UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DyOb-SLAM : DYNAMIC OBJECT TRACKING SLAM SYSTEM

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

RUSHMIAN ANNOY WADUD

Norman, Oklahoma

2021

DyOb-SLAM : DYNAMIC OBJECT TRACKING SLAM SYSTEM

A THESIS APPROVED FOR THE

SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Wei Sun, Chair

Dr. Samuel Cheng

Dr. Jie Cai

Acknowledgements

I would like to express my appreciation and gratitude to my advisor, Dr. Wei Sun for his guidance, valuable comments and feedback at different stages of the thesis. I also want to give a big thanks to him for having a strong belief in me and preventing me from any kinds of trouble while doing my research. I would like to give a huge thanks to Dr. Samuel Cheng and Dr. Jie Cai for their willingness to serve on my thesis committee.

I would specifically like to thank my dearest family members for all of their support in all of my critical moments. My father, my mother, my elder sister and my elder brother - I am nothing without you. Thank you for always giving me the love and care I probably do not deserve. Thank you for always believing in me. I would like to thank Joyeeta Iqbal Prova for always being by my side and give constant support when the days were the darkest.

I would like to appreciate all the moral boost given to me by my friends from Bangladesh - Nafi, Rashik, Arif, Asif, Ayon, Sakib, Saifur, Mehnaz. I would also like to acknowledge my friends and elder brothers in Norman - Pritom, Ramiz, Tasfiq, Shatil, Shadman, Himadri, Dip, Mohan, Tanvir, for all the help and support I have got since I came to Norman for my Masters program. I might have missed a lot of people but I want to thank everyone from the bottom of my heart.

Lastly, I want to thank the OU Supercomputing Center for Education and Research (OSCER) team for providing me the resources and help that was required for the execution of my system. I would like to appreciate the effort of Dr. Henry Neeman, Mr. David Akin and Mr. Jason Speckman from OSCER to help me with the issues.

Contents

List of Figures

Abstract

Simultaneous Localization & Mapping (SLAM) is considered as the process of building a mutual relationship between localization and mapping of the subject in its surrounding environment. With the help of different sensors, different types of SLAM systems have been developed to simplify the problem of building the relationship between localization and mapping. A limitation in the SLAM process is to consider dynamic objects into calculation of mapping the environment, which is computationally heavier. Dynamic objects have not been taken into consideration while building a SLAM system until recently. The proposed system, DyOb-SLAM, is a Visual SLAM system which can produce an output considering dynamic objects in the environment. With the help of a neural network and a dense optical flow algorithm, dynamic objects and static objects in an environment can be differentiated. DyOb-SLAM creates two separate maps, for both static and dynamic contents. For the static features, a sparse map is obtained. For the dynamic contents, a trajectory global map is created as output. As a result, a frame to frame real-time based dynamic object tracking system is obtained. Through the pose calculation of the dynamic objects and the camera, DyOb-SLAM can estimate the speed of the dynamic objects with time.

The performance of DyOb-SLAM is observed by comparing it with a similar Visual SLAM system, VDO-SLAM and the performance is measured by calculating the camera and object pose errors as well as the object speed error. In the following chapters, the entire SLAM system along with some details of its predecessors are explained.

Chapter 1

INTRODUCTION

With the advent of modern technology, people can now use GPS to know and understand their surrounding and also to locate themselves in the GPS map. Robots can also use GPS to track themselves in the global environment. But GPS is not always the best way for gaining automation, especially in indoor areas. In robotics, understanding or recognizing the environment of a robot by itself is one of the first steps for automation. This includes understanding its local position. Simultaneous Localization and Mapping (SLAM) [4] is the system in which a robot can create a map of the surrounding environment by adding the key features in the map which the sensors are perceiving (Mapping) and at the same time localize itself in the map (Localization), estimating its state or pose in the surrounding environment.

The main goal of SLAM is to get a precise mapping of the unknown environment and to find the robot's current local position in the map in real time. With SLAM, a robot can know its location and just like humans, can decide to move in any direction or do some certain functions in an automated fashion. But the application of SLAM is not only limited to automated robots or drones; SLAM can also be used in Augmented Reality (AR)

technology. In AR technology, SLAM can be used to create the augmented map of the world and locate the device's 3D position in the world by calculating the spatial relationship between the device itself and multiple keypoints configured in the environment.

The Simultaneous Localization and Mapping problem can be considered as maintaining a mutual relationship between the mapping and localization of a robot in an explored environment. Without mapping, the subject cannot be localized and without the pose estimation of the subject, the map cannot be formed. With the help of the sensors, the significant landmarks or key-features can be located, which will be processed by the device to match and link them with the previously observed landmarks, as well as store them for mapping purpose. The state and position of a robot can be estimated after updating the landmark features and can be used for mapping as well. Many sensors have been utilized in SLAM, such as laser range sensors, rotary encoders, inertial sensors, GPS, and cameras. Depending on the sensors, SLAM can be classified into various types. In this proposal only Visual SLAM is focused.

The prime goal of Visual SLAM is to estimate the camera trajectory and reconstruct the surrounding environment as a map. In this technique, the camera takes consecutive frames of the environment and by setting some key-points in these frames, the position of the robot or drone is tracked at first. Then, the local map is created using these points and the robot or drone localizes itself in the map by optimizing the results. The results can be optimized by minimizing the difference between the projection points and the actual points.

The visual sensors are of different types:

- **Monocular** : Singular camera.
- **Stereo** : Combination of two monocular cameras
- **RGB-D** : Also called "Depth Camera", gives depth in pixel as output.

Most of Visual SLAM algorithms are based on an assumption called "Scene Rigidity Assumption" or static world assumption. This assumption has been developed in many approaches of SLAM systems where it is considered that the environment does not contain dynamic objects and is completely static. Although the scene rigidity assumption is required for the ease of computation since dealing with dynamic objects is computationally expensive, it creates a limitation for the real world based applications of Visual SLAM. SLAM systems developed in recent years have taken dynamic objects into consideration. These algorithms function mostly in two different ways:

1. The moving objects detected from the sensors are treated as outliers and removed from the estimation process.

2. After detecting the moving objects, they are tracked separately using multi-target tracking approaches.

DynaSLAM [5] functions in the former way, where it detects "prior" dynamic objects, i.e. objects which are potentially dynamic, segmenting the objects using a Mask-RCNN model and then removes the segmented portions from the frames. The map that DynaSLAM generates is based on the static objects in the surrounding environment. VDO-SLAM [6], on the other hand, is a system which functions in the latter way, i.e. it tracks the dynamic objects and estimates the object poses (both static and dynamic).

1.1 DyOb-SLAM

The proposed SLAM system, DyOb-SLAM is a combination of DynaSLAM and VDO-SLAM. Figure 1.1 shows the output of the DyOb-SLAM system. Different features are mentioned below:

- The system consists of a Mask-RCNN module for segmenting out the dynamic objects based on prior.
- For getting a robust system of dynamic object tracking, a dense optical flow and a scene flow algorithm are implemented.
- The back-end of the system consists of Bundle Adjustment module for the static object points, the Partial Batch Optimization module for creating local maps and a Full Batch Optimization module for the final result - a global map.
- With the help of the Bundle Adjustment [7] algorithm, a sparse map can be obtained with the static feature points.
- The whole SLAM system is run in a cloud network to obtain fast and improved data in comparison with the ones obtained in the local system.

So, as outputs, the followings are obtained:

1. A current frame showing the ORB features along with mask information and object labels.
2. A sparse map based on the static features.
3. A global map showing the dynamic contents and their motion updated with time and each frame.

The whole system will be explained in details in chapter 4

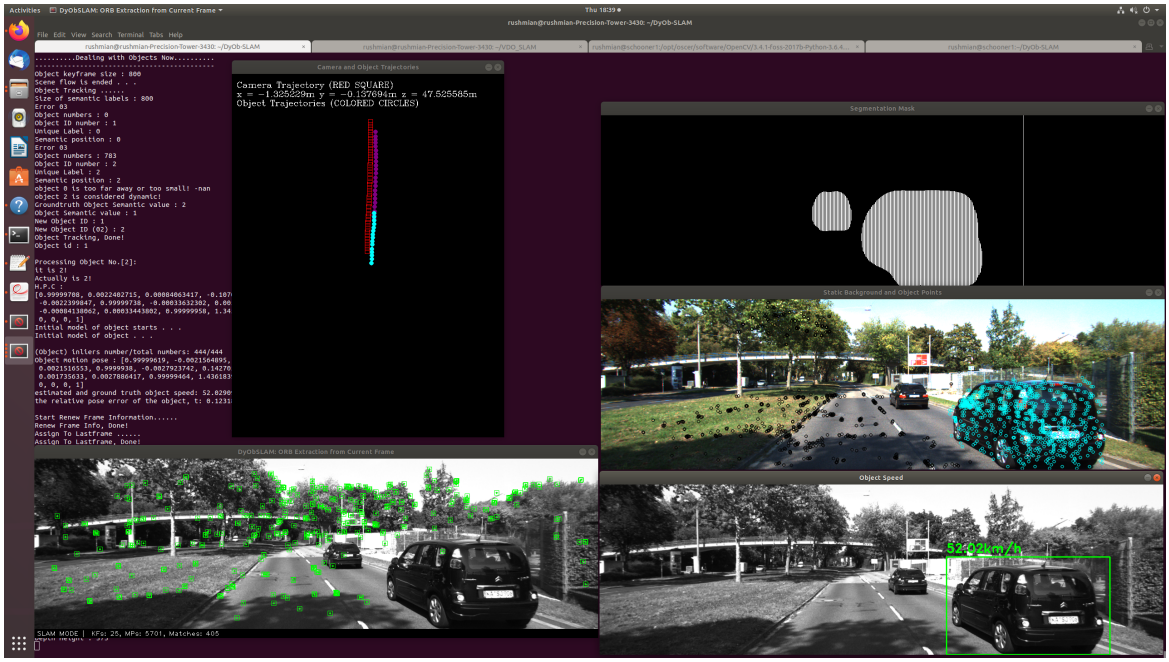

Figure 1.1: Output of DyOb-SLAM

Chapter 2

LITERATURE REVIEW

There are two types of approach in Visual SLAM - "Feature based" and "Direct based". Feature based approach [8] [9] [10] is based on key-point matching and builds a sparsely reconstructed map of the environment, whereas direct based approach gives a dense reconstruction map [11] or semi-dense map [12]. Direct based method usually requires more computational cost, i.e. stronger GPUs.

Some of the popular feature based SLAM systems are MonoSLAM [13], PTAM [8], ORB-SLAM [9], ORB-SLAM2 [10], OpenVSLAM [14]. MonoSLAM is the first real time monocular visual SLAM system which provides a real time probabilistic 3D map updated by Extended Kalman Filter method. PTAM [8] is the first visual SLAM system to have separate but parallel tracking and mapping threads. The idea of Orb-SLAM [9] and Orb-SLAM2 [10] came from PTAM's system but with an added thread - loop closure. Loop closure thread takes the last keyframe processed by the local mapping thread, and tries to detect and close loops [9]. The features that are extracted from the image frames in [9] and [10] are called ORB (Oriented FAST and Rotated BRIEF) [15], which are rotation invariant and resistant to noise. It is based on two types of descriptor - FAST (Features from

Accelerated Segment Test) and BRIEF (Binary Robust Independent Elementary Features). In both [9] and [10], ORB features are extracted and then the features are accumulated to initialize the map by performing Bundle Adjustment.

Direct method of visual SLAM is based on formation of dense or semi-dense maps. KinectFusion [11], a dense SLAM, gives an output of a dense 3D reconstruction of complex room sized scenes using the depth images obtained from the Kinect depth sensor. The scene model is maintained with a volumetric, truncated signed distance function (TSDF) representation using the consecutive depth frames and the associated camera poses. ElasticFusion [16] is a real time dense SLAM which is capable of capturing comprehensive dense globally consistent surfel-based maps of room scale environments. Direct methods like LSD-SLAM [12] produces semi-dense global maps consisting of keyframes as vertices and 3D similarity transforms as edges. Direct Sparse Odometry (DSO) [17] combines the benefits of direct methods (seamless ability to use and reconstruct all points instead of only corners) with the flexibility of sparse approaches (efficient, joint optimization of all model parameters).

