CLASSIFIED OBJECT LOCALIZATION IN SLAM

AND LOOP CLOSURE THROUGH REINFORCEMENT LEARNING

by

Asif Iqbal

APPROVED BY SUPERVISORY COMMITTEE:

Nicholas Gans, Chair

Nasser Kehtarnavaz

Carlos Busso-Recabarren

Mehrdad Nourani

Ryan McMahan

Dedicated to my teachers and parents

CLASSIFIED OBJECT LOCALIZATION IN SLAM AND LOOP CLOSURE THROUGH REINFORCEMENT LEARNING

by

ASIF IQBAL, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

December 2019

ACKNOWLEDGMENTS

First and foremost, I would like to thank and express my utmost gratitude to my PhD advisor, Dr. Nicholas R. Gans. He has provided me extensive guidance and excellent mentorship for the past six years at UTD for my MS and PhD programs. I would also like to thank my PhD committee members, Dr. Nasser, Dr. Busso, Dr. Nourani and Dr. McMahan. Their insightful comments and suggestion were greatly helpful in fulfilling my PhD dissertation. The courses I took with Dr. Nasser, Dr. Busso and Dr. Gans have also immensely helped me during my research. I am also indebted to all the people at Texas Instruments for supporting me.

I like to thank all the SeRViCE Lab members for helping and advising me during my graduate studies. Yingmao, Pablo, Kaveh, Jinfu, Bashir, and Yujie are a few friends I am indebted to for all of the valuable time and knowledge they have given me.

Lastly, I am thankful to my beloved parents, Zakia Yousuf and Yousuf Ali Mridha, living in Bangladesh. They have been patient and forbearing during my long PhD studies and have given me constant courage and strength to move forward in my life.

November 2019

CLASSIFIED OBJECT LOCALIZATION IN SLAM

AND LOOP CLOSURE THROUGH REINFORCEMENT LEARNING

Asif Iqbal, PhD
The University of Texas at Dallas, 2019

Supervising Professor: Nicholas Gans, Chair

Maps generated by many visual Simultaneous Localization and Mapping (SLAM) algorithms consist of geometric primitives such as points, lines or planes. These maps offer a topographic representation of the environment, but they lack semantic information about the scene and objects in the environment. Object classifiers leveraging advances in machine learning are highly accurate and reliable, capable of detecting and classifying thousands of objects. Classifiers can be incorporated into a SLAM pipeline to add semantic information to a scene. Frequently, this semantic information is conducted for each frame of the image, but semantic labeling is not persistent over time. Another element of SLAM is loop closure, which determines previously visited locations in the trajectory generated during the mapping and localization process. Identifying these loops in the trajectory is challenging due to the changes of viewing angles, illumination and environmental dynamics etc.

In this dissertation, we introduce two novel approaches to address these problems. First, we present a non-parametric statistical approach to perform association/matching between detected objects over consecutive image frames. An unsupervised clustering method then localizes these associated classified objects in accrued map. We test our approach on several public data sets and our own data-set, which shows promising results in terms of objects correctly associated from frame to frame and localization of the existing objects in the map.

We also have tested our algorithm on three data sets in our lab environment using tag markers to demonstrate the accuracy of classified object localization process. Second, we present a solution to the loop closure problem based on deep reinforcement learning. The framework is a reward-driven optimization process to learn loop closure detection. We demonstrate the framework in a simulated grid environment that generates data for a learning agent. The agent learns from data to perform loop closure in different environments. We demonstrate our results based on the rewards from the simulation and correct loop closure detection, and show that our outcomes are comparable to traditional loop-closure methods.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The consistent probabilistic mapping of an environment is a fundamental problem in robotics. The origin of this problem occurred when probabilistic methods were introduced into robotics and artificial intelligence. The solution to this problem was coined Simultaneous Localization and Mapping (SLAM) [26]. This is the problem of localizing a mobile robot moving through an unknown environment while building a distinct map of the surroundings. SLAM is an important research topic, with numerous notable contributions. Excellent overviews with extensive references were provided [95], Durrant-Whyte and Bailey [26], and others. Even though SLAM has been studied and researched over many years, the complete practical and perceptual representation of maps remains of huge interest to all. Vision-based SLAM (VSLAM) uses cameras as the primary sensor (e.g., [31, 35]). An important area of current research is adding semantic information in the environment map, which can be used for navigation, localization, retrieval, etc.

