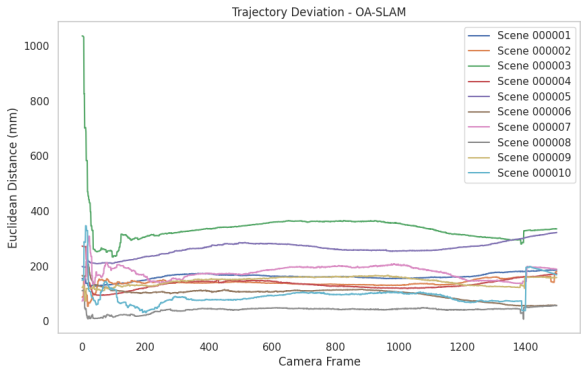

Figure 5.30: Trajectory Error Comparison - OASLAM

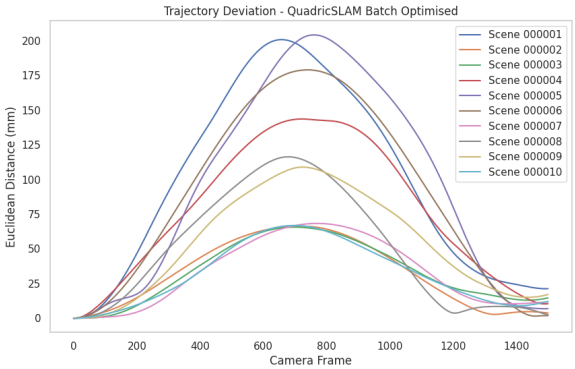

Figure 5.31: Trajectory Error Comparison - QuadricSLAM(batch)

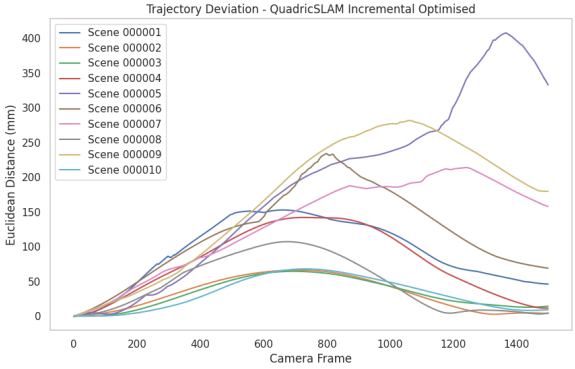

Figure 5.32: Trajectory Error Comparison - QuadricSLAM(incremental)

The above three plots indicate the trajectory deviation error for each SLAM in all the scenes. All the plots show that, given a SLAM algorithm, the pattern of the error is the same. Only the max value changes. In OA-SLAM, scene 000003 had the highest error and this was the scene where the camera

height was lowest at 300 mm. For QuadricSLAM in both modes, scene 000005 gave the highest error. In this scene, most of the objects were small and horizontally placed on the scene.

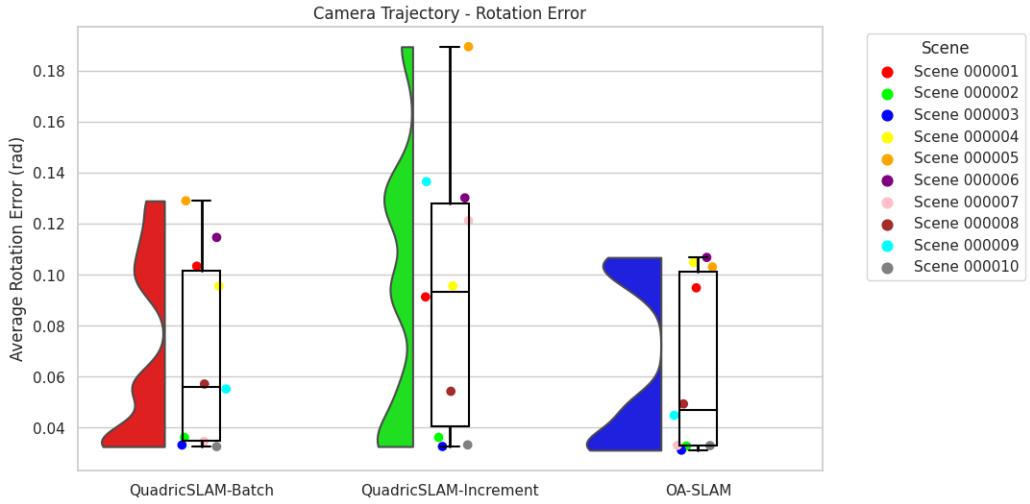

Figure 5.33: Average Rotation Error Comparison as Combined Plot

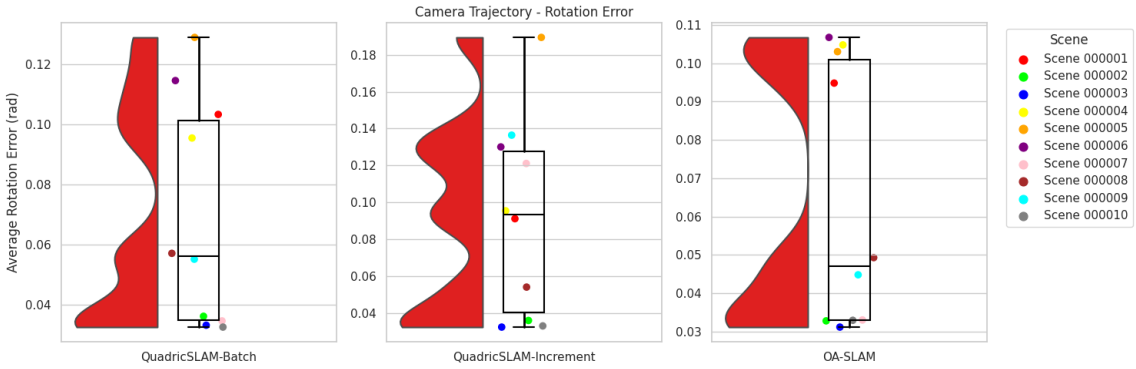

Figure 5.34: Average Rotation Error Comparison as Individual Plots

The above plots indicate that the OA-SLAM had the OA-SLAM had the least camera trajectory rotation error. And the value is also similar for QuadricSLAM in batch mode. In the scenes like 000002 and 000003, the rotation error was low for all the 3 SLAM modes and in these scenes, most of the objects were placed in the vertical pose.

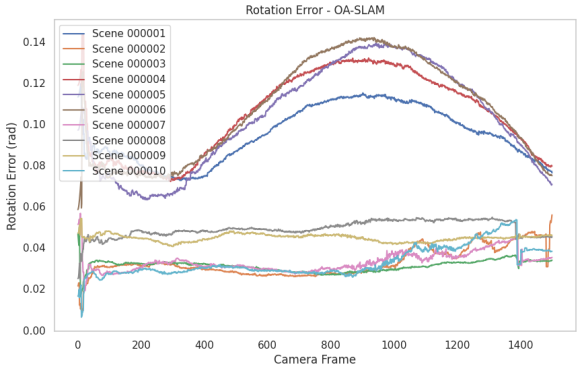

Figure 5.35: Rotation Error Comparison - OASLAM

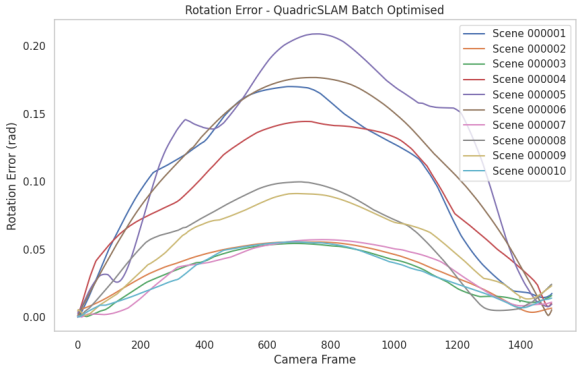

Figure 5.36: Rotation Error Comparison - QuadricSLAM(batch)

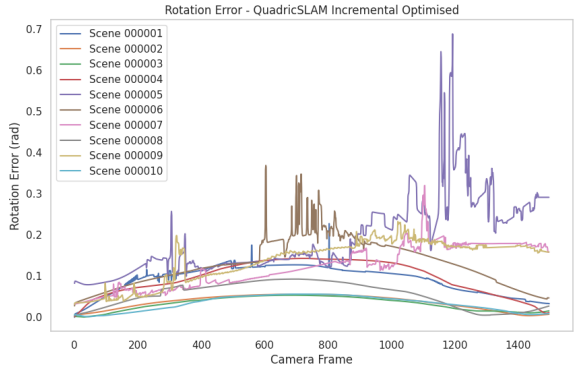

Figure 5.37: Rotation Error Comparison - QuadricSLAM(incremental)

The rotation error for each frame in each scene is plotted for the 3 SLAM modes. The plots are similar to the trajectory plots.