2.1 Mapping

The most important factor of SLAM is how precisely the robot maps its environment. In general, mapping is a very challenging aspect in SLAM. The environment can be mapped in many ways, depending on what sort of algorithm is used. Some of the well known map representations are described as follows with examples -

2.1.1 Metric Map

In [18], metric representation or metric mapping is defined as a symbolic structure that encodes the geometry of the environment, i.e. the environment is represented in terms of

geometric relations between the objects and a fixed reference frame [19]. For 2D cases, there are two types of paradigms for metric representation - “landmark-based maps” and “occupancy grid maps”. Landmark-based maps give a sparse set of landmarks as the output map, while occupancy grid maps provide a discretized map of the environment assigned with an array of cells, where each cell has a probability of region occupation. For 3D cases, there are many forms of mapping.

1. Feature maps represent the environment in a form of sparse geometric shapes such as points and straight lines [19]. The features which are detected to be set in the map are described by the location and geometrical shapes. Orb-SLAM [9] and Orb-SLAM2 [10] detects the ORB [15] features and map out the environment using the ORB features in their correct location and shape in the form of points. This accumulation of points with the help of Bundle adjustment technique, forms a point cloud map.

2. In dense mapping systems, information collected from all the pixels from the image are applied. KinectFusion [11] uses the depth information collected from the sensors and incrementally fuse the consecutive depth frames to make a 3D reconstruction. In CNN-SLAM [20], depth measurement is also used to reconstruct the 3D space depending on the selected keyframes and their pose. In ElasticFusion [16], a dense surfel map is created by fusing surfel model with the initialized deformation graph which mirrors the surfel model. LSD-SLAM [12] being a semi-direct SLAM system generates a large-scale, consistent, semi-dense map using a semi-depth map along with a pose graph optimization tool “g2o” [21].

3. In 3D spaces, occupancy grid map works the same way as in 2D spaces - it creates a discretized map of the environment. But in 3D cases, the volume of the key features are considered to be discretized and reconstructed in the map. A very important

method for occupancy grid map that is followed in three dimensional cases is use of octree. Octo-map [22] is based on an octree data structure and explicitly represents not only occupied space, but also free and unknown areas.

2.1.2 Topological Map

A topological map is basically a graph representation of the surrounding. It usually consists of nodes which describe the certain landmarks and contain the distinct features. In [23], the topological map is created based on the "Generalized Voronoi Graph (GVG)" which is a one-dimensional set of curves that captures the salient topology of the robot's environment. Topomap [24] transforms a sparse feature based map into a 3D topological map by extracting the occupancy information from the point cloud.

2.1.3 Semantic Map

Semantic SLAM is an approach to SLAM method which adds semantic information about the environment. Semantic map creation is highly dependent on deep learning techniques. Some of the semantic map induced SLAM systems are SLAM++ [25], CNN-SLAM [20], SemanticFusion [26] and more. A design is made in [25] where high quality 3D models of the repeatedly occurring objects were established in the map and a mesh of the objects were extracted from truncated signed distance volume using marching cubes. In CNN-SLAM [20], reconstructed maps were generated using the semantic labels obtained from Convolutional Neural Network (CNN) and fusing it with the depth map. Semantic-Fusion [26] map deals with a combination of semantic labelling from CNN and a dense SLAM system, ElasticFusion [16], providing a long term semantic dense correspondence.

Object detection is one of the prime aspect of Computer Vision algorithms. Usually semantic mapping process of visual SLAM techniques depends on a neural network structure. After training these modules on different databases these neural networks detect the

objects with the help of probabilistic bounding boxes or mask segmentation. Some of the popular neural network architectures are - Single Shot Detector (SSD), You Only Look Once (YOLO), Region based-CNN (RCNN) and Mask-RCNN.

- **SSD:** Single Shot detector (SSD) [27] produces default boxes as discretized bounding boxes and generates scores for presence of each object category in each default box. Detect-SLAM [28] uses SSD as the object detector in its system.

- **YOLO:** You Only Look Once (YOLO) [29] is considered as one of the fastest detector. The reason is that this network is based on a single layered convolutional layer in which multiple bounding boxes and class probabilities of those boxes are predicted simultaneously. YOLO is used as the detector part of SLAM in [30].

- **R-CNN:**Region based CNN (R-CNN) [31] uses a region proposal algorithm to localize objects and computes the fixed size CNN input from each region proposal. There have been many updates on the R-CNN structure making the detector more accurate and faster.

  - **Fast R-CNN:** For each object proposal in R-CNN, the neural network is forward passed, as a result the training process slows down. In Fast R-CNN [32] the training algorithm is improved and performed in a single stage using multi-task loss and can update all the network layers.

  - **Faster R-CNN:** In Faster R-CNN [33], a full convolutional network "Region Proposal Network" is introduced in the Fast-RCNN system, which predicts bounding boxes for objects and their probabilistic scores at each position.

  - **Mask R-CNN:** The most recent version of R-CNN is Mask-RCNN [1] which gives a more accurate result in object detection. In this method, an object in-

stance segmentation network is added on the Faster R-CNN model, which runs parallely with the branch of bounding box recognition. Many SLAM systems have recently added Mask-RCNN for the detector part - Mask-Fusion [34], DynaSLAM [5], DetectFusion [35] and more. Mask-RCNN is mostly used for dynamic object detection purpose.

In SLAM methods, the most common assumption made is that the observed scene in the environment is static. This assumption leaves out the dynamic objects. As a result, in a system if there is no moving objects, then any motion of an object is treated as an outlier and thus occluded from the tracking and mapping [35]. This will result in failure of tracking and subsequently the mapping. For this purpose, dynamic object detection is widely used currently to eliminate this issue. Some of the dynamic object detector SLAM methods are described below:

- **Detect-SLAM:** In Detect-SLAM [28], ORB-SLAM2 [10] and the object detector SSD [27] mutually co-exist in the system to detect the dynamic objects for occlusion in the local map, and with the help of SSD, an instance level semantic map are formed based on the static objects.

- **Detect-Fusion:** With the help of YOLOv3 [36], the dynamic objects are detected and then an instance level segmentation is applied using 3D geometric segmentation method. The dynamic contents are omitted in the tracking stage. [35]

- **DynaSLAM:** Based on ORB-SLAM2, DynaSLAM [5] detects and segments the dynamic contents using Mask-RCNN [1] to obtain a tracking output with less error. The background where the dynamic object occlusions take place are then inpainted with an algorithm from camera's previous data.

- **Mask-Fusion:** In [34], a purely geometric map of the static scene is constructed. If the static objects are manipulated by moving them, the map will still contain the objects after occluding the human hand or disturbance.

2.2 Optimization Techniques

Every SLAM system is based on two parts - Front end and Back end. The front end of SLAM consists of the tracking and mapping portion - where the sensor data is turned into models. These sensor data is optimized later in the back end. The back end can feedback its output to the front end again to optimize the results. Optimization based SLAM systems usually consist of two parts: one portion identifies the constraints of the problem depending on the sensor data and finds correspondences between the newly found keypoints and the previous developed map points, while the other portion fine-tunes the pose of the sensors and objects eventually updating the map to obtain a whole perspective of the explored environment. Filter based approach of optimization in SLAM systems can be identified by two main branches - Bundle Adjustment and Graph based approach.

- Bundle Adjustment [7] optimizes the 3D structure and the pose information in a joint fashion. The main objective of Bundle Adjustment is to minimize the reprojection error, i.e, the distance between the detected key features in an image and the reprojected past features.

- Graph based approach deals with the graphical representation of the relationship between the key features observed and the sensor or object poses. This graphical method is later on converted to an optimization framework, i.e. a cost function to be minimized.

Bundle Adjustment is one of the most used framework for optimization in SLAM sys-

tems. Different systems have used this algorithm in different ways according to the requirement. In [37], to solve the bundle adjustment complexity of multiple variables, two key frames are chosen among all the frames to represent a given driving scenario. A technique of using sliding windows over the selected keyframes have been used in [38] to locally minimize the reprojection error. Local Bundle Adjustment used in [39] optimizes the camera points observed from the last frames based on the 2D reprojections of the points at the frames to obtain areal time localization system. Two objective functions based on vision and inertial sensor data have been optimized in [40] which are weighed with a machine learning approach. In [41], only a set of frames are chosen to be optimized and the other frames are marginalized out in the process.

Graph based SLAM approach is based on the graphical representation of the SLAM problem. The graphical method can be converted to objective functions where different algorithms like Gauss-Newton, Levenberg-Marquardt, Gauss-Siedel relaxation, etc. can be applied. In [42], a tree based parameterization technique is used to describe the nodes of the graph which is then optimized using a stochastic gradient descent algorithm. In [43], a hierarchical representation of the SLAM problem is used in a manifold structure, where the lowest level represent the ground information and the highest level represent the structural information of the explored environment. g2o [21] uses a Hessian Matrix structure for solving the complex nature of data optimization. iSAM [44] or Incremental Smoothing and Mapping uses QR factorization method to obtain a sparse smooth information matrix.

There are also other methods of optimization used in the solving the SLAM problem. One of them is pose graph optimization. In [18] pose graph optimization is defined as the system in which variables to be estimated are poses sampled along the trajectory of the robot and where each factor imposes a constraint on a pair of poses. In [45], a non-linear least-

squares optimization method, Gauss-Newton method is used at the back-end, which deals with singularities in the representation of the robot poses in an elegant manner. In most systems like ORB-SLAM [9] and PTAM [8], an outlier rejection method is used which is called RANSAC (Random Sample Consensus), to minimize the reprojection error. RS-SLAM [46] utilized RANSAC sampling to check for inlier misclassifications in the original correspondences to mitigate the errors.

2.3 Challenges

The biggest challenge in SLAM methods is projecting dynamic objects in the local map without affecting localization. The dynamic object-aware SLAM systems mentioned previously [28] [35] [5] [34] treat dynamic objects as outliers and filter them to get a static object based environment. This is a big limitation in real life based scenarios, especially in online application of autonomous driving.

The recent dynamic object based SLAM systems not only detects the objects but also tracks the motion of these objects. Some of the recent works in the SLAM based technology deal with mapping and localization of dynamic objects. In CubeSLAM [47], 3D objects are detected in dynamic scenes forming 3D shaped cuboids on the dynamic objects, instead of 2D bounding boxes and tracked after optimizing the object data with a multi-view Bundle adjustment. VDO-SLAM [6] integrates dynamic and static structures in the environment into a unified estimation framework using pre-processed object detection information and optical flow algorithm. ClusterVO [48] is a visual odometry (VO) system in which a heterogeneous clustering approach is used to cluster the tracked keypoints in the segmented regions of dynamic objects obtained from the object detector YOLOv3 [36] and simultaneously estimates the semantic spatial and motion information. DOT or Dynamic Object Tracking [49] combines instance segmentation and multi-view geometry to generate

masks for dynamic objects, which works like a tracker. The improved and extended version of DynaSLAM [5], DynaSLAM II [50] deals with an object-aware Bundle Adjustment [7] to jointly optimize both the static and dynamic contents to obtain a tracker for multiple dynamic objects and a global map including dynamic contents. Besides, other dynamic object-aware SLAM systems such as - in [51], dynamic objects are detected and tracked with a 3D box inference method and a dynamic object based Bundle Adjustment (BA) [7] approach is done to continuously track the state of the objects.