Most SLAM algorithms generate a map consisting of estimated geometric features such as points, lines or planes. These maps do not include semantic meaning or information [10, 15]. We seek to add semantic information to maps in the form of labeled, persistent objects present in the environment, such as furniture, office equipment, kitchen items, etc. It is not sufficient for objects to be detected and classified in an image, but objects must be matched/associated with the correct objects in previous images over time. This remains a difficult problem in SLAM and object classification.

Loop closure is another import element of SLAM, which indicates a return to a previously visited location in the SLAM trajectory. Loop closure also has been attempted using extracted visual features to search and locate a matching location in the previous observations from the sensor. This requires storing previously observed visual features, which may require large amount of memory as the map becomes larger.

This dissertation addresses these aforementioned problems. We present a solution using a vision based system to localizing objects in the scene while simultaneously localizing and building a map. No specific information about the objects is needed in advance, in contrast with recent approaches that require specific models of objects. We also provide a solution to loop closure problem using reinforcement learning method in a simulated environment. To achieve this, we leverage several already developed methods such as deep machine learning, clustering and reinforcement learning.

1.1 What is SLAM?

Simultaneous Localization and Mapping (SLAM) is the problem of localizing a robot moving through an unknown environment while building a distinct map of the surroundings. SLAM has been heavily researched in the last few decades. It has made its succession from theoretical development to practical implementation in recent years. SLAM initially was thought of as either a localization or a mapping problem. The breakthrough happened after the realization of SLAM as combined localization and mapping process. The solution to the SLAM problem is the estimation of motion between observations and a generalization of the observations into a map representation. This solution has been regarded as one of the most prestigious successes of the robotics community, and it has paved the way for fully autonomous robots.

SLAM has been studied dating back to 1986 and continues into the current period. A thorough analysis of the initial SLAM problem is given by Durrant-Whyte and Bailey in [26, 5]. In its formative period from 1986 to 2004, SLAM was introduced as a probabilistic problem, which included approaches from Kalman Filter, Particle filter and maximum likelihood estimation. The later years focused on improving observability, convergence and use of different type of sensors. Recent years, we have seen the introduction of efficient SLAM solvers such as g2o [50] and GTSAM [24]. There are many existing SLAM solutions based on