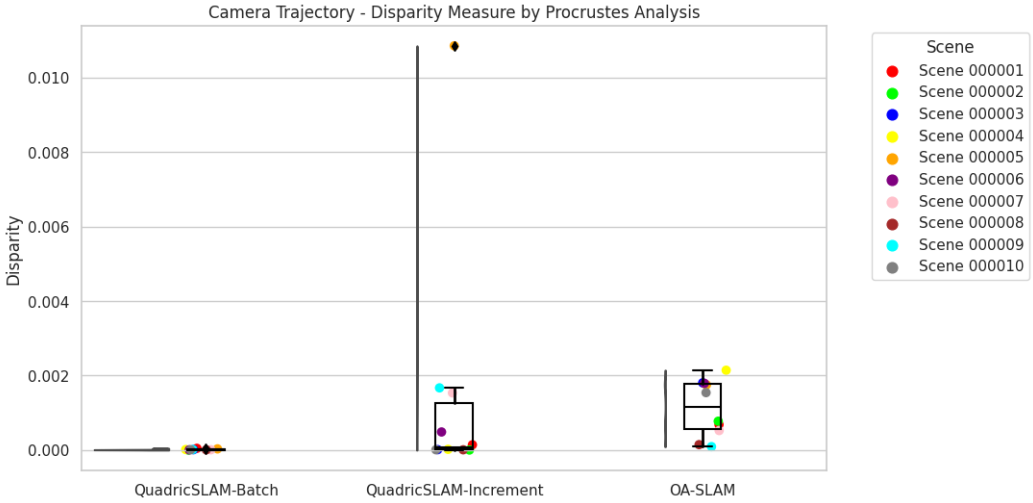

Figure 5.38: Procrustes Analysis Comparison as Combined Plot

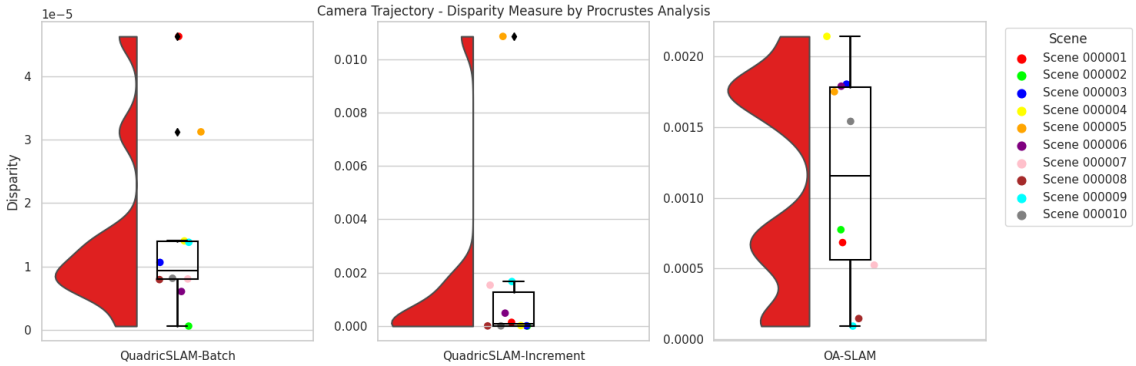

Figure 5.39: Procrustes Analysis Comparison as Individual Plots

The above plots are for the disparity measure by Procrustes analysis. OA-SLAM had higher dissimilarity. This is because, the initial camera initialization process takes some frames and during this time, the camera pose estimated is very erroneous. The trajectory plotted in these areas is not similar to a circular trajectory thereby causing a higher dissimilarity. Since, in QuadricSLAM, the odometry is provided as input, even though there are errors, it won't make the trajectory deviate heavily from the circular pattern and thereby they have a lower dissimilarity score.

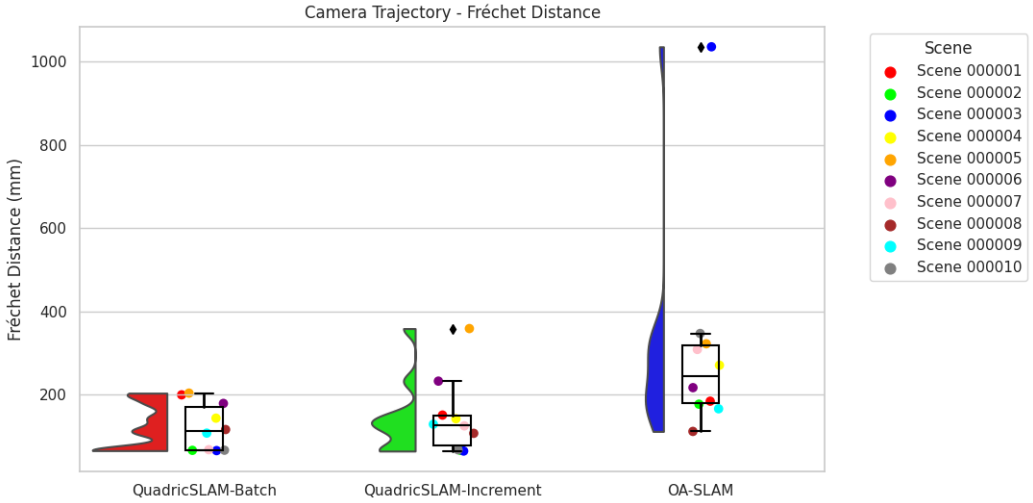

Figure 5.40: Fréchet Distance Comparison as Combined Plot

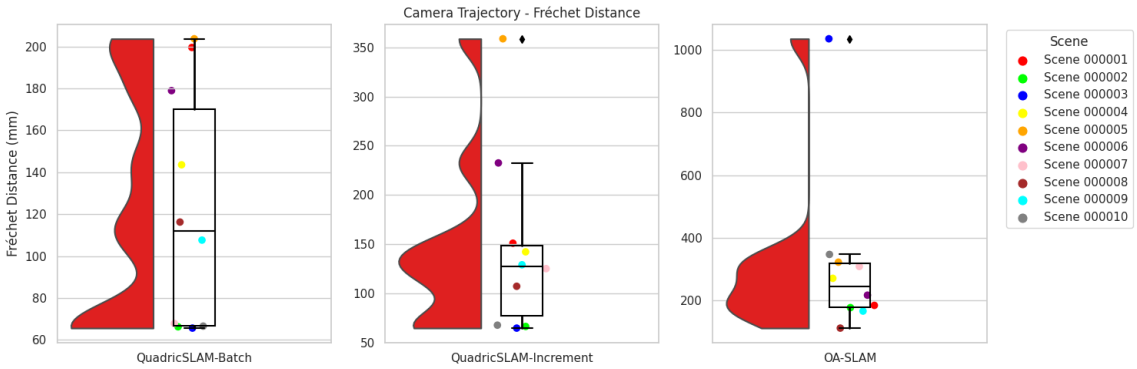

Figure 5.41: Fréchet Distance Comparison as Individual Plots

The above plots for Fréchet Distance also indicate similar results as that of the disparity score. OA-SLAM's error is affected by the initial trajectory and also the scaling bias error. QuadricSLAM in both modes have similar results.