Another big challenge in SLAM is to find a solution for running computationally heavy SLAM systems in real time. For example - using deep learning in SLAM systems to produce semantic information needs a lot of energy for computation which cannot be run efficiently in real time. One solution to run heavy computational systems in real time is cloud computation. The higher computational jobs are performed in a cloud to get real time data for building up the map and localization information. In robotics, there are many cloud computation platforms for robots like - Rapyuta [52] which is an open source Platform-as-a-Service (PaaS) framework designed specifically for robotics applications (SLAM), DAvinCi [53] which is a software framework that provides the scalability and parallelism advantages of cloud computing for service robots in large environments, ROS-bridge which bridges communication between a robot and a single ROS (Robot Operating System) [54] environment in the cloud.

The proposed SLAM system DyOb-SLAM deals with the solution of these mentioned challenges. The two SLAM systems - DynaSLAM [5] and VDO-SLAM [6], which the proposed system DyOb-SLAM is based upon, are described below:

2.4 DynaSLAM

DynaSLAM [5] is a SLAM system which deals with dynamic moving objects. The whole system is built on Raul Mur's ORB-SLAM and ORB-SLAM2 and it can be implemented with Monocular, stereo and RGB-D image sets. The system is comprised of a neural network Mask-RCNN to segment out the dynamic objects along with a multi-view Geometry algorithm, a tracking component to track the static objects and a mapping component to map out the static feature points. The main contribution of this system is that it has a dynamic object occlusion algorithm, along with a background inpainting algorithm to inpaint the occluded spaces in the frames with previously observed background.

2.4.1 Mask-RCNN and Multi-View Geometry

DynaSLAM uses a combination of a convolutional neural network, Mask-RCNN [1] and a multi-view geometry algorithm for instance segmentation of the dynamic objects. The Mask-RCNN model was trained with MS-COCO [55] dataset and the system works by segmenting the potential dynamic objects first, for example- person, cars, birds, etc. The multi-view geometry algorithm helps to segment out the dynamic objects which are not potentially dynamic but movable. The combination of the two models give a proper and robust result in detecting the dynamic objects.

2.4.2 Low-Cost Tracking

The low cost tracking module is a computationally lighter version of the tracking module of ORB-SLAM2 [10]. After the segmentation of the input images, the low-cost tracking module determines the camera pose using the static part of the environment. The tracked static parts are then fed to the multi-view geometry module to detect the moving objects in the surrounding.

2.4.3 Tracking & Mapping

The tracking and mapping part of DynaSLAM is similar to ORB-SLAM2 [10], only the difference is that in DynaSLAM the inputs are the RGB and depth images along with the segmentation masks. The ORB [15] features are extracted from all the regions of the frames excluding the segmented portions. A sparse point cloud map is created using the static background.

2.4.4 Background Inpainting

After segmentation of the potential dynamic objects as well as the moving objects, the segmented dynamic objects are occluded from the frames. To get a complete static environment, DynaSLAM has a background inpainting feature which inpaints the occluded dynamic object region with the static background using the previously observed frames.

2.5 VDO-SLAM

Visual Dynamic Object-aware SLAM system, or VDO-SLAM [6] is designed by Jun Zhang, Mina Henein, Robert Mahony and Viorela Ila which is the first dynamic SLAM system to perform motion segmentation. The system's novelty is that it can track multiple dynamic objects with the semantic information, estimate the camera pose of both static and dynamic structures and it has an object velocity extracting algorithm. The tracking of multiple dynamic objects is done using dense optical flow algorithms.

2.5.1 Pre-Processing the inputs

The system takes stereo or RGB-D images as inputs. For detection of dynamic objects, instance level semantic segmentation is done for potentially dynamic objects. Dense optical flow is taken into measure for better tracking of the dynamic objects. The semantic information (.txt file) and optical flow data (.flo file) are provided as input.

2.5.2 Tracking

Tracking of static and dynamic objects is composed of two modules - camera ego-motion estimation and object motion tracking. For camera ego-motion module, the camera pose is estimated along with the object feature information. The corner features of objects are first detected and then tracked with optical flow.

For object motion tracking, at first the segmented objects are classified into static and dynamic objects and then the dynamic objects are associated across pairs of consecutive frames to estimate the scene flow. The magnitude of scene flow gives us information about the dynamic objects if it surpasses a threshold value. Then for tracking the moving objects, the optical flow estimation is used.

2.5.3 Mapping

After camera pose estimation and tracking the objects and surrounding, only the inlier points are saved into the map. The output is a global map and a local map which is extracted from the global map. Both the maps are updated with a batch optimization processes - Local batch and Global batch. After the local map is constructed, a factor graph optimization is performed for better refinement of the points and then updated into the global map. The local map consists of only the previous frame information, while the global map consists of the tracking outputs - the camera poses and object motion information along with the inlier structure.

Chapter 3

THEORY

In this chapter, the theory behind the methodologies used in the DyOb-SLAM system are explained in details. The object detector consists of a Mask-RCNN [1] module for instance segmentation and a PWC-Net [56] trained input for optical flow purpose. The tracking module estimates the camera and object motion pose information from the 3D-2D reprojection errors. The mapping module along with the optimizing back-end of the system uses the local and global map formation algorithms and optimizing them with Bundle Adjustment [7] techniques. These methodologies are described in the following sections:

3.1 Mask-RCNN

Mask-RCNN [1] is an extended version of Faster-RCNN [33] which has an extra branch for producing output as object masks along with the bounding box recognition. From Faster-RCNN two outputs are obtained - a class label and a bounding box offset. Mask-RCNN gives a third output, which is an object mask. The network in Mask-RCNN is similar to the one of Faster-RCNN as the two stages of the network in serial being a Region Proposal Network (RPN) which proposes bounding boxes for objects and a layer that extracts features from each bounding boxes, helping in classification of the objects. The

difference is that in Mask-RCNN, the second stage also produces a binary mask in parallel for each Region of Interest (RoI).

The figure 3.1 shows the framework of Mask-RCNN for instance segmentation of objects. For extracting the features from the RoI, a standard operation is executed called RoIPool [32] which quantizes the floating numbers of RoI and discretizes them. After being discretized, the numbers are separated into spatial bins from which the feature values are aggregated using a discretizing method like Max-Pooling. The main objective of max pooling is to down-sample the image or reducing its dimensions and then to predict the features based on the classes the model has been trained to detect. In Mask-RCNN, the quantization of RoIPool is replaced by a new layer called RoIAlign which aligns the extracted features with the inputs. After the features are properly extracted and aligned, a Km^2 dimensional mask is produced as outputs for each RoI, where K is the number of $m \times m$ resolution binary masks for each of K number of classes.

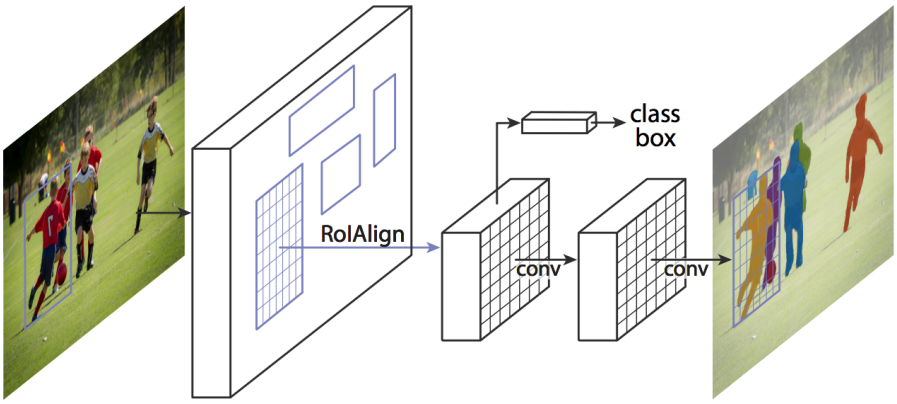

Figure 3.1: Framework of Mask-RCNN for instance segmentation [1]

In the convolutional neural network (conv) portion, the features in the images are extracted using a network called ResNet-50 [57] which consists of 50 layers. This network is a part of Faster-RCNN [33] and in its final convolutional layer of the fifth stage, the features are extracted. This is termed as the *head* of the Mask-RCNN convolutional network.

The head of the network also consists of a Feature Pyramid Network (FPN) [58] which have fewer filters compared to that of Faster-RCNN. The RoI features are extracted by the different levels of the FPN backbone.

The Mask-RCNN module used in DyOb-SLAM is a pre-trained model which has been trained on the COCO [55] dataset. It creates pixel-wise semantic segmentation and instance object labels. The Mask-RCNN network is trained to detect potentially dynamic objects like - person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra and giraffe. After instance segmentation of the frames, the network is further designated to allocate the objects of different classes with different shades which will be easy for mapping the object trajectories in the global map. The details of Mask-RCNN used in DyOb-SLAM can be found in chapter 4.

3.2 Optical Flow

Optical Flow is one of the most promising methods to analyze the motion characteristics of dynamic objects in computer vision. With the help of optical flow, velocity shifts as well as the displacement of points of dynamic objects can be tracked. This method determines the displacement of the intensity of each pixel (x,y) of the dynamic object regions at each frames in a certain time range, i.e. (dx, dy) at time dt. In [59], optical flow has been explained as an equation of intensity of the pixels for a pixel displacement (dx, dy, dt) as -

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{3.1}$$

which later can be transformed to a standard optical flow equation -

$$(\frac{\partial I}{\partial x})V_x + (\frac{\partial I}{\partial y})V_y + (\frac{\partial I}{\partial t}) = 0 \tag{3.2}$$

where V_x and V_y are the horizontal and vertical velocity components and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the partial derivatives of intensity with respect to x, y and t.

In DyOb-SLAM, dense optical flow method is chosen to track the dynamic objects, which gives flow vectors of all the pixels in the frame. Its details will be found in the chapter 4.

3.3 PWC-Net

PWC-Net [56] is a CNN model for optical flow algorithm. It processes the optical flow estimate from the current frame using it to warp the CNN features from the next frame. The warped features of second image and first image's features are used to create a cost volume to estimate the actual optical flow using the CNN model. In DyOb-SLAM, the dense optical flow used is at first trained by PWC-Net on FlyingChairs [60] dataset and then fine-tuned on Sintel [61] and KITTI [62] training datasets. The dense optical flow data for the experimented datasets are stored and then used as inputs directly.

3.4 Bundle Adjustment

Bundle Adjustment has been defined in [63] as the most accurate way to recover structure and motion by performing robust non-linear minimization of the measurement (re-projection) errors. It is a core component in state-of-the-art multi-view geometry systems. In [7], the author described Bundle Adjustment as the process which minimizes the sum of errors between 2D observations and the predicted 2D points, where the predicted points are re-projected from 3D structures by camera parameters. It is actually a least square problem as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} (u_{ij} - \pi(C_j, X_i))^2 \tag{3.3}$$

In [64], Bundle Adjustment is defined as a problem of refining visual representation to

produce 3D structure of the surroundings and creating parameters like camera pose. The cost function shown in equation 3.3 is a quadratic function for feature reprojection errors and to obtain robustness, outlier screening is required. The following points are mentioned in [64] to understand why Bundle Adjustment is better than other optimization techniques:

- Bundle Adjustment is very flexible in handling varieties of different features, scene types, information sources and error types.
- Since it uses accurate statistical error models, the end result of Bundle Adjustment is close to accurate and easily interpretable.
- Bundle Adjustment can be very efficient when solving larger problems as economical and rapidly convergent numerical methods are used.

One of the most prominent algorithms used for Bundle Adjustment error function is the Levenberg-Marquardt Algorithm [65]. The algorithm is defined as an iterative technique that locates a local minimum of a multivariate function that is expressed as the sum of squares of several non-linear, real-valued functions. It is also considered as a combination of the steepest descent and Gauss-Newton methods. In LM algorithm, if the current solution is not very close to the local minimum, the algorithm will behave like the steepest descent method which is slow but will converge. If the solution is close to the local minimum, the algorithm will act like the Gauss-Newton method which exhibits fast convergence.