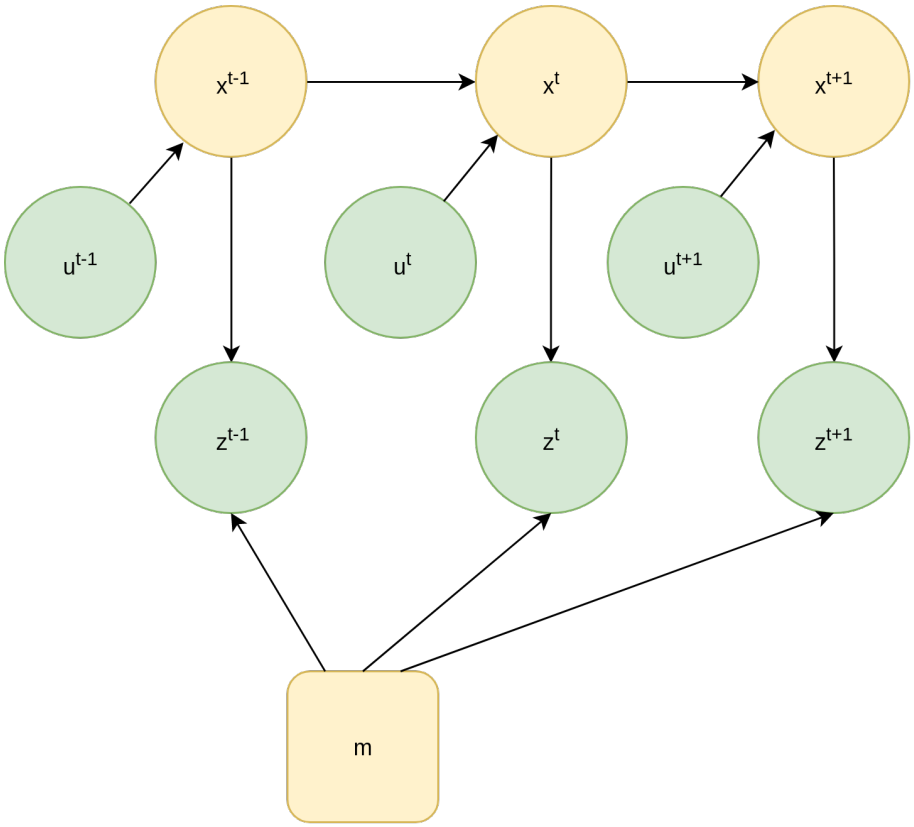


Figure 1.1: A generalization of the SLAM problem. The green circles are observed states and the yellow are hidden states.

the type of map representation, sensor, algorithm etc. The prominent list of SLAM methods includes EKF SLAM, Fast SLAM, GraphSLAM, Occupancy Grid SLAM, ORB SLAM, etc. Figure 1.2 shows several output of existing SLAM solutions.

The structure of the SLAM problem consists of many parts, but mainly there are two parts that require probabilistic modeling. These are an observation and motion model. The other essential parts may include *Landmark Extraction, Data Association, State Estimation and Update and Loop Closure.*

SLAM, in general terms, is seen as a chicken or egg problem. The localization and mapping are both needed to solve the problem. A map is needed for the localization problem, and pose estimate of the required for the mapping problem. SLAM is primarily defined using a probabilistic model. Given a robot's control or odometry sequence, $U^T = \{u^1, u^2, ..., u^t\}$ and

measurements from sensor, $Z^T = \{z^1, z^2, ..., z^t\}$, SLAM aims to find the map of the environment, m, and a sequence of the robot's path or locations in the map, $X^T = \{x^1, x^2, ..., x^t\}$. Now, the *full SLAM* problem requires the following probability distribution be computed repeatedly,

$$p(X^T, m|Z^T, U^T). \tag{1.1}$$

This means the *full SLAM* is trying to calculate the joint posterior probability over X^T and m from observed states, U^T and Z^T. Generally, this estimation has been demonstrated with Expectation Maximization, extended Kalman filter, etc. The existing methods using *full SLAM* solutions tends to be offline and process data in batches. The other solution is to use *online SLAM* solution which is defined as,

$$p(x^t, m|Z^T, U^T). \tag{1.2}$$

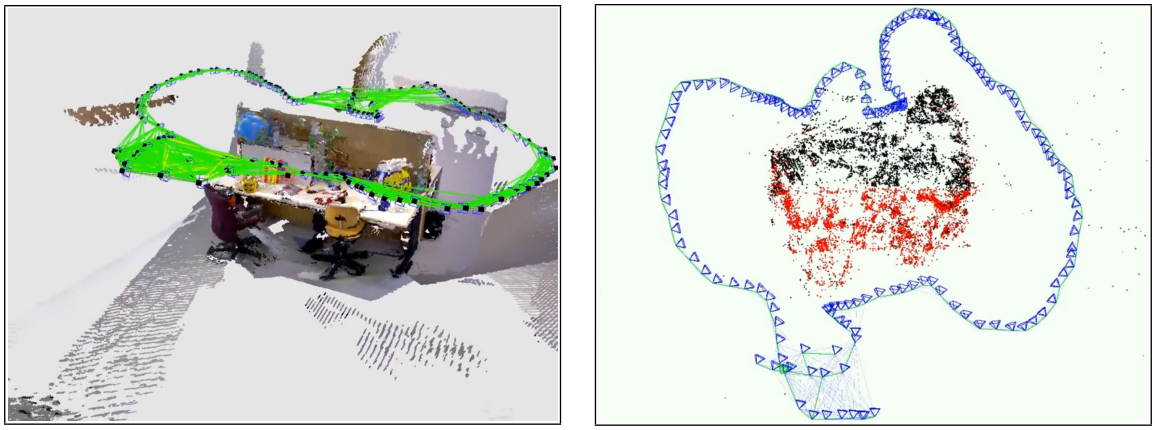
Online SLAM tries to estimate the current location instead of the full sequence of locations. These methods are incremental and can be performed in real time. Our proposed method in this dissertation is based on *online SLAM*. Figure 1.1 shows the a general SLAM problem. It shows the connection between the variables u, x, z and m.