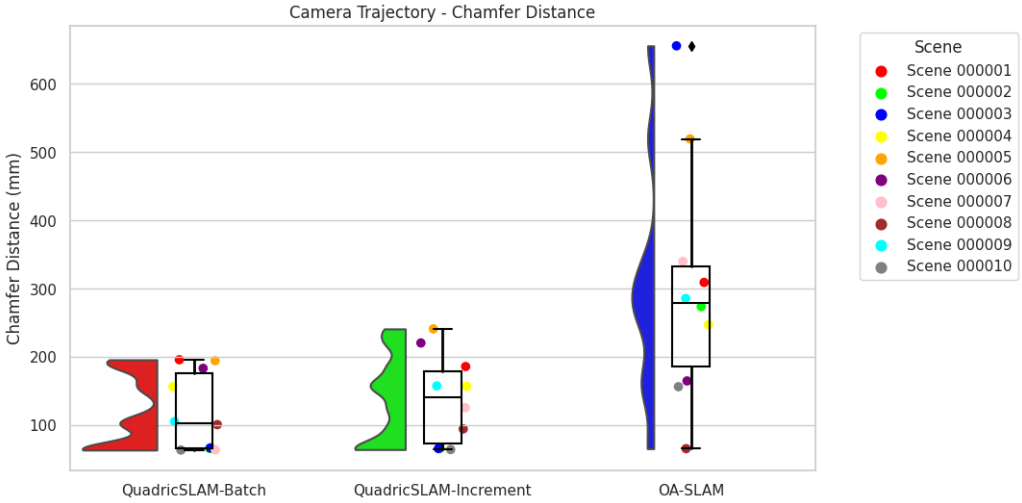

Figure 5.42: Chamfer Distance Comparison as Combined Plot

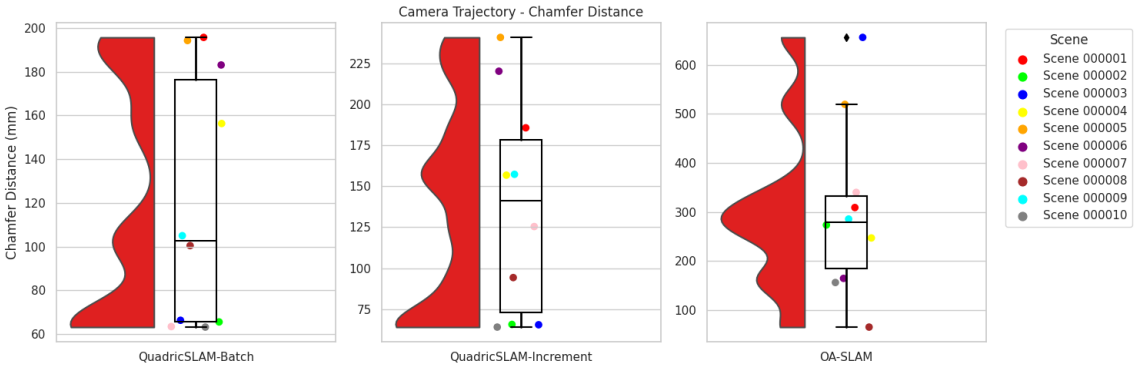

Figure 5.43: Chamfer Distance Comparison as Individual Plots

The above plots for Chamfer Distance also indicate the same results. The distance score for OA-SLAM is almost twofold as that of QuadricSLAM. The QuadricSLAM have a similar distance of 100 mm and 150 mm for batch and incremental modes.

5.2.2 Result on usage of noisy bounding box

The same scene 000009 that is described in the section 5.1 is corrupted with observation noise. This is induced by varying the centre of the bounding box, the width and height of the bounding box and even removing the bounding box for 1 of the object in each frame randomly. The results are plotted below.

OA-SLAM OUTPUT

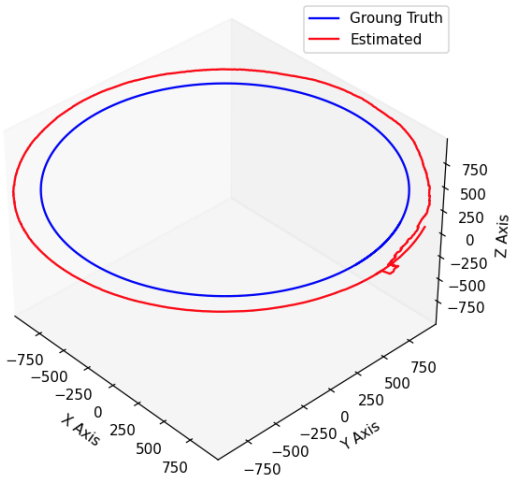

Figure 5.44: Estimated Trajectory

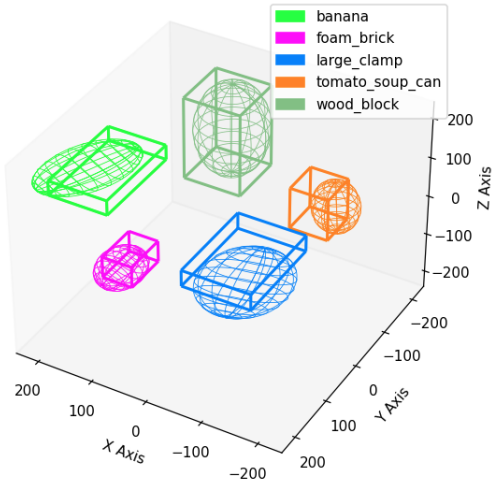

Figure 5.45: Estimated Objects

As compared to the trajectory plot 5.3 and object plot 5.4 for the scene without noise, the results are very much exactly the same. This indicates the robustness of OA-SLAM against the noise in the bounding boxes. This is also due to the fact that it relies heavily on the appoint reprojection errors for mapping the environment rather than the object observations only. This proves that OA-SLAM can be used in real-world applications with a high level of uncertainty.

QuadricSLAM OUTPUT - BATCH OPTIMIZATION

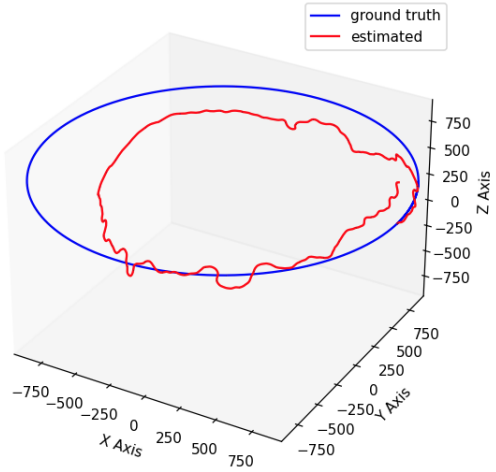

Figure 5.46: Estimated Trajectory

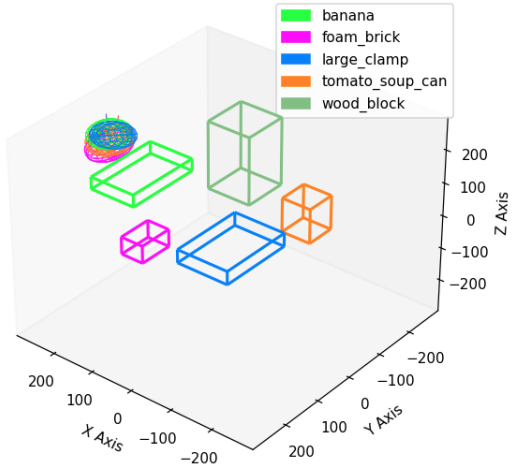

Figure 5.47: Estimated Objects

In the case of QuadricSLAM in batch optimization mode, it heavily suffered from the noises in the

bounding boxes. This is due to their high dependence on the object for joint optimization thereby injecting the errors into the trajectory estimation. As compared to the non-noisy scene estimated trajectory5.5, here the entire trajectory was contracted towards inside the ground truth trajectory. Assuming the trajectory as a string with constant length, to compensate for the wriggles in the string caused by the errors in the object observations, the circumference covered by the string is reduced. The length of the estimated trajectory is determined by the accumulated odometry measure between each camera pose node in the factor graph.

When we look into the object map, as compared to the non-noisy one5.6, here the entire object models fell apart and converged to some random location that is far apart from their ground truth location. These observations indicate the non-robust behaviour of QuadricSLAM in batch optimization mode in real-world scenarios.

QuadricSLAM OUTPUT - INCREMENTAL OPTIMIZATION

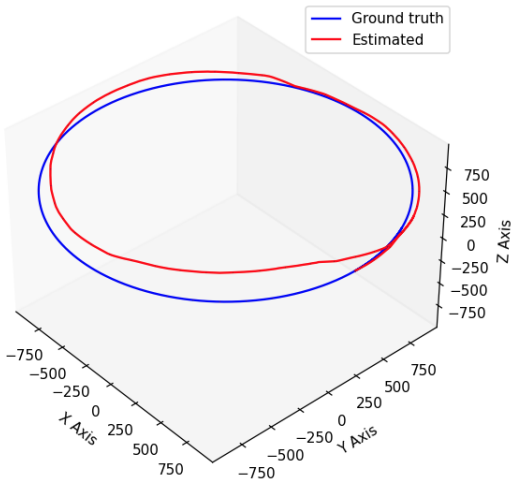

Figure 5.48: Estimated Trajectory

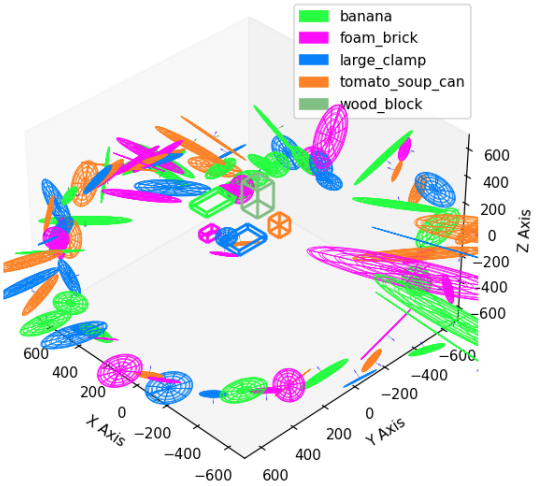

Figure 5.49: Estimated Objects

Even though the estimated trajectory is very erroneous as compared to its non-noisy counterpart5.7, when compared with the QuadricSLAM in batch mode operating on noisy observations, here the trajectory is much better. Since the optimization is performed in each iteration, the trajectory is not much wriggled as all the constraints are not applied together but rather one by one. Whereas the object plot is very erroneous as compared to its non-noisy version5.8. Here none of the objects are properly associated causing all the individual detections to be initialized as new instances of objects within the map. These results show the incapability of QuadricSLAM towards noisy environments.