Chapter 4

METHODOLOGY

DyOb-SLAM is a system which tracks dynamic objects, maps both the static and dynamic objects separately and simultaneously estimates the camera and object poses by comparing with the ground truth information. The system is comprised of -

- an object detector module
- a tracking component
- two different mapping algorithms
- an orientation optimizing back-end

The input to the system are stereo, RGB-D images. Figure 4.1 shows the block diagram of the DyOb-SLAM system.

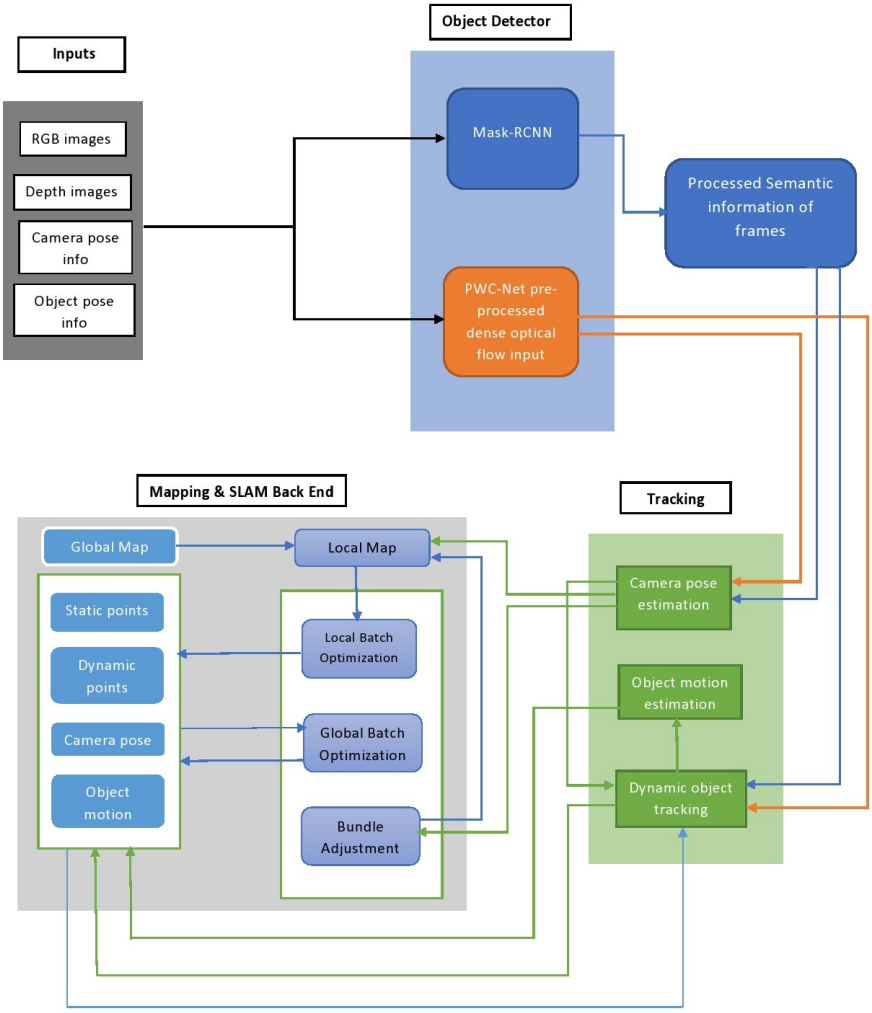

Figure 4.1: Block diagram of DyOb-SLAM

The object detector module at first creates instance-level semantic segmentation information of dynamic objects present in each frames. The dynamic object segmentation is based on *prior*, i.e. the objects which are potential dynamic objects or movable in real world, for example - car, people, etc. Due to the semantic segmentation of *a priori* dynamic contents, the static and dynamic objects are separated which will be easier to track the objects separately. With the help of a dense optical flow algorithm, the number of dynamic objects to be tracked is maximized. The dense optical flow information is pre-processed using PWC-Net, which samples all the points of dynamic contents in the frame from the semantic information.

The semantic information along with the optical flow information are then passed onto the Tracking module which tracks the dynamic points extracted from the semantic and optical flow information and produces camera and object pose information. It also compares the pose information with the ground truth information provided and gives out pose errors as output. Next, the tracking information is projected in two different maps - a sparse point-cloud map for the static contents and a global map for the dynamic contents along with the camera which provides trajectory information along each frames. The sparse map is optimized using a Bundle Adjustment algorithm and the global map is optimized using batch optimization process (both full and partial). The different stages are described in the subsections below.

4.1 Object Detector

4.1.1 Mask-RCNN

In the system, a Convolutional Neural Network (CNN) is used to segment out the potential dynamic objects from the frame. An instance-level semantic segmentation algorithm module, Mask-RCNN [1] is used which is an extended version of Faster-RCNN [33] with an added branch for predicting an object mask in parallel with the bounding box feature. It can extract both pixel-wise semantic segmentation and instance labels of objects. In this system, both the functions are used - the *priori* dynamic objects are segmented out and instance labels are obtained to track the dynamic pixels.

The input to the Mask-RCNN module are stereo, RGB-D images. The network has been trained in such a way it can detect about 20 different potential dynamic objects, for example - people, car, trains, truck, birds, dog, cat, etc. The network has been trained on MS COCO [55] and trained to segment out the selected classes. The main concept is to

segment out these classes and obtain an output matrix of size $m \times n \times l$ where m,n are the row and column of the input matrix of images and l is the number of objects in the frames. For each class of objects, a specific ID value has been assigned which is obtained in the output matrix where the pixel has been masked by the neural network. For each value in the matrix, a specific color of mask is assigned for visualizing dynamic object segmentation in the tracking output (see figure 4.3).

The figure 4.2 shows the semantic segmentation of a frame scene. The dataset that is used to test the system contains cars and other vehicles, which are the only objects segmented out. The ground truth of the dataset contains the object pose and semantic information. The Mask-RCNN's output matrix is then processed for the other modules (Tracking and Mapping) in such a way that the dynamic objects detected are relabelled according to the ground truth. As a result the output becomes closer to ground truth information.

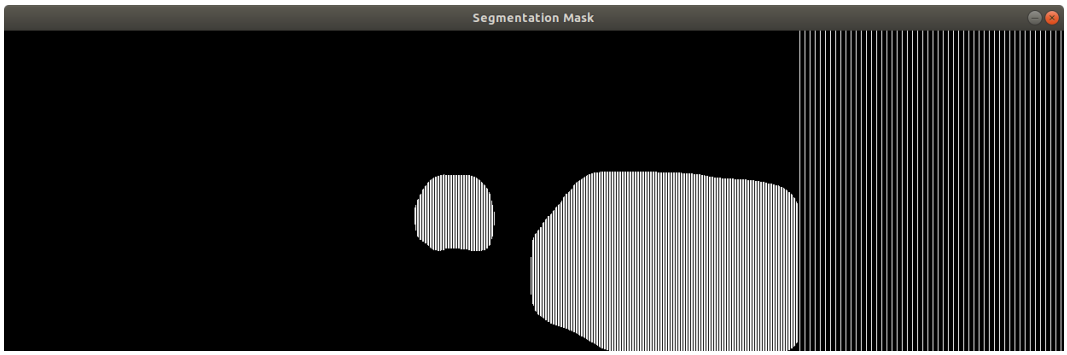

Figure 4.2: Semantic Segmentation of scene using Mask-RCNN in the experiment

4.1.2 Dense Optical Flow

With the help of PWC-Net [56], a dense optical flow algorithm is used to at first preprocess the optical flow information of the input images and saved as *.flo* files. These files are used as input to the system. The dense optical flow information sample all the points from the dynamic objects within the segmented masks. This helps to maximize the number

of tracked points and later on used for tracking multiple objects. Even if the semantic segmentation fails at one point, dense optical flow information can help obtain the object masks again by tracking the unique ID for each points in the mask around the object. Since sparse feature matching method is not very effective for tracking dynamic contents in the long term consecutive frames, optical flow estimation has been used for that purpose.

4.2 Tracking

The input to the tracking component are the RGB images, the depth information of each frames, the segmentation masks and the optical flow information obtained from the object detector module. Multiple functions take place simultaneously in the tracking component and they are divided in 3 modules -

- ORB Feature Extraction
- Camera Pose Estimation
- Object Motion Tracking

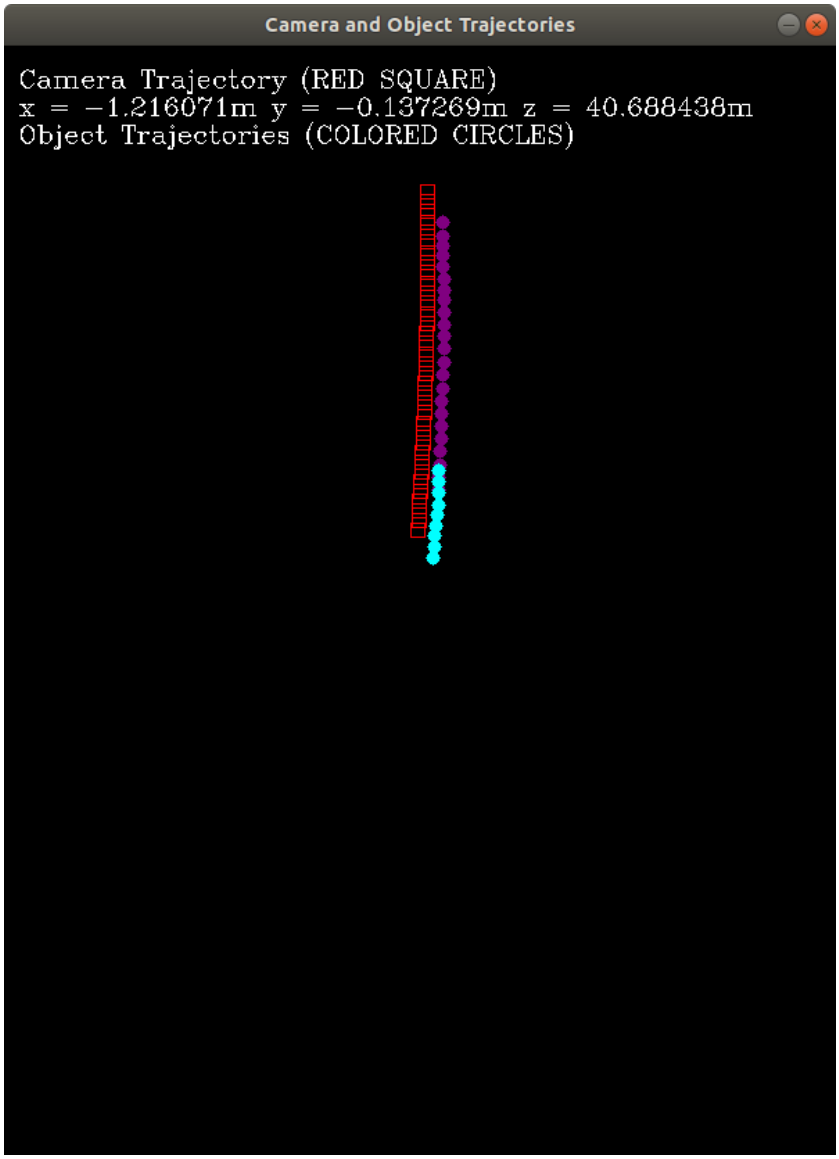

Figure 4.3: Tracked Dynamic Objects

4.2.1 ORB Feature Extraction

This module is similar to the process used in ORB-SLAM [9] or ORB-SLAM2 [10]. This module consists of the following -

1. **Localization:** It localizes the camera, finding feature matches in every frame and forms visual odometry tracks of unmapped regions.

2. **Loop Closing:** It uses a place recognition algorithm to detect and validate large loops. For place recognition, a Bag of Words module DBoW2 [66] is used.