To solve either SLAM problem, we need to model two distributions mentioned earlier, the observation and motion model. This can be described in the following form,

$$\begin{aligned} \text{Observation Model:} \quad & p(z^t|x^t, m) \\ \text{Motion Model:} \quad & p(x^t|x^{t-1}, u^t) \end{aligned} \tag{1.3}$$

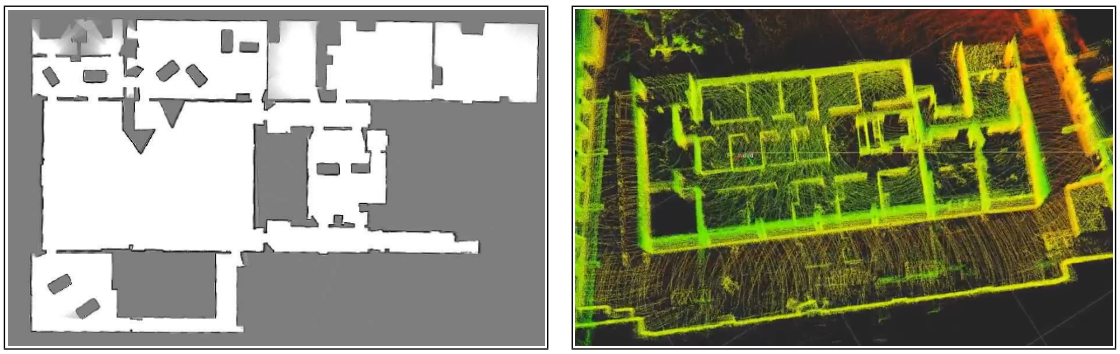
1.2 What is Classified Object Localization in SLAM?

An essential part of SLAM is landmark observation and establishment. A consistent, full solution to the combined localization and mapping problem would require a joint state composed of the vehicle pose and every landmark position. In theory, the trajectory of



(a) Direct SLAM [28]

(b) ORB SLAM [71]



(c) Occupancy Grid SLAM [42]

(d) Lidar SLAM [104]

Figure 1.2: The figure shows output of various SLAM solutions.

the observation sensor and location of all landmarks are calculated without the need for any a prior knowledge of environment. In practice, some solutions of SLAM have known or prior knowledge about landmarks. Landmarks can be geometric features like extracted image features, walls, corners, tag markers, etc. Although the localization process essentially requires pose estimation, which can be estimated with a limited number of feature points, a complete map of the environment requires a large number of features.

Geometric feature-based SLAM approaches generally build maps containing geometric and physical information but lack semantic information of the scene such as difference. SLAM based on geometric features have matured to an extent where a precise, large-scale map can be generated [70]. However, recognizable objects in the environment cannot be infused in

the map using most visual feature based SLAM systems. In Figure 1.2 the output of the SLAM solution resembles the environment, but it does not incorporate any of information about any objects in it.

Such objects can be now classified with high accuracy in an image using deep convolutional neural network (CNN). This object is localized in an image frame, but the location in the map remains unknown. Also the existence of multiple similar objects in sequential images need to be distinguished, which an object detection module does not solve. This dissertation demonstrates the procedure of localizing the classified objects in the map.