EVALUATION

1. Object mapping comparison

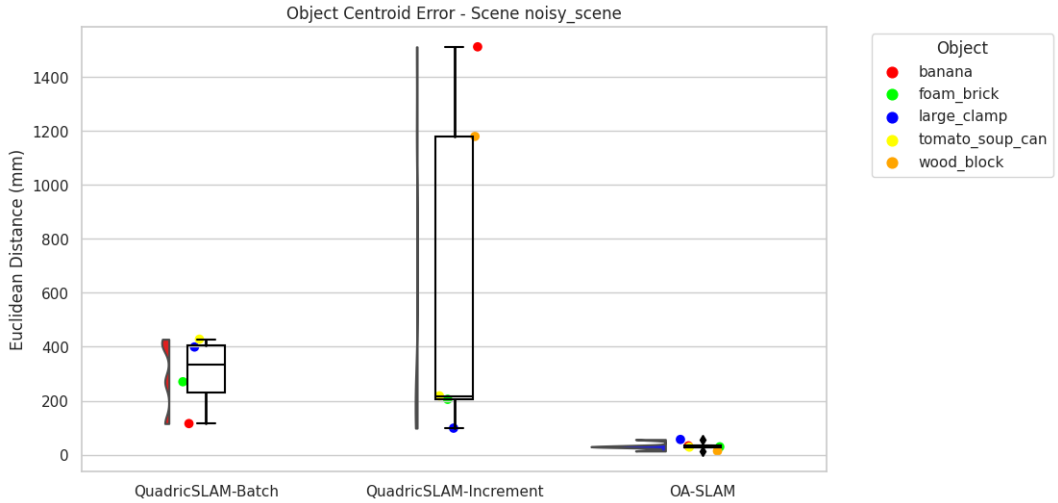

Figure 5.50: Object Centroid Error as Combined Plot

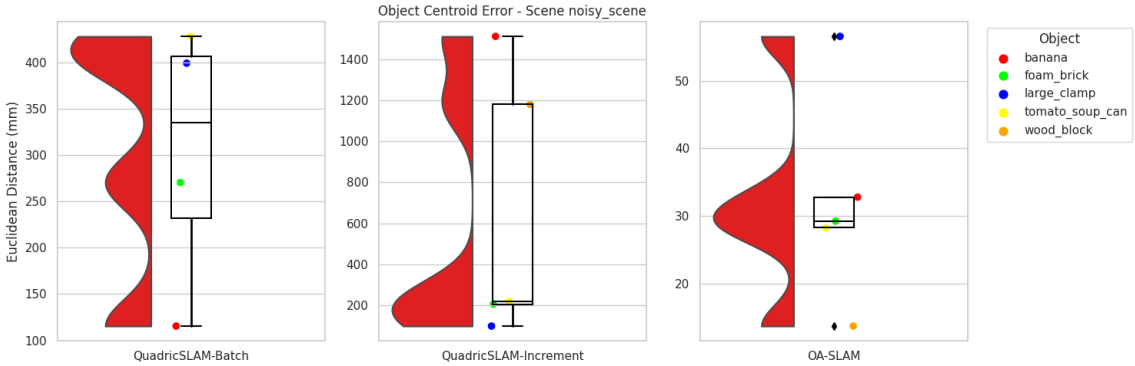

Figure 5.51: Object Centroid Error as Individual Plots

In the plot5.50, it can be seen that the QuadricSLAM in incremental mode has the highest object centroid error followed by the QuadricSLAM in batch mode and OA-SLAM has the least error. Even though the objects are not properly associated and optimized in QuadricSLAM in incremental mode, the object that is closest to the ground truth is considered for comparison. In the non-noisy scene, as seen in figure5.9, QuadricSLAM in batch mode was having a lower error than OA-SLAM. But now, the case just reversed and this is due to the non-robust behaviour of QuadricSLAM towards noisy observations. When we computed the **average object centroid error** of each SLAM, for OA-SLAM, its **32.09 mm**, for QuadricSLAM in batch mode **303.28 mm** and for the incremental mode, its **642.65 mm**. OA-SLAM

had very similar values as before whereas the QuadricSLAM had errors by a far higher magnitude.

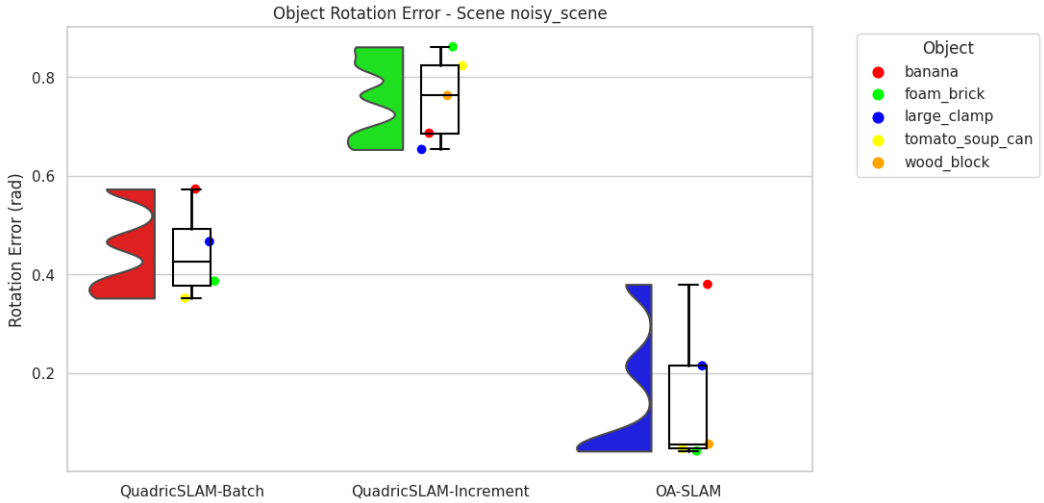

Figure 5.52: Object Rotation Error as Combined Plot

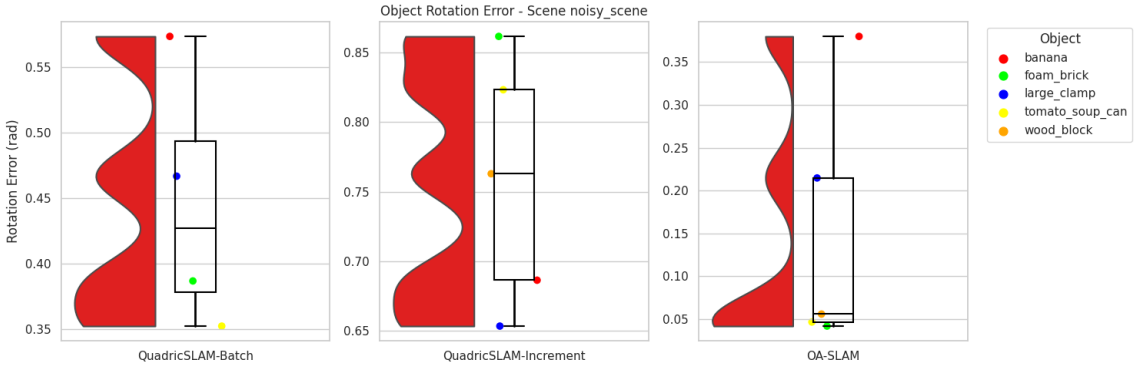

Figure 5.53: Object Rotation Error as Individual Plots

The average rotation error is **0.15 rad, 0.45 rad, 0.76 rad** for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. In the case of OA-SLAM, the average error reduced as compared to its non-noisy counterpart whereas it increased for QuadricSLAM in both modes. In both noisy and non-noisy cases, banana still exhibited high rotation error. The maximum error didn't cross 0.4 rad for OA-SLAM.