Figure 4.4 shows the output of ORB [15] feature extraction. It detects the *corner* features in the frames, i.e. ORB features and extract them to form a sparse map as output. The ORB features are extracted from the static part of the image frames excluding the segmented mask portions of dynamic objects.

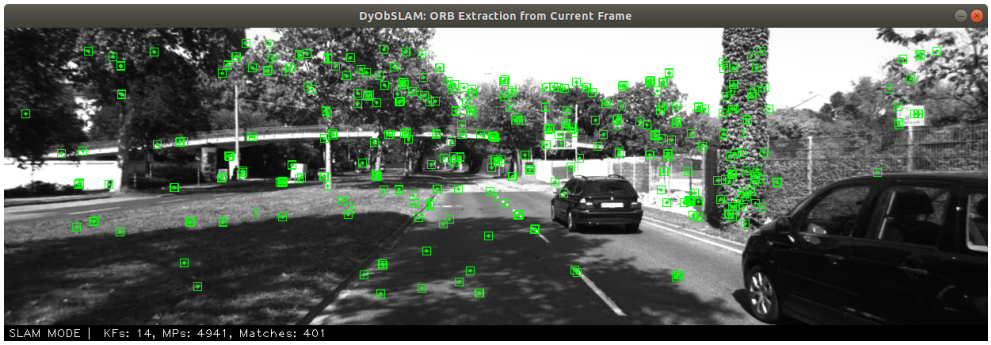

Figure 4.4: ORB feature extraction in the current frame

The ORB features are used for tracking, place recognition (Loop Closing) and local mapping functions and these features are very robust to rotation and scale [10] [15]. ORB, which is in short for Oriented FAST [67] and Rotated BRIEF [68], is a combination of FAST (Features from Accelerated Segment Test) keypoint detection method and BRIEF (Binary Robust Independent Elementary Features) descriptor which uses binary test for smoothing the noisy patches of pixels.

Since the system deals with stereo and RGB-D image types, the ORB extraction is done for both the images (left and right) and the ORB extractor searches for an ORB match in both the images. This matched keypoint associated with the depth information are used for differentiating between close and far points. Keypoints are considered close points according to [10] when the associated depth is less than 40 times the stereo/RGB-D baseline. Otherwise, the keypoint will be considered a far point. The baseline has been calculated according to [69]. The close points are usually considered and triangulated while the far

points are discarded. But the far points are triangulated if they are viewed from multiple view points.

4.2.2 Camera Pose Estimation

After the sparse and dynamic features are separated using the object detector module, the camera pose is estimated using the static feature points. For initializing the process, motion models are generated to compare the inlier numbers depending on the camera re-projection error. Two models are formed for robust estimation - one is used by considering the previous camera motion and the other produces a new motion transform using PnP based RANSAC algorithm. Each of the models creates a number of inliers and the model with the most inliers is chosen for initialization.

For camera pose estimation, at first the reprojection error equation is established. If P_{k-1} is a set of static points at frame (k-1) in the global reference frame and C_k is the set of corresponding static feature points in the image at frame k, then the camera pose X_k according to [6] is estimated by minimizing the reprojection error -

$$e(X_k) = C_k - \pi({X_k}^{-1} P_{k-1}) \tag{4.1}$$

Here $\pi(.)$ is a projection function. A least squares error function is established from equation 4.1 using Lie-Algebra parameterization of SE(3). This least squares error function is then minimized using the Levenberg-Marquardt algorithm [65].

4.2.3 Object Motion Tracking

The Mask-RCNN and optical flow modules segment out the potentially dynamic objects separating the static features from dynamic. Using a scene flow algorithm, the motion of the dynamic objects are estimated. This algorithm helps to further detect the dynamic objects properly. The scene flow algorithm can decide whether an object is in motion. Since the scene flow estimation of static objects is zero, a threshold of 0.12 is selected to decide whether the object is static or dynamic. If the magnitude of a scene flow vector of a certain point in the frame is greater than the threshold, that point is considered as dynamic. The scene flow vector is calculated using the camera pose and the motion of the object's point between two consecutive frames.

With the help of the dense optical flow information, a point label is associated with the dynamic object points. If the first dynamic object is detected, the point label will read $l = 1$ where $l \in \mathcal{L}$ and $\mathcal{L}$ are the point labels. For static objects and background, value of l is considered to be 0. So for frame k, the point labels will be aligned with the corresponding point labels obtained in previous frame k-1.

Similar to the camera pose estimation, object pose estimation is also calculated at first by calculating the reprojection error and then Lie-Algebra parameterization of SE(3). If the object point motion from frame k-1 to k in the global reference frame is ${}^{k-1}O_k$ the motion estimation equation can be derived as -

$$P_k = {}^{k-1}O_k P_{k-1} \tag{4.2}$$

This equation is the point motion estimation equation. Here, P_k is the set of static points in the frame k of image and P_{k-1} is the static points in the frame k-1. Using equation 4.2 the

reprojection error between the object point in global reference frame and the static points in image frame -

$$e({}^{k-1}O_k) = C_k - \pi({X_k}^{-1}[{}^{k-1}O_k]P_{k-1}) \tag{4.3}$$

After Lie Algebra parameterization of SE(3) the optimal solution is obtained minimizing the least squares error function.

With the help of the dynamic object motion tracking, the object speed is also calculated from the difference of estimated speed v_e and ground truth speed v_g, i.e.,

$$E = v_g - v_e \tag{4.4}$$

4.3 Mapping

The mapping component produces two types of map - Sparse Map for the static features and Global Map for the dynamic contents and camera motion.

4.3.1 Sparse Map

Like ORB-SLAM [9] or ORB-SLAM2 [10], the ORB features extracted in the tracking component are used to produce a sparse point-cloud map of the static background. This sparse map is also a local map consisting of triangulated ORB features from connected keyframes. The corresponding ORB features are matched with the previous keyframes and new points are generated into the local map. For triangulation of the ORB matches, the parallax error, reprojection error and scale consistency are checked. Figure 4.5 shows the sparse map obtained from the ORB extraction.

4.3.2 Global Map

The inputs to the formation of global map are the output from the tracking component, i.e. the camera pose information and the object motion. With each frame and gradual change in time step, the detected object motion and camera pose are saved and continuously updated. The inlier points obtained from the previous frames are utilized to gather the track correspondences in the current frame to estimate the camera pose and object motion. With different assigned colors for different values in the pixel matrices, the camera and object trajectories are visualized in the global map. It is only based on the dynamic content and camera. Figure 4.3 shows the global trajectory map of the detected dynamic objects.

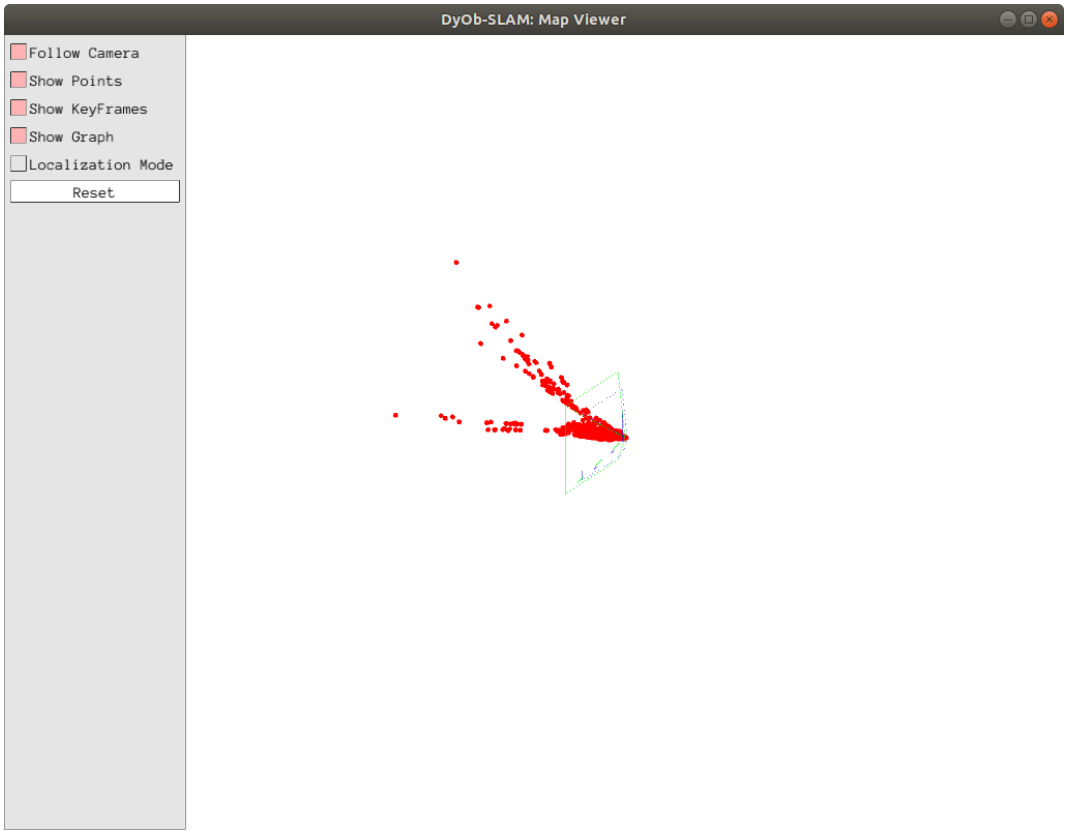

Figure 4.5: Sparse map of the static features in the scene

4.4 SLAM Back-end

The SLAM back-end is the part of the system where the data and output obtained from the other modules at the front-end are optimized to get a better and more optimized output as a whole. DyOb-SLAM uses the g2o [21] module for all the optimization functions and the Levenberg-Marquardt [65] method is implemented from it. The SLAM system is designed using three different types of optimization techniques for optimized static featured map, camera pose error and dynamic object motion estimation:

- Bundle Adjustment
- Local Batch Optimization
- Global Batch Optimization

4.4.1 Bundle Adjustment (BA)

The selected keyframes and sparse map points obtained from the mapping component are optimized by the Bundle Adjustment. It is used only for optimization of different attributes of the static features. The Levenberg-Marquadt method is used for Bundle Adjustment which is implemented in g2o.

Three different types of Bundle Adjustment are used in the system. To optimize the camera pose estimation obtained from the tracking component, a **Motion-Only Bundle Adjustment** is performed. This optimizes the camera orientation and minimizes the reprojection error obtained from the matched keypoints. **Local Bundle Adjustment** is used to optimize the selected local keyframes and local map points of the static features. After loop closure, **Full Bundle Adjustment** is used to optimize all the map points and keyframes except the origin keyframe to achieve the optimal solution.

4.4.2 Local Batch Optimization

Local Batch Optimization is used for formation of the local map and to be used as input of Global Batch Optimization. Its purpose is to ensure correct camera pose which is assisted by the Bundle Adjustment algorithm and sent to the global batch optimizer to form a precise global map. It optimizes the camera pose estimation by minimizing the reprojection error using the Levenberg-Marquardt method. For local optimization, only the static features are optimized as dynamic contents are big constraints.

4.4.3 Global Batch Optimization

After local batch optimization, its output along with the output of tracking components are directly used to optimize the global map. The tracked object points are optimized fully to form the global map with every consecutive time steps and obtain updated object poses. This optimization minimizes the pose error of both camera and objects. The global map is obtained after all the time steps and frames are processed and the pose estimations are globally optimized.

Chapter 5

CLOUD COMPUTING

With the gradual growth in the world of robotics, heavy computation based systems which deal with big forms of data are being executed for which a single computing system is not able to process efficiently. Such heavy computational systems, like SLAM systems that have developed over time, require more speed and accuracy to function like an online, real time based system. For such reasons, Cloud based services have been chosen as the latest method to solve this issue.