1.2.1 Related Work

There is notable research interest in semantic mapping of environments [9, 10, 15]. Most of these works use SLAM as a process to leverage the position and trajectory of a sensing element and object centric semantic information is generally collected from a another process such as object detection, 3D object database, etc. The object centric information also needs a data association process for it to be added in SLAM. Data association for detected objects has been attempted in several recent SLAM solutions [9, 66]. These methods formulated the data association as a optimization problem over different state probabilities extracted from sensor and object detection. *In contrast*, we perform data association in a continuous fashion using a non-parametric statistical method.

Pillai et al. [75] presented an object recognition system that uses SLAM to provide consistent object proposals over consecutive frames. They showed that data association for objects in a map can be solved given a map and known robot poses [66, 75]. Salas-Moreno et al. [80] proposed an SLAM method that can match objects in the scene to a database of 3D models and establish them as landmarks. The 3D model database must be manually established prior to execution. Figure 1.3 shows the results of SLAM++ where the matched chair models were placed in the map. *In contrast*, our approach can detect and localize previously unknown objects without knowledge of the full robot pose or map.

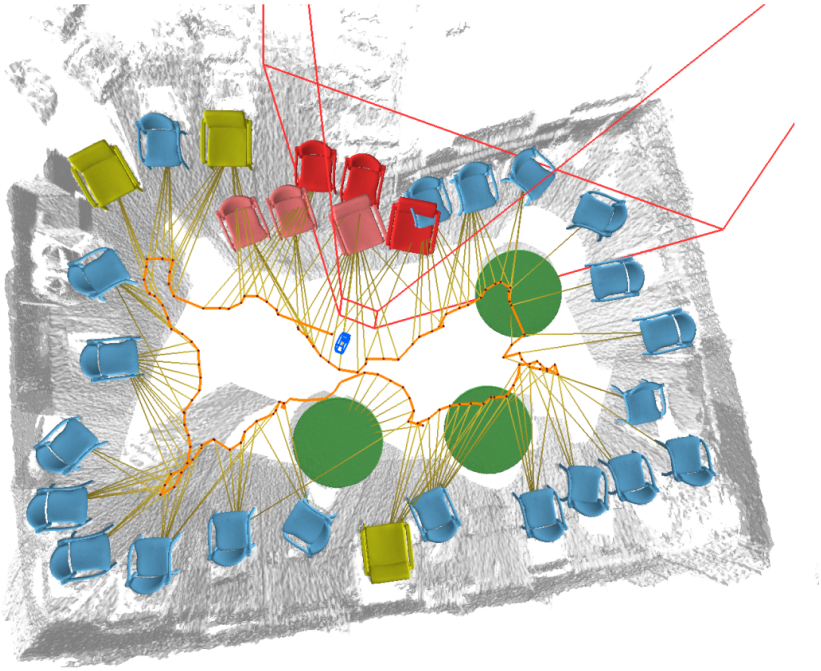


Figure 1.3: SLAM++ result [80]

Nicholson et al. [74] proposed a object oriented QudaraticSLAM where object landmarks are parameterized as constrained dual quadratics. Their method has shown how dual quadratics can be combined with object detection to integrate 3D landmark locations within the map in SLAM. Caste et al. [11, 12] presented an approach that can recognize known planar objects and localize them in the map; however, it is restricted to planar environments. SemanticFusion [59] demonstrated a dense semantic 3D reconstruction using ElasticFusion SLAM and CNNs. These works add semantic labels to each point in the map after creation, but this approach is not object-centric. Integrating recognized objects in SLAM was performed in [27, 103]. Figure 1.4 shows the SemanticFusion result in an indoor environment, where detected objects are localized as highlighted in the map. Atanasov et al. [9] separately addressed pose and data association optimization as separate problems in a SLAM pipeline. *In contrast*, the approach in this dissertation utilizes general purpose visual object detection and classification methods (e.g., CNNs