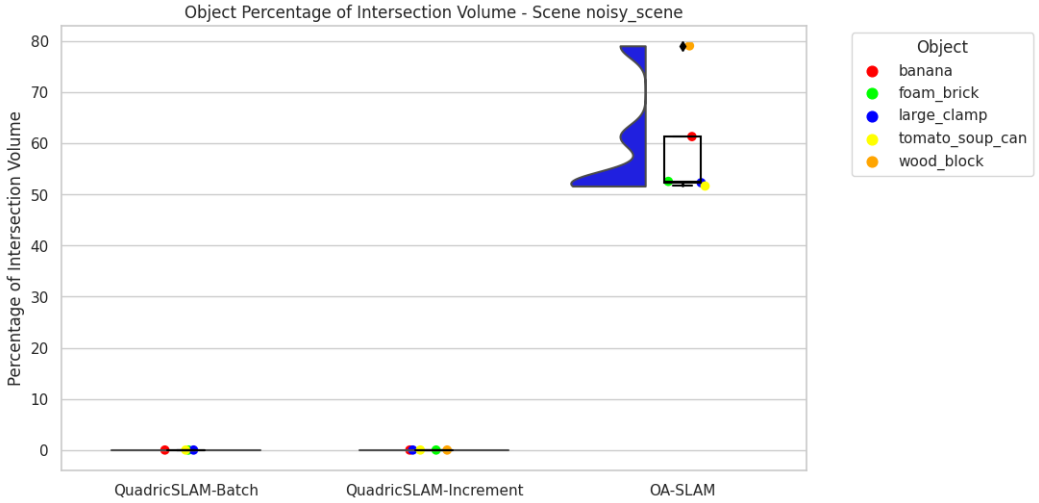

Figure 5.54: Object Overlap Percentage as Combined Plot

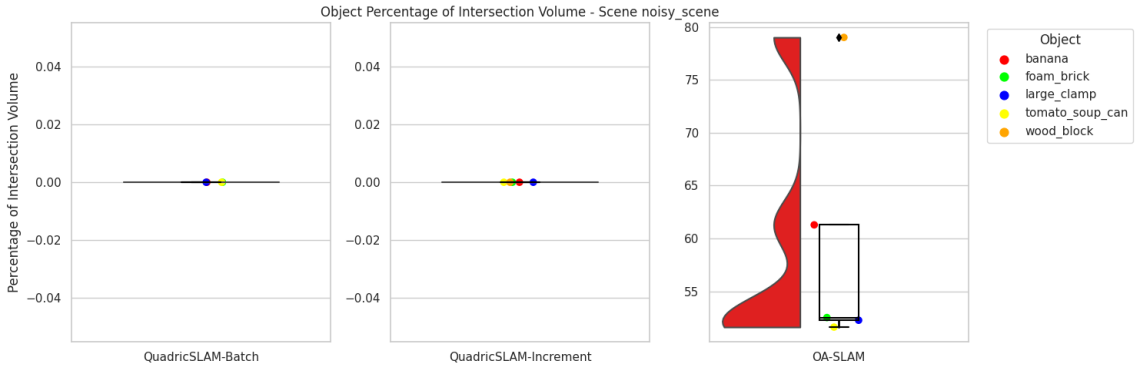

Figure 5.55: Object Overlap Percentage as Individual Plots

59.34%, **0%** and **0%** are the average overlap percentage for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. There is clearly no overlap for any of the objects in both the modes of the QuadricSLAM. The OA-SLAM gave a slightly better result than its non-noisy counterpart. As discussed before, bigger objects like wood block had the best overlappings.

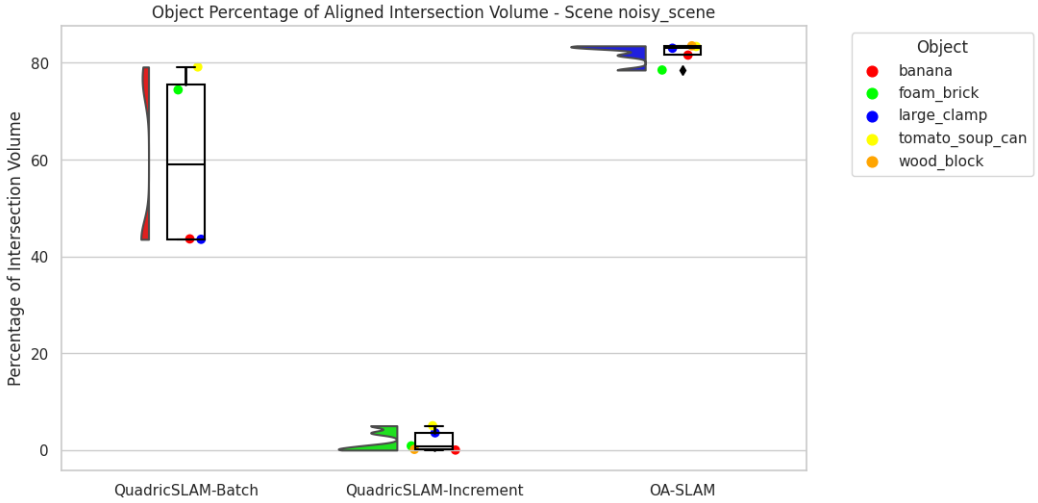

Figure 5.56: Object Aligned Overlap Percentage as Combined Plot

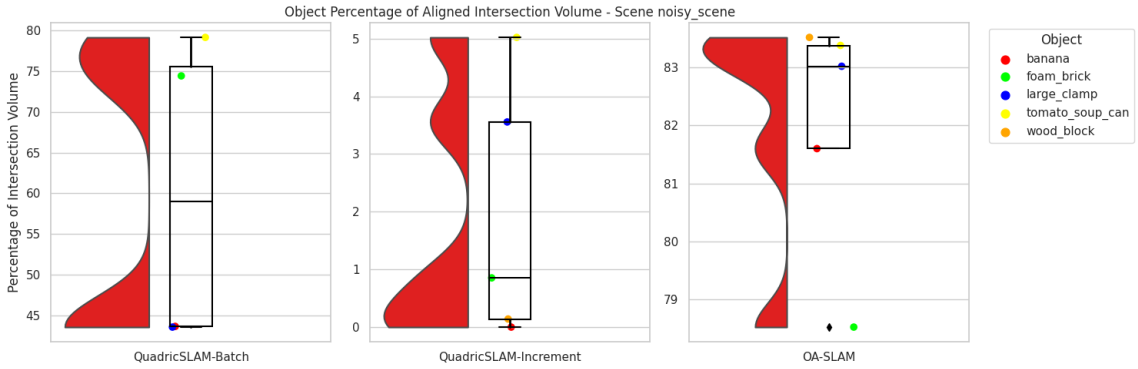

Figure 5.57: Object Aligned Overlap Percentage as Individual Plots

82.01%, **60.18%** and **1.92%** are the average overlap aligned percentages for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively.

The OA-SLAM had similar results as before. The result of QuadricSLAM in batch mode indicates that it was able to reconstruct the shape to some extent, but since the centroid of the object was varied, it could achieve overlapping without alignment. For QuadricSLAM in increment mode, it shows that some of the object detections were able to associate together which gave a really bad reconstruction.

2. Camera trajectory comparison

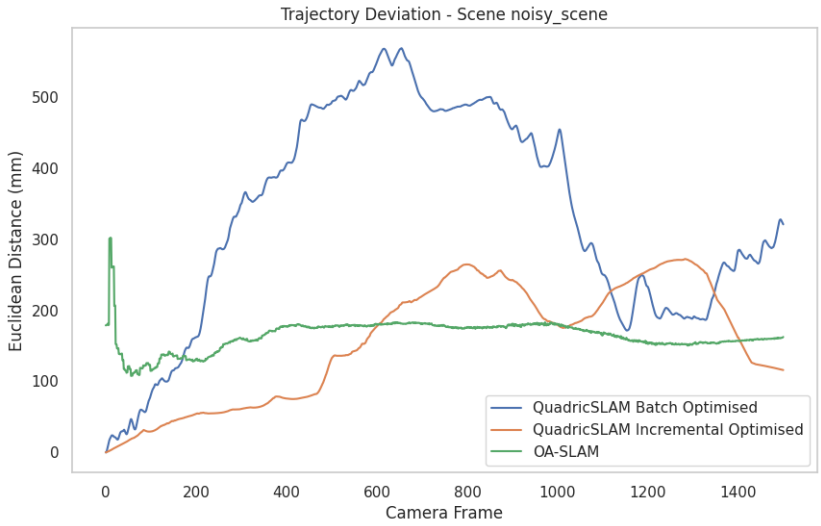

Figure 5.58: Trajectory Error Comparison

165.16 mm, **365.18 mm** and **178.18 mm** are the root mean square error for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. In contrast to the non-noisy scene trajectory error plot5.18, here the QuadircSLAM in incremental mode has lower errors than the batch mode. OA-SLAM had slightly more error as compared to its non-noisy counterpart. As seen before, the OA-SLAM has almost constant errors. QuadricSLAM in batch mode performed the worst as the whole trajectory collapsed causing the error to peak more than **500 mm**.