Cloud has been defined in [70] as a networking system on which any robot or automation system relies on for supporting the operation using data or code. Cloud Computing is a process of running a system in a virtual environment to obtain services like - storage or high processing power. With the help of Cloud computing technology, the computation, data sensing and memory usage for any robotics system can be distributed into multiple computing environments, i.e. all these functions will not be done in a single computer system. In [71], a cloud based SLAM system is designed where an open source framework, IoTCloud [72] has been used to connect the SLAM system to cloud services. IoT-Cloud is an open source cloud framework designed by the Indiana University which consists of

some distributed nodes to submit information from the IoT (Internet of Things) devices to different cloud services and also from the cloud services to the devices. SLAM systems like C2TAM [73] is based on a distributed framework of a Visual SLAM system in which the maps built are stored in the clouds and can be fused with newly explored maps. For C2TAM system, Bundle adjustment [7] has been performed in the cloud server for building up the maps. In [2], a framework of Visual SLAM has been shown (figure 5.1) which consists of an ROS [54] middleware providing necessary modules for the SLAM implementation and a cloud server is used for feature matching functions.

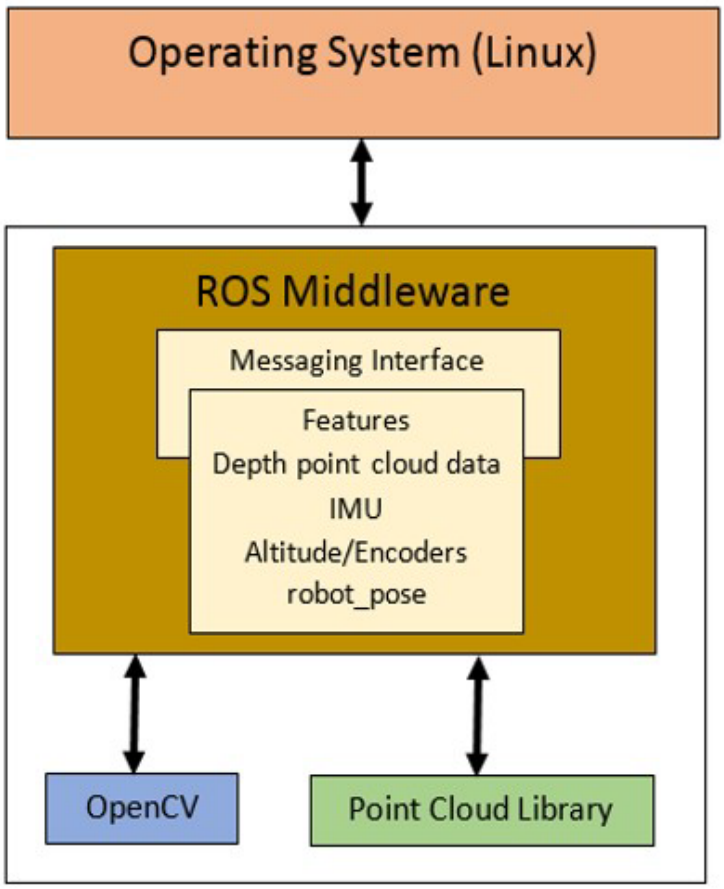

Figure 5.1: Framework of [2] using an ROS as middleware

Cloud computing can be differentiated in 3 different model types - Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Software as a Service (Google Docs, ZOHO [74]) is the type of cloud service which provides clients

with necessary softwares provided by the cloud which are called cloud applications. This type also provides the platform for the user to run their systems and the necessary infrastructures required for the systems to run on the platform. Platform as a Service (Google Application Engine [75]) is the type of cloud service which provides only the platform to run the system and the IT infrastructures required to run. Infrastructure as a Service (Amazon Web Services [76]) is the type which provides only the IT infrastructures like storage, computing and networks. In the robotics world, the three major frameworks popularized in the SLAM system development sector - DAvinCi [53], Rapyuta [52] and C2TAM [73]. These frameworks are widely used in the three cloud model types described above, for example - Rapyuta [52] works like a Platform as a Service framework, while DAvinCi [53] uses ROS [54] as a middleware to provide the necessary packages and then passes on to a Software as a Service cloud model. The DAvinCi server acts like a Platform as a Service.

5.1 Cloud computing in SLAM systems

Cloud computing technology is desired in SLAM systems, especially in the systems which contain modern object detection algorithms, as these algorithms require very high end GPUs (Graphic Processing Unit). In the proposed case, deep neural network Mask-RCNN requires high level GPUs which a single computer system would be expensive to be applied on. In [77], [71] and [78], there are mentions of different algorithms which are used to deploy a certain portion of the entire SLAM system in the cloud. For example - The Cloud Chaser [77] system deployed its object detection algorithm in cloud to get better performance and avoid latency issues. DyOb-SLAM consists of a dynamic object detector which is computationally heavier in comparison with the other modules in the system. Besides, the output of object detector is also pre-processed which results in a slower computation time, leading to weaker performance for the SLAM system. To obtain a much

better performing system, we deployed the proposed SLAM system in a cloud environment. A Software as a Service (SaaS) cloud environment has been chosen for running DyOb-SLAM which has been described in chapter 6 section 6.3. The output results are also discussed in the section. The results are also compared to the ones resulting in the local computer.

5.2 Edge-Fog Cloud computing

Although cloud computing solves the requirement for heavy computation based functions in SLAM, a small disadvantage of cloud computing is the network latency. The network latency is observed in many systems which are based on object detection and/or surveillance. The output obtained from the cloud may have a latency for which a real-time system will be compromised. One solution for solving the network latency of cloud computing is using "Edge computing" and "Fog computing", where the data transfer is done in a relatively less distance from the user or source of data. Both Edge and Fog are enablers of data traffic to the cloud [79]. The difference between Edge computing and Fog computing is that Edge computing is data computation which occurs at the edge of the network in close proximity to the physical location of the source of data, while Fog computing acts like a mediator between the Edge and the Cloud for various purposes. If the Edge computing sends high stream of data directly to the cloud, the Fog computing layer can receive the data and manipulate according to the system's requirement.

Due to the network latency issue, some systems have considered Edge-computing and Fog-computing for real-time based systems. In [80], the object detector module which is an R-CNN model is divided into two components such that for each component, a single contiguous object detection model is obtained running in an edge-cloud environment for optimal performance in delivering a real-time based object detection system. EdgeLens [81]

is a fog-cloud based system where a deep learning model is deployed to obtain real-time based object detection. It consists of Fog devices like smart-phones or tablets which is connected to an Edge-cloud server Aneka [82] to distribute the object detection tasks across different Fog and cloud resources.

Edge and Fog cloud computing are currently found to be used in different SLAM systems as well. Edge-SLAM [83] demonstrated a split architecture of a Visual SLAM system where the tracking computation is done in the local device and the rest of the computation (local mapping and loop closing) are done in the Edge server. In [84], a Mesh algorithm based SLAM system is designed where an edge node layer is used for extracting the image feature data and the cloud server to manage the edge nodes as well as provide the global map. A visual SLAM system, *edge*SLAM, has been designed in [85] which consists of a semantic segmentation algorithm to enhance localization and mapping accuracy and offloads the entire computation in an Edge server to get a real-time based semantic SLAM system. The system is designed into two where the tracking and mapping are performed in the local device and the optimization and segmentation are done in the Edge server.

5.3 Cloud computing in DyOb-SLAM

To make DyOb-SLAM a better performing SLAM system compared to other systems, the idea is to add a cloud computing portion for the SLAM system. Since the proposed SLAM system consists of high GPU-required object detector modules, the system would require a super-computing environment other than any local computing system. For such reasons, the DyOb-SLAM system is run in a chosen super-computing environment based at the University of Oklahoma (OU). This is the OU Supercomputing center for Education and Research (OSCER) [86] which have been discussed in section 6.3. The main idea behind using the OSCER system is to evaluate the DyOb-SLAM system's performance in a cloud

environment and to understand if it performs better than its performance in a local based computing system.

The SLAM system is then planned to run in an Edge-Cloud based environment which have been discussed in the Chapter 7. Since the purpose of the SLAM system is to obtain a real-time based system, the latency issue that cloud computing usually faces can be solved if the system runs in an Edge environment. The system requires a fast and heavy computing SLAM performance if it is applied on a real life based scenario.

Chapter 6

RESULTS & DISCUSSION

The proposed system has been evaluated on the basis of camera pose, object motion, object speed and moving object tracking performance. At first the experiment is done in a local computer system. The experiment is done in an Intel Core i7-8700 CPU @ 3.20GHz × 12 system with a 4 GB GPU of NVIDIA Quadro P620 processor. Then the SLAM system is run in a cloud environment, which has been chosen to be the OSCER [86] environment. The OSCER system has been described in details in this chapter. The SLAM system, at present, is only applicable for outdoor scenes, mostly with scenes having objects like - car or any other vehicles. The main dataset that has been effective to evaluate the performance is the KITTI Tracking dataset [87].

6.1 KITTI Tracking Dataset

The KITTI Tracking Dataset [62] [87] is a dataset collection for the use of autonomous driving and mobile robotics research. The calibrated, synchronized and time-stamped dataset collection consists of real world traffic situations. The data are developed for stereo, optical flow, visual odometry/SLAM and 3D Object detection experiments.

The KITTI data sequence that was chosen for this experiment is *kitti-0000-0013*. This

dataset contains 55 sequences with RGB images (see figure 6.1), depth images, timestamps and ground truth information of camera and object pose. The sequences are of a simple scene where two cars are driving on a road.

Figure 6.1: RGB image of a frame of *kitti-0000-0013*

A settings (.yaml) file is set for the KITTI dataset where the parameters have been fixed for running the SLAM system. Firstly, the camera parameters like the camera calibration and distortion parameters (fx, fy, cx, cy), the camera frames per second, frame height and width are chosen according to the dataset which have not been altered. Since the dataset is comprised of RGB-D images, the depth parameters need to be set as well. The depth value has been set for the two different features - static and dynamic. The depth variables in the settings file, *ThDepth* is denoted as the depth value for the static features and *ThDepthObj* denotes the depth value for dynamic features. ThDepth has been originally (by the makers of KITTI dataset) set to 40.0 and ThDepthObj has been set to 25.0. **For the sake of the experiment the value of ThDepthObj has been set to 10.40**. These depth values help to differentiate the close and far points in the images. Besides, the depth map factor is set to 256, which is a scale factor that multiplies the input depthmap.

6.2 Experimental Data

6.2.1 Camera and Object Pose

The main comparison of performance of the DyOb-SLAM system is established with VDO-SLAM. The table in figure 6.2 shows the average of both Camera pose error and object pose error containing both translational and rotational errors. Each sequence has been run 5 times for both DyOb-SLAM and VDO-SLAM to locate the change in the non-deterministic output.

	CAMERA POSE ERROR (AVERAGE)				OBJECT POSE ERROR (AVERAGE)			
	Translation (t)		Rotation (R)		Translation (t)		Rotation (R)	
Number of iterations	DyOb-SLAM	VDO-SLAM	DyOb-SLAM	VDO-SLAM	DyOb-SLAM	VDO-SLAM	DyOb-SLAM	VDO-SLAM
01	0.077713	0.074721	0.054386	0.037299	0.103237	0.079002	0.364453	0.319280
02	0.075487	0.071230	0.044760	0.037202	0.087921	0.078896	0.417796	0.320715
03	0.075489	0.076412	0.051999	0.039815	0.093773	0.080748	0.418121	0.317692
04	0.075989	0.070711	0.044463	0.039258	0.082121	0.083999	0.481621	0.317329
05	0.077622	0.074034	0.050578	0.038402	0.089987	0.080106	0.398700	0.326143

Figure 6.2: Table of Comparison of errors for DyOb-SLAM and VDO-SLAM

Let the ground truth motion transform be denoted as $\mathcal{T}$, where $\mathcal{T} \in SE(3)$ and the estimated motion transform be denoted as $\mathcal{T}_{est}$. The pose error P will generally be calculated as -

$$P = \mathcal{T}_{est}^{-1}\mathcal{T} \tag{6.1}$$

At the different camera frames and time steps, the root mean squared errors (RMSE) for both camera pose and object motion pose are computed. After all the frames are processed, the end result is an average of all the RMS errors calculated in each frames. The average pose error is calculated for both camera and object motion.