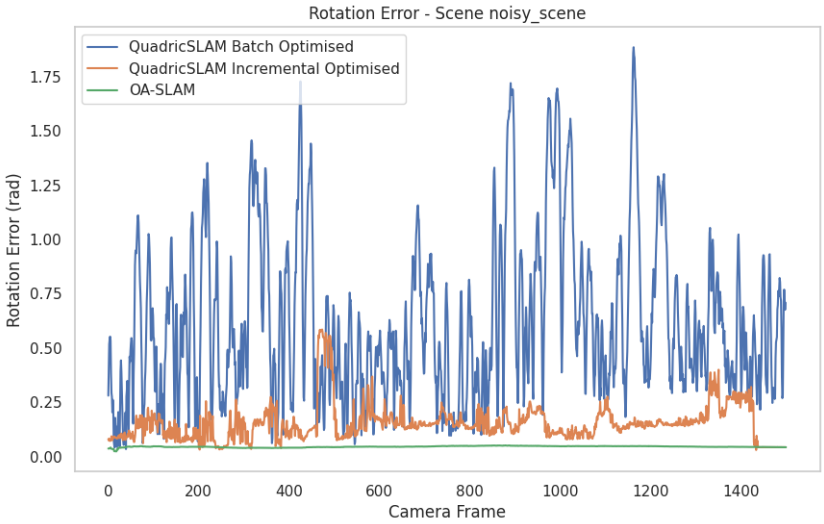

Figure 5.59: Rotation Error Comparison

0.05 rad, **0.62 rad** and **0.15 rad** are the average trajectory rotation error for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. OA-SLAM had the least error among all. Both OA-SLAM and QuadricSLAM in incremental mode had similar errors as compared to their non-noisy counterpart. But in the case of QuadricSLAM-batch mode, the collapsed trajectory gave a very noisy error which even crosses **0.5 rad**.

Procrustes analysis - The estimated disparity value are $\mathbf{2.07}e^{-4}$, $\mathbf{2.71}e^{-2}$ and $\mathbf{2.10}e^{-3}$ for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. The lowest is for OA-SLAM followed by QuadricSLAM in incremental mode. The batch mode trajectory is highly deviated from the ground truth resulting in a higher disparity.

Fréchet distance - The estimated Fréchet Distance are **299.35 mm**, **566.55 mm** and **262.21 mm** for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. Fréchet Distance is in general higher for all the SLAMs as compared to non-noisy counterparts. The higher value for QuadricSLAM in batch mode is due to the collapse of the trajectory. Since the QuadricSLAM in incremental mode followed the trajectory closely, it had a lower error as compared to OA-SLAM. But the expense of poor object mapping.

Chamfer distance - The estimated Chamfer Distance are **322.24 mm**, **520.32 mm** and **275.90 mm** for OA-SLAM, QuadricSLAM-batch and QuadricSLAM-incremental respectively. The pattern of error is similar to that of Fréchet distance

5.2.3 Result on localization task

To test the relocalization capability of object-oriented SLAM, the localization mode in OA-SLAM is utilized. There are two localization modes: one using only points and the other one utilizing both objects and points. For the points-only localization mode, the current observation needs to be searched using the bag-of-words approach to identify the most probable location. Whereas in the objects+points mode, if the bag-of-words approach couldn't identify the exact pose, then the knowledge of the objects is utilized to perform 2D-3D correspondence(2D information from the query image and 3D information from the available map) using the PnP algorithm. To test the performance, we have mapped the environment scene_000009 using all the 1500 images to create a complete map called Full map and also created a map using the first 750 of the images called Half map. In both the maps, we are performing both the localization modes using either a single query image and using two query images. The image which we selected was 001071.png which belonged to the unmapped region. This image can be seen in the figure5.60 below. This particular image is selected because it is identified as a keyframe by the OA-SLAM during mapping and only keyframes are available in the visualization mode. So, when we perform localization, we can visualize and see whether the estimated pose is overlapping the keyframe pose for 001071.png.

Figure 5.60: Query image from scene 000009 for localization

Localization capability on full map

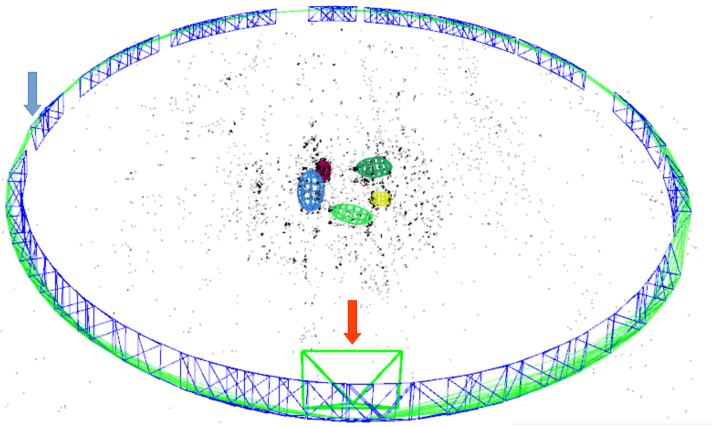

Figure 5.61: Localization using only points on the single query image

The blue arrow is the ground truth frame pose and the red arrow is the estimated pose. All the blue frustum indicates the keyframes stored by the map of OA-SLAM. The object and the corresponding colours are blue - wooden block, light green - banana, brown - tomato soup can and dark green - wrench. The yellow object is foam brick which is not important in this case as it is occluded in the query image. The green bigger frustum is the estimated camera pose as a result of relocalization.

In the above image5.61, the localization mode was using only points using the query image 001071.png. We can see that the localization is erroneous or in other words, localization was not possible. This maybe due to the fact that since it is a monocular RGB image, it needs at least 2 images to perform triangulation and the bag-of-words-based localization can be made by finding the correspondence between the keypoints in the image and the mappoints in the map. To test this, the nearby image 001070.png is also passed along with the 001071.png as query images and the localization was successful as seen in the figure5.62

below.

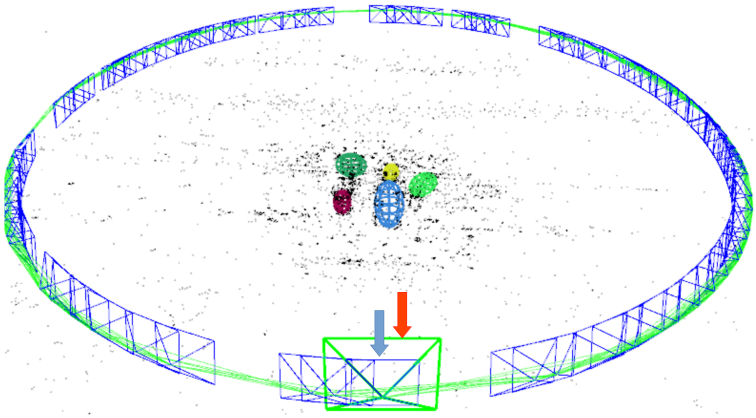

Figure 5.62: Localization using only points on the query image + 1 nearby image

Now, we can test the performance of the object+points mode. When we passed only a single query image 001071.png, points were not able to perform localization which triggered the object-based localization mode. Using the PnP algorithm, multiple hypotheses for the poses were generated and in each of the poses, 3D to 2D projection of the ellipsoid to ellipse in the images were done. The IoU between the bounding box in the query image and this reprojection is used to estimate the most valid pose. As a result, the relocalization was able to be performed using a single image itself as seen in the figure5.63 below.

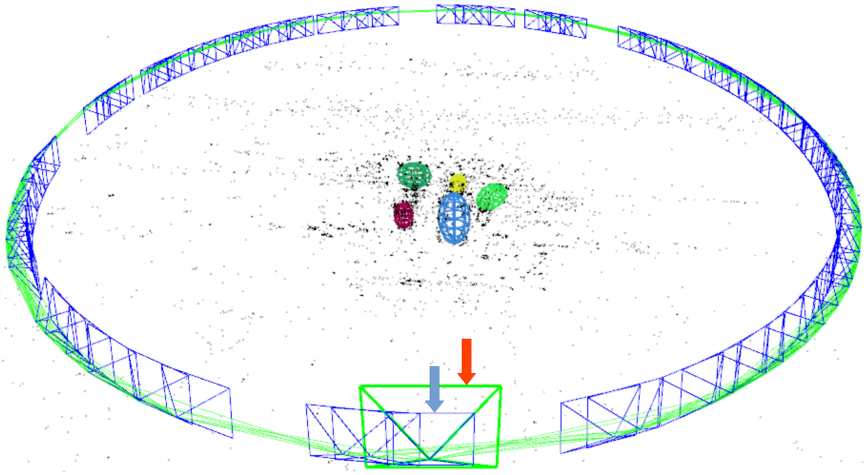

Figure 5.63: Localization using points+objects on the single query image

Localization capability on half map

Now, another test is performed where we map only half of the environment and use a query image from the unseen part of the environment to check if relocalization is possible. In the below figure5.64, the

localization was performed in points mode with 10 query images(001071.png and 9 frames before that). Still, the localization was not possible as it couldn't match any points with the bag-of-words approach as it is an unseen part of the world. Even some of the points that were mapped in the first half are not visible in the current viewpoint of the query image and some points are occluded by the objects. Hence, relocalization was not possible with only points in the unseen part of the scene.