6.2.2 Object Speed

For both DyOb-SLAM and VDO-SLAM systems, the object speed error has been evaluated. The linear velocity of each points in the object pixels are estimated. If the pose change is expressed as H and m_k be a point in the object pixel at frame k where $m = [m_x, m_y, m_z, 1]$, then the estimated velocity can be expressed as -

$$\mathcal{V}_{est} = m_k - m_{k-1} \tag{6.2}$$

$$\mathcal{V}_{est} = H_k m_{k-1} - m_{k-1} = H_k m_{k-1} - I_4.m_{k-1} \tag{6.3}$$

Here, I_4 is an identity matrix. The equation 6.2 states that the difference of the point coordinates from frame k-1 to k at the time step gives the estimated velocity. The equation 6.3 on the other hand simplifies equation 6.2 converting the new frame position of point as a function of the pose change matrix H.

The figure 6.3 shows a bar chart of the average object speed error for DyOb-SLAM and VDO-SLAM at the 5 iterations. If $\mathcal{V}$ is the ground truth speed information, the velocity error -

$$V_{err} = |\mathcal{V}_{est}| - |\mathcal{V}| \tag{6.4}$$

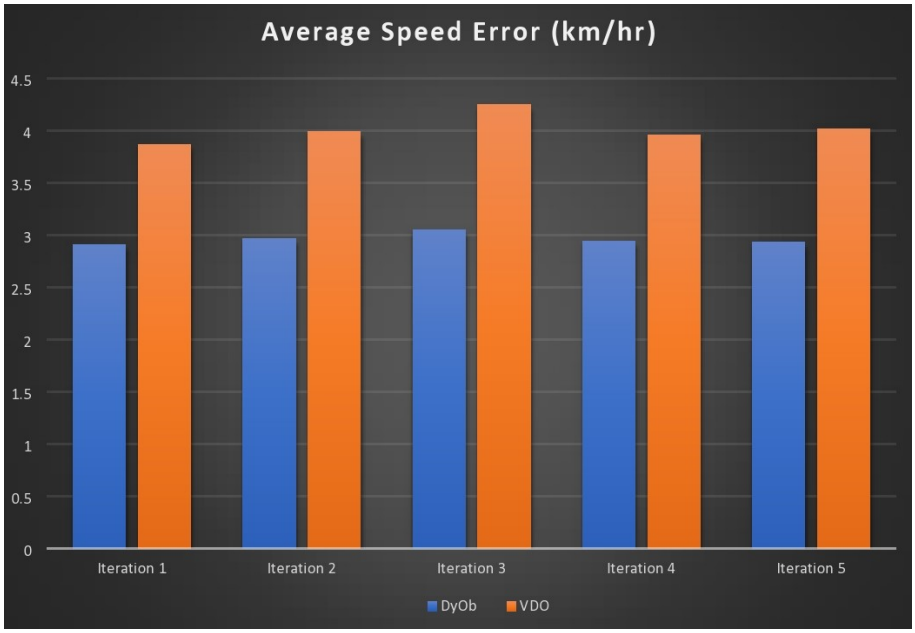

Figure 6.3: Average object speed error for DyOb-SLAM and VDO-SLAM

The object speed is calculated at the end of processing every frame. In figure 6.4 it is shown that the detected dynamic objects are bounded by bounding boxes and the speed of the objects are printed in the processed frame. The average of these errors are obtained as the output of the SLAM system.

Figure 6.4: Object Speed Calculation in each frame

6.3 OSCER

For better performance of DyOb-SLAM, the supercomputer of the University of Oklahoma, OSCER [86] has been selected as a cloud to run the system in. The OU Supercomputing center for Education and Research (OSCER) [86] is a platform made by The

University of Oklahoma (OU) Information Technology division where undergraduate and graduate students, faculties and staff can run advanced computing algorithms for science and engineering research and education purposes. It has a High Performance Computing (HPC) infrastructure which provides hardware and software resources, technology transfer support and outreach support. HPC [3] is a multi-processor environment where multiple jobs can run in a parallel way, i.e. can run multiple jobs at the same time. An overview of the HPC infrastructure of OSCER is shown in figure 6.5.

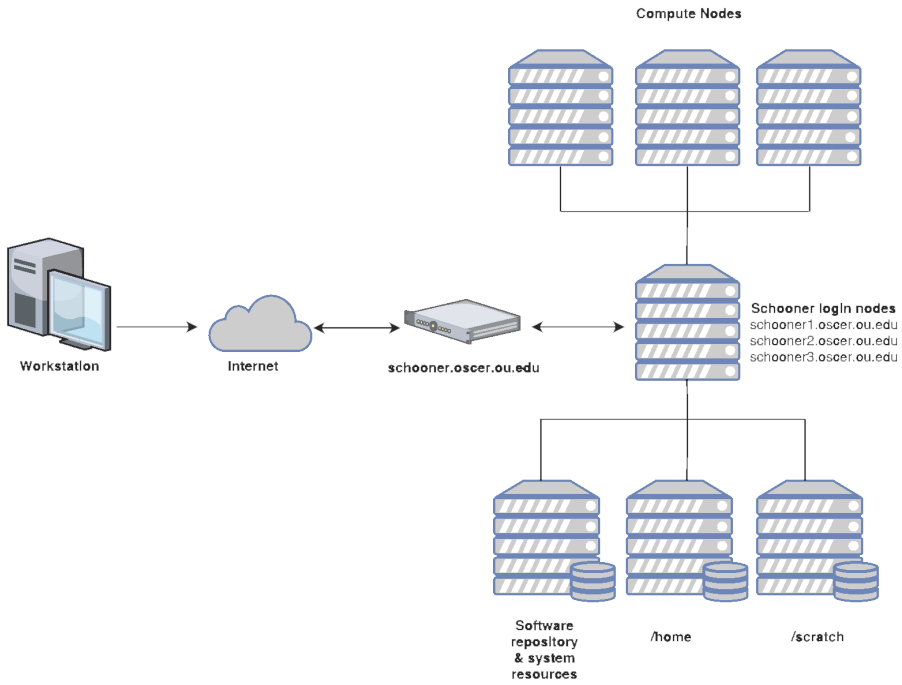

Figure 6.5: Infrastructure of OSCER [3]

OSCER was chosen as the cloud computing module for DyOb-SLAM. The idea is to at first run the system in it and see if the performance is better. The next step is discussed in the Future work chapter in chapter 7.

6.4 Discussion of Results

The key difference between VDO-SLAM and DyOb-SLAM is the semantic segmentation process in the system. For VDO-SLAM, the semantic segmentation is done in a

pre-processed way shaping the segmentation data according to the ground truth. On the other hand DyOb-SLAM directly uses a Mask-RCNN network which segments out the dynamic objects and later on the data is then processed (see figure 4.2). Due to this, the average pose and speed errors are different for the two SLAM systems.

6.4.1 Camera and Object Pose Error

As we can see from the Table 01 (figure 6.2), the proposed system DyOb-SLAM has slightly higher average pose error for both camera and object in every iterations. Nevertheless, both the systems have very low average pose error. It can be observed from the table that the camera translational error of DyOb-SLAM differ about a range of 0.003-0.005 than that of VDO-SLAM. The camera rotational error difference is seen to be about a range of 0.007-0.017 which is a bigger difference compared to the translational error difference. For object pose error, the translational error difference ranges about 0.009-0.03 and the rotational error difference ranges about 0.05-0.17. It is well observed that for DyOb-SLAM system the first iteration gave higher outputs than the next 4. It is also observed, for both camera and object poses, the rotational error is bigger compared to the translational error.

6.4.2 Object Speed Error

On the other hand, for the average object speed performance, it is observed that DyOb-SLAM performs better than VDO-SLAM which can be seen from figure 6.3. The object speed error for DyOb-SLAM gives a range of 0.96-1.20 more than the object speed error found from VDO-SLAM. The difference is very high and it can be concluded that the DyOb-SLAM system gives a satisfactory object speed error data.

6.4.3 Discussion

The settings file that has been obtained from the KITTI dataset has been modified for the experiment. **The Depth value for the objects have been lowered to 10.40, previously**

which was 25.0 . With Mask-RCNN's segmentation information, the objects could not be categorized as multiple. The scene contains different vehicles. The neural network segments out even the static prior outlier points. To solve the matter and to only segment out the required two cars that need to be tracked, the depth value has been compromised to a limit. As a result of this, the two focused cars in the frames can be identified as separate objects, instead of one. For this reason, the pose errors in the proposed system are a bit higher than that of VDO-SLAM. Another reason is the computational energy. If the computational energy of the two system is compared, the proposed system is more expensive than VDO-SLAM. But since the aim of DyOb-SLAM system is to produce a real time dynamic object tracking SLAM system in the future, this compromise can be done.

The speed of the detected objects is calculated when the objects are detected and segmented out. After the dynamic objects are labelled with an object ID, the bounding boxes along with the speed calculation appear as seen in figure 6.4. Due to focusing the two vehicles at the close point portion in the frames which have been discussed before, the object bounding boxes appear as per the segmented objects. This means, that when the detected objects are in close point depth scale, the bounding boxes along with the speed calculation appear accordingly. So, after the first car goes out of the depth scale, the speed of the car is not measured.

Chapter 7

LIMITATIONS & FUTURE WORK

The performance of the proposed SLAM system is not satisfactory compared to the closest similar system (VDO-SLAM) [6]. Although the pose error is very similar to that of VDO-SLAM, the performance could be better. One scope to improve the results is by changing the object detector module. Due to the nature of the pre-trained Mask-RCNN model used, the pose information of detected objects have major differences than the ground truth pose information. The object detector module needs to produce a more robust and precise output from the combination of the Convolutional Neural Network and the optical flow models. For that, the Mask-RCNN model firstly needs to produce instance labels for detected objects. The PWC-Net [56] model can also be directly used to process the optical flow data for each frames. The optical flow network should be fine-tuned as a result of which, a better optical flow map can be obtained giving precise information about the detected moving objects. However, this would increase the computation requirement which would further slow the entire system more. Due to this, the system will be needing a stronger platform to run so that the latency problem of running the system can be avoided and a real time system can be obtained. One way is by running the system using cloud computing technology.

The current DyOb-SLAM system uses the object detector in such a way that the output of the object detector is pre-processed according to the ground-truth information of the objects. The ground truth information is also used for evaluating the pose error. To produce a real-time based SLAM system, the ground-truth information cannot be executed in the system. As a result, the pre-processing of the semantic information obtained from the Mask-RCNN module will not be required. For this, the combination of the instance label based Mask-RCNN module and PWC-Net can directly be used for Tracking and Mapping of the system.

Since the DyOb-SLAM system gives less efficient estimation of pose and has a low computing time due to the Mask-RCNN processing, the system was intended to run in the OSCER [86] to see how the proposed SLAM system would work in a cloud environment. The purpose is to get a faster computational and more accurate SLAM system so that it can give more precise data in a fast manner. Due to lack of time and missing resources, the experiment could not be run in the OSCER system. Since the OSCER system is used by multiple users, the job running in the system takes time as each job is run in a serial way. Besides, while compiling the system in the OSCER environment, some pre-requisite modules for the DyOb-SLAM code were not present in the OSCER environment, as a result of which the compilation could not be successful. Without modules like - OpenCV's extra packages where optical flow assisting header files are present, the DyOb-SLAM system will not compile. There were a lot of pre-requisites to this module which took time to be loaded in the OSCER environment.

A real life based application for DyOb-SLAM is to use it particularly for search missions. Since the SLAM system can detect dynamic objects, this could be applied on Quadcopters or Unmanned Aerial Vehicles to locate the dynamic contents in its field of view. The

dataset on which DyOb-SLAM system has been experimented on is the KITTI [62] dataset which is based only on autonomous driving technology. The KITTI dataset only consists of ground based vehicles and terrestrial environment. The next step for DyOb-SLAM system will be focused on aerial datasets and to form an efficient system for unmanned aerial vehicles according to the required application.