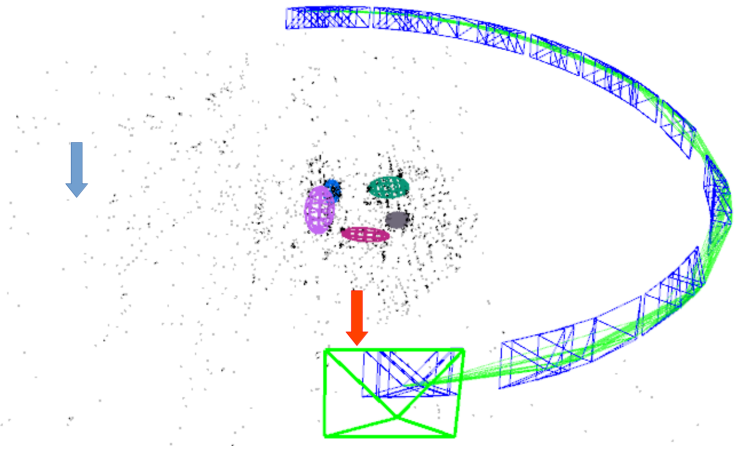

Figure 5.64: Localization using points on the query image + 9 nearby images

When the objects+points mode is enabled, this problem doesn't exist and the relocalization was possible with the single query image 001071.png as seen in the figure5.65 below. This is due to the fact that the objects are not much occluded as compared to points and objects can be viewed from any viewpoint as compared to feature points which are visible only within a certain viewing cone. This experiment highlights the importance of objects in aiding the localization process and the tracking was also possible by passing more query images into the system.

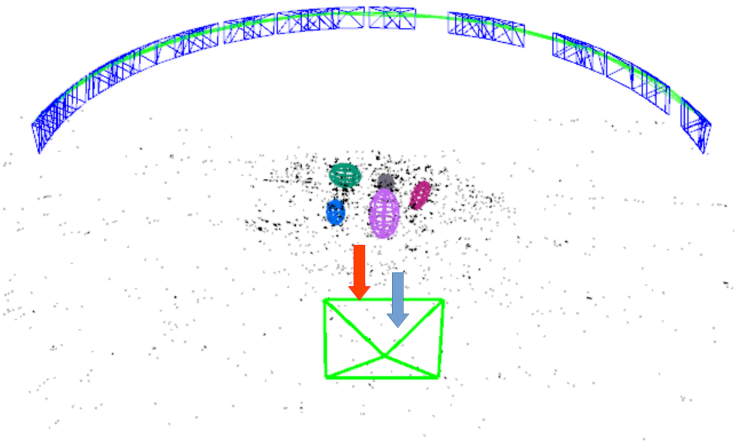

Figure 5.65: Localization using points+objects on the single query image

6

Conclusions

6.1 Contributions

This project aimed to compare the performance capabilities of semantic object-oriented SLAM methods such as QuadricSLAM and OA-SLAM. We created a dataset consisting of 10 scenes in BOP format which can be used for quantitative analysis of the mapped object poses and shapes. We tested both the SLAM algorithms on these datasets and defined evaluation metrics for quantitative analysis of the camera trajectory and the objects. Also, we modified a scene to have noisy bounding boxes and were able to evaluate its effect on both the SLAM algorithms. Finally, the localization capability of objects was tested and proved in a scene where query images from the unseen part of the mapped environment were used. The experiments help to identify the areas of improvement and propose some suggestions for future work. Also, a short survey into the history of 2D and 3D SLAM was done to give an idea of what led to the use of coarse-object models in sparse maps.

6.2 Lessons learned

An SLAM algorithm with the representation of semantic objects along with a sparse map performs better relocalization from different viewpoints than an SLAM algorithm with non-semantic sparse points. This is due to the property of the bigger size of the object and they could be viewed from any direction such as top, side, front, or back. This helped OA-SLAM to achieve relocalization within 1 query image. The splitting of tracking and mapping on separate threads helps to achieve real-time operation capability as the continuous input images are not being blocked from processing because the expensive optimization step is moved to another thread. Culling of non-robust map points and redundant keyframes helps in maintaining a smaller robust map for faster optimization steps and efficient storage. Sparse maps are much more efficient in the localization process as compared to dense maps given a limited memory capability. A tabletop scene can be mapped with around 5.5MB using OA-SLAM which could have gone above 100MB using dense point cloud maps. The map generated by OA-SLAM is more accurate than QuadricSLAM because the feature points act as a stable anchor during the mapping process and the objects can be reliable only after the accurate mapping of the object. Until an object is fully mapped, its pose can very rapidly between multiple frames and this caused the errors in QuadricSLAM as it relied only on the objects and rather neglected the point features. A common issue with monocular camera-based SLAM

which is scale drift error can be seen in OA-SLAM also. It could be fixed either by placing an object of known size while it starts operation to initialize the scale as in MonoSLAM [21] or a depth camera can be used. If a depth camera is used, then rigid transformations can be used instead of similarity transformations in the SLAM framework which can fasten the operation cycle.

Even though the camera trajectory estimated by OA-SLAM has some errors, it is only due to the constant error bias of the scaling. The plots showed that the shape of the trajectory is exact except for the scale. This is also a reason why the disparity is low for OA-SLAM when analysed using Procrustes analysis. Another takeaway is that, the more the number of constraints in the factor graph, the more accurate the map would be. OA-SLAM has point and object factors which can also include plane factors as in structure-aware SLAM [46] which can improve the mapping accuracy. In all these SLAM approaches, the observations and the camera poses are jointly optimised. Data association is solved to an extent using the Hungarian algorithm which is utilised in OA-SLAM. However, the invalid class predicted by the YOLO for a constant period of time can create another object model at the same place as the correct object model which visually appears as 1 ellipsoid inside another ellipsoid. This observation was seen in OA-SLAM and there should be a mechanism to reject one object model if one is present in another object. As graph-based SLAM approaches rely heavily on the loop closure for the full graph optimizations, the loop closure candidate should be accurate. If it is not the correct one, it can induce additional errors in the framework. So an inlier checking mechanism is implemented in OA-SLAM which can reject false positive loop closure candidates. The tracking thread of OA-SLAM has a smaller graph that can perform joint optimizations in a fast real-time manner and still output an accurate pose of the camera. It uses a moving window to select the part of the map that should be present in the smaller graph.

6.3 Future work

As mentioned before, adding additional constraints can help factor graphs arrive at better estimates. Since the code for structure-aware SLAM [46] is not open-sourced, implementing the plane constraint defined in that paper in OA-SLAM would be a possible future work. Adding planes would also help to visually enrich the map generated with more information for the user who is viewing the map. The availability of planes such as walls can help in segmenting a bigger indoor environment into separate rooms and thereby a topological map can be created. This topological map can aid in global path planning. The planes such as floors, can help the robot identify a traversable path. Finally, the sparse map with points, objects and planes can be voxelised and converted into a 3D occupancy grid map for navigation. Another use case is the creation of a digital twin map in the Gazebo simulator. Since now we have planes and objects with labels and orientations, using a general CAD model for each class of objects, a digital twin can be created so that a user can easily view the map and understand the scene rather than looking into a map with ellipsoids of different colours.

Even though we were able to perform global relocalization on a tabletop scene, in a real large indoor environment, there can be multiple instances of the same objects and thereby multiple hypotheses exist for the robot's position. However, in OA-SLAM only one hypothesis with the highest score is selected. A particle filter-like mechanism could be implemented to generate particles at each of these hypotheses based

on their score and can be further converged as more distinct observations are seen. Also, the objects seen in the current view can also aid in predicting what type of room the robot is in. For example, if kitchen utensils are seen, then the most probable location is the kitchen. Another much-unexplored domain is the textual cues which can aid in localization. For example, in the university, even though all the rooms look the same, the textual cue of the class number which is inscribed near the door of the room can help in a quick relocalization. Also, other textual cues like, the direction of the exit, and the direction of the information desk can also provide more information on the current location.