The future idea is to run a specific portion of the DyOb-SLAM system in a cloud environment which can give real time accurate data to the core of the system. The system will be a combined form using Edge nodes for computation and cloud for storage. Since the object detector takes more time to process data, it could be deployed at the Edge node which can give highly processed output from the Edge computing. The cloud environment can store the output map results for an elongated period to be used later for obtaining a more precise global map of the environment. Cloud can also provide the infrastructure for necessary processing of the data. A network will be needed to be designed such that the processes running in the local system, edge nodes and cloud could function together to obtain the dynamic object tracking output in an online fashion.

Chapter 8

CONCLUSION

With such advancement in the world of robotics, more SLAM systems are being developed through time which include dynamic content in their processing algorithms. DyOb-SLAM is one of those types of SLAM systems which can include dynamic objects in its mapping and track the objects in every frames. With the help of advanced technologies like convolutional neural networks (Mask-RCNN [1]) and dense optical flow algorithms (PWC-Net [56]), dynamic contents can be accurately detected and used for further processing. The back-end of the DyOb-SLAM system can thoroughly give more optimized data for which an optimized sparse map of static keypoints in the environment can be obtained, along with a global map of camera and object trajectories with subsequent processing of each frame.

In the future, to obtain an even better SLAM processing system, a cloud environment can be used to divide the system into two and process data parallelly in the main processing unit and the cloud at the same time. In that way, the pose estimation error can be lowered and a faster driven output can be obtained.

Bibliography

[1] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[2] B. A. Erol, S. Vaishnav, J. D. Labrado, P. Benavidez, and M. Jamshidi, "Cloud-based control and vslam through cooperative mapping and localization," in *2016 World Automation Congress (WAC)*, 2016, pp. 1–6.

[3] "High Performance Computing at OU," https://www.ou.edu/oscer/getting_started/getting_started_hpc_intro, accessed: 2021.

[4] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[5] B. Bescós, J. M. Fácil, J. Civera, and J. Neira, "Dynaslam: Tracking, mapping and inpainting in dynamic scenes," *CoRR*, vol. abs/1806.05620, 2018.

[6] J. Zhang, M. Henein, R. E. Mahony, and V. Ila, "VDO-SLAM: A visual dynamic object-aware SLAM system," *CoRR*, vol. abs/2005.11052, 2020. [Online]. Available: https://arxiv.org/abs/2005.11052

[7] Y. Chen, Y. Chen, and G. Wang, "Bundle adjustment revisited," *CoRR*, vol. abs/1912.03858, 2019. [Online]. Available: http://arxiv.org/abs/1912.03858

[8] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," pp. 1–10, 2007.

[9] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *CoRR*, vol. abs/1502.00956, 2015. [Online]. Available: http://arxiv.org/abs/1502.00956

[10] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," *CoRR*, vol. abs/1610.06475, 2016. [Online]. Available: http://arxiv.org/abs/1610.06475

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," vol. 11, pp. 127–136, 2011.

[12] J. Engel, T. Sch¨ops, and D. Cremers, "Lsd-slam: Large scale direct monocular slam," p. 834–849, 2014.

[13] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," vol. 29, June 2007.

[14] S. Sumikura, M. Shibuya, and K. Sakurada, "Openvslam: a versatile visual slam framework," pp. 2292–2295, 2019.

[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," p. 2564–2571, 2011.

[16] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "Elasticfusion: Dense slam without a pose graph," *The International Journal of Robotics Research*, vol. 35(14), p. 1697–1716, 2016.

[17] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 611–625, April, 2017.

[18] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 30(6), pp. 1309–1332, 2016.

[19] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "An overview to visual odometry and visual slam: Applications to mobile robotics," *Intelligent Industrial Systems*, vol. 1(4), p. 289–311, 2015.

[20] K. Tateno, F. Tombari, I. Laina, and N. Navab, "Cnn-slam: Real-time dense monocular slam with learned depth prediction," July, 2017.

[21] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," May, 2011.

[22] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," 2012.

[23] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (slam): Toward exact localization without explicit localization," *IEEE Transactions on Robotics and Automation*, vol. 17(2), pp. 125–137, April, 2001.

[24] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual slam maps," May, 2018.

[25] 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," June, 2013.

[26] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," 2017.

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[28] F. Zhong, S. Wang, Z. Zhang, C. Chen, and Y. Wang, "Detect-slam: Making object detection and slam mutually beneficial," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 1001–1010.

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," June, 2016.

[30] L. Zhang, L. Wei, P. Shen, W. Wei, G. Zhu, and J. Song, "Semantic slam based on object detection and improved octomap," *IEEE Access*, vol. 6, October, 2018.

[31] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," June, 2014.

[32] R. Girshick, "Fast r-cnn," December, 2015.

[33] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[34] M. Rünz and L. Agapito, "Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects," *CoRR*, vol. abs/1804.09194, 2018. [Online]. Available: http://arxiv.org/abs/1804.09194

[35] R. Hachiuma, C. Pirchheim, D. Schmalstieg, and H. Saito, "Detectfusion: Detecting and segmenting both known and unknown dynamic objects in real-

time SLAM," *CoRR*, vol. abs/1907.09127, 2019. [Online]. Available: http://arxiv.org/abs/1907.09127

[36] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[37] E. Royer, M. Lhuillier, M. Dhome, and T. Chateau, "Localization in urban environments: monocular vision compared to a differential gps sensor," vol. 2, 07 2005, pp. 114– 121 vol. 2.

[38] Z. Zhang and Y. Shan, "Incremental motion estimation through modified bundle adjustment," in *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, vol. 2, 2003, pp. II–343.

[39] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, 2006, pp. 363–370.

[40] J. Michot, A. Bartoli, and F. Gaspard, "Bi-objective bundle adjustment with application to multi-sensor slam," 01 2010.

[41] K. Konolige and M. Agrawal, "Frameslam: From bundle adjustment to real-time visual mapping," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1066–1077, 2008.

[42] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard, "Efficient estimation of accurate maximum likelihood maps in 3d," 10 2007, pp. 3472 – 3478.

[43] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2d and 3d mapping," 05 2010, pp. 273–278.

[44] M. Kaess, A. Ranganathan, and F. Dellaert, "isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008.

[45] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2(4), pp. 31–43, 2010.

[46] G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Rs-slam: Ransac sampling for visual fastslam," September, 2011.

[47] S. Yang and S. A. Scherer, "CubeSLAM: Monocular 3d object detection and SLAM without prior models," *CoRR*, vol. abs/1806.00557, 2018. [Online]. Available: http://arxiv.org/abs/1806.00557

[48] J. Huang, S. Yang, T. Mu, and S. Hu, "Clustervo: Clustering moving instances and estimating visual odometry for self and surroundings," *CoRR*, vol. abs/2003.12980, 2020. [Online]. Available: https://arxiv.org/abs/2003.12980

[49] I. Ballester, A. Fontan, J. Civera, K. H. Strobl, and R. Triebel, "Dot: Dynamic object tracking for visual slam," 2020.

[50] B. Bescós, C. Campos, J. D. Tardós, and J. Neira, "Dynaslam II: tightly-coupled multi-object tracking and SLAM," *CoRR*, vol. abs/2010.07820, 2020. [Online]. Available: https://arxiv.org/abs/2010.07820

[51] P. Li, T. Qin, and S. Shen, "Stereo vision-based semantic 3d object and ego-motion tracking for autonomous driving," *CoRR*, vol. abs/1807.02062, 2018. [Online]. Available: http://arxiv.org/abs/1807.02062

[52] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," May, 2013.

[53] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "Davinci: A cloud computing framework for service robots," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 3084–3089.

[54] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009.

[55] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[56] D. Sun, X. Yang, M. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," *CoRR*, vol. abs/1709.02371, 2017. [Online]. Available: http://arxiv.org/abs/1709.02371

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[58] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016. [Online]. Available: http://arxiv.org/abs/1612.03144

[59] C. Chen, S. Ye, H. Chen, O. V. Nedzvedz, and S. V. Ablameyko, "Integral Optical Flow and its Application for Monitoring Dynamic Objects from a Video Sequence," *Journal of Applied Spectroscopy*, vol. 84, no. 1, pp. 120–128, Mar. 2017.

[60] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," *CoRR*, vol. abs/1512.02134, 2015. [Online]. Available: http://arxiv.org/abs/1512.02134

[61] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *ECCV*, 2012.

[62] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[63] R. Szeliski, *Computer vision algorithms and applications*. London; New York: Springer, 2011. [Online]. Available: http://dx.doi.org/10.1007/978-1-84882-935-0

[64] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment — a modern synthesis," in *Vision Algorithms: Theory and Practice*, B. Triggs, A. Zisserman, and R. Szeliski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 298–372.

[65] M. Lourakis and A. Argyros, "Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment?." vol. 2, 01 2005, pp. 1526–1531.

[66] D. Galvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[67] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," vol. 3951, 07 2006.

[68] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," vol. 6314, 09 2010, pp. 778–792.

[69] L. Paz, P. Pinies, J. Tardos, and J. Neira, "Large scale 6dof slam with stereo-in-hand," *Robotics, IEEE Transactions on*, vol. 24, pp. 946 – 957, 11 2008.

[70] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[71] S. Kamburugamuve, H. He, G. Fox, and D. Crandall, "Cloud-based parallel implementation of slam for mobile robots," 03 2016.

[72] S. Kamburugamuve, L. Christiansen, and G. Fox, "A framework for real time processing of sensor data in the cloud," *Journal of Sensors*, vol. 2015, pp. 1–11, 04 2015.

[73] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2tam: A cloud framework for cooperative tracking and mapping," *Robotics Auton. Syst.*, vol. 62, pp. 401–413, 2014.

[74] "Zoho for enterprise," https://www.zoho.com/, accessed: 2021.

[75] "Google app engine," https://code.google.com/, accessed: 2021.

[76] "Amazon elastic compute cloud (amazon EC2)," https://aws.amazon.com/pm/ec2/, accessed: 2021.

[77] Z. Luo, A. Small, L. Dugan, and S. Lane, "Cloud chaser: Real time deep learning computer vision on low computing power devices," *CoRR*, vol. abs/1810.01069, 2018. [Online]. Available: http://arxiv.org/abs/1810.01069

[78] V. K. Sarker, J. Peña Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Offloading slam for indoor mobile robots with edge-fog-cloud computing," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, 2019, pp. 1–6.

[79] "Understanding Edge Computing vs. Fog Computing," https://www.onlogic.com/company/io-hub/edge-computing-vs-fog-computing/, accessed: 2021-11-12.

[80] J. Herrera, M. A. Demir, P. Yousefi, J. J. Prevost, and P. Rad, "Distributed edge cloud r-cnn for real time object detection," in *2018 World Automation Congress (WAC)*, 2018, pp. 1–5.

[81] S. Tuli, N. Basumatary, and R. Buyya, "Edgelens: Deep learning based object detection in integrated IoT, fog and cloud computing environments," in *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, 2019, pp. 496–502.

[82] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: A software platform for .net-based cloud computing," *CoRR*, vol. abs/0907.4622, 2009. [Online]. Available: http://arxiv.org/abs/0907.4622

[83] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 325–337. [Online]. Available: https://doi.org/10.1145/3386901.3389033

[84] W. Zhao, X. Wang, B. Qi, and T. Runge, "Ground-level mapping and navigating for agriculture based on iot and computer vision," *IEEE Access*, vol. 8, pp. 221 975–221 985, 2020.

[85] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual SLAM," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1828–1837.

[86] "OU Supercomputing Center for Education & Research," https://ou.edu/oscer, accessed: 2021-14-10.

[87] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231 – 1237, Sep. 2013.