The current SLAM method can only map objects that are detected by the YOLO model. If the YOLO is well-trained to classify all the objects in the world, then it would be ideal. But in reality, it detects only a limited number of classes. So to map all the objects in the environment, models like the Segment Anything Model (SAM) can be used to create the bounding box for all the unseen objects and a tracking mechanism should be implemented to associate the bounding boxes to the same object in the subsequent images. Even though their actual class name is unknown, it can be queried with the human operator or by searching the internet for the class name using the cropped images.

A

OA-SLAM Parameters

The below parameters are used in the OA-SLAM configuration.yaml file.

Parameter	**Value**
Camera.fx	888.888916015625
Camera.fy	1000.0
Camera.cx	320.0
Camera.cy	240.0
Camera.k1	0.0
Camera.k2	0.0
Camera.p1	0.0
Camera.p2	0.0
Camera.width	640
Camera.height	480
Camera.fps	30.0
Camera.bf	40.0
Camera.RGB	1
ThDepth	40.0
DepthMapFactor	5000.0
ORBextractor.nFeatures	1000
ORBextractor.scaleFactor	1.2
ORBextractor.nLevels	8
ORBextractor.iniThFAST	20
ORBextractor.minThFAST	7

B

Synthetic Dataset Configuration

The below parameters are used to generate the dataset.

Scene name	**YCB objects used**	**Camera trajectory radius**	**Camera height**
000001	003_cracker_box, 004_sugar_box, 011_banana, 021_bleach_cleanser, 052_extra_large_clamp	1.2	1.0
000002	002_master_chef_can, 003_cracker_box, 004_sugar_box, 005_tomato_soup_can, 006_mustard_bottle	1.2	0.3
000003	007_tuna_fish_can, 008_pudding_box, 009_gelatin_box, 010_potted_meat_can, 011_banana	1.2	0.3
000004	019_pitcher_base, 021_bleach_cleanser, 024_bowl, 025_mug, 035_power_drill	0.17	0.9
000005	036_wood_block, 037_scissors, 040_large_marker, 051_large_clamp, 052_extra_large_clamp	1.0	0.9
000006	061_foam_brick, 004_sugar_box, 007_tuna_fish_can, 010_potted_meat_can, 021_bleach_cleanser	1.0	0.9
000007	024_bowl, 036_wood_block, 051_large_clamp, 002_master_chef_can, 005_tomato_soup_can	1.2	0.3
000008	006_mustard_bottle, 010_potted_meat_can, 024_bowl, 037_scissors, 061_foam_brick	1.2	0.5
000009	051_large_clamp, 005_tomato_soup_can, 011_banana, 036_wood_block, 061_foam_brick	1.2	0.5
000010	009_gelatin_box, 035_power_drill, 061_foam_brick, 005_tomato_soup_can, 019_pitcher_base	1.2	0.5

References

[1] S. Yang and S. Scherer, "CubeSLAM: Monocular 3-D Object SLAM," *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 925–938, 2019.

[2] M. Zins, G. Simon, and M.-O. Berger, "OA-SLAM: Leveraging Objects for Camera Relocalization in Visual SLAM," in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, Oct 2022, pp. 720–728.

[3] L. Nicholson, M. Milford, and N. Sünderhauf, "QuadricSLAM: Dual Quadrics From Object Detections as Landmarks in Object-Oriented SLAM," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 1–8, Jan 2019.

[4] A. Elfes, "Using Occupancy Grids for Mobile Robot Perception and Navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[5] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[6] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3818–3825.

[7] S. Thrun, "Learning Metric-Topological Maps for Indoor Mobile Robot Navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370297000787

[8] B. Kaleci, K. Turgut, and H. Dutagaci, "2DLaserNet: A deep learning architecture on 2D laser scans for semantic classification of mobile robot locations," *Engineering Science and Technology, an International Journal*, vol. 28, p. 101027, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098621001397

[9] N. Sünderhauf, F. Dayoub, S. McMahon, B. Talbot, R. Schulz, P. Corke, G. Wyeth, B. Upcroft, and M. Milford, "Place Categorization and Semantic Mapping on a Mobile Robot," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 5729–5736.

[10] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning Deep Features for Scene Recognition using Places Database," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/3fe94a002317b5f9259f82690aeea4cd-Paper.pdf

[11] N. Zimmerman, T. Guadagnino, X. Chen, J. Behley, and C. Stachniss, "Long-Term Localization Using Semantic Cues in Floor Plan Maps," *IEEE Robotics and Automation Letters*, vol. 8, no. 1, pp. 176–183, 2023.

[12] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J. Fernandez-Madrigal, and J. Gonzalez, "Multi-Hierarchical Semantic Maps for Mobile Robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2278–2283.

[13] A. Pronobis and P. Jensfelt, "Large-scale Semantic Mapping and Reasoning with Heterogeneous Modalities," in *IEEE International Conference on Robotics and Automation*, 2012, pp. 3515–3522.

[14] S. Thrun, "Robotic Mapping: A Survey," 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:14633188

[15] A. Doucet, N. de Freitas, K. P. Murphy, and S. J. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks," *ArXiv*, vol. abs/1301.3853, 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:2948186

[16] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[17] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2011, pp. 155–160.

[18] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient Sparse Pose Adjustment for 2D mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 22–29.

[19] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," in *Proceedings of the National Conference on Artificial Intelligence*, 01 1999, pp. 343–349.

[20] P. Pfaff, W. Burgard, and D. Fox, "Robust Monte-Carlo Localization Using Adaptive Likelihood Models," in *European Robotics Symposium*, H. I. Christensen, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 181–194.

[21] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[22] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234.

[23] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Computer Vision – ECCV*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443.

[24] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense Tracking and Mapping in Real-Time," in *International Conference on Computer Vision*, 2011, pp. 2320–2327.

[25] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-Time Dense Surface Mapping and Tracking," in *10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[26] 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 *IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1352–1359.

[27] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," in *Computer Vision – ECCV*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 834–849.

[28] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, *RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 477–491. [Online]. Available: https://doi.org/10.1007/978-3-642-28572-1_33

[29] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "ElasticFusion: Real-Time Dense SLAM and Light Source Estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016. [Online]. Available: https://doi.org/10.1177/0278364916669237

[30] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4628–4635.

[31] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831

[32] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1689–1696.

[33] D. Ball, S. Heath, J. Wiles, G. Wyeth, P. Corke, and M. Milford, "OpenRatSLAM: an open source brain-based SLAM system," *Autonomous Robots*, vol. 34, no. 3, pp. 149–176, Apr 2013. [Online]. Available: https://doi.org/10.1007/s10514-012-9317-9

[34] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, pp. 573–580.

[35] S. Panchangam, "Benchmarking Uncertainty Estimation of Deep Learning Models Using Synthetic Dataset," Master's thesis, Hochschule Bonn-Rhein-Sieg, Sankt Augustin, January 2023.

[36] N. Sünderhauf and M. Milford, "Dual Quadrics from Object Detection BoundingBoxes as Landmark Representations in SLAM," 2017.

[37] K. Kaminski, "Data association for object-based SLAM," Master's thesis, KTH Royal Institute of Technology, Stockholm, Sverige, 2020.

[38] F. Dellaert and G. Contributors, "Georgia Tech Borg Lab - GTSAM," https://github.com/borglab/gtsam, May 2022. [Online]. Available: https://github.com/borglab/gtsam

[39] N. Jablonsky, M. Milford, and N. Sünderhauf, "An Orientation Factor for Object-Oriented SLAM," *ArXiv*, vol. abs/1809.06977, 2018.

[40] T. Lemaire and S. Lacroix, "Monocular-vision based SLAM using Line Segments," in *Proceedings IEEE International Conference on Robotics and Automation*, 2007, pp. 2791–2796.

[41] M. Kaess, "Simultaneous Localization and Mapping with Infinite Planes," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 4605–4611.

[42] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, Nov 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[43] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," in *Computer Vision – ECCV*, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[44] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *International Conference on Computer Vision*, 2011, pp. 2564–2571.

[45] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[46] M. Hosseinzadeh, Y. Latif, T. T. Pham, N. Sünderhauf, and I. D. Reid, "Structure Aware SLAM Using Quadrics and Planes," in *Asian Conference on Computer Vision*, 2018.

[47] Y. Taguchi, Y.-D. Jian, S. Ramalingam, and C. Feng, "Point-Plane SLAM for Hand-Held 3D Sensors," in *IEEE International Conference on Robotics and Automation*, May 2013, pp. 5182–5189.

[48] A. J. B. Trevor, S. Gedikli, R. B. Rusu, and H. I. Christensen, "Efficient Organized Point Cloud Segmentation with Connected Components," in *In: 3rd Workshop on Semantic Perception Mapping and Exploration (SPME), Karlsruhe, Germany*, 2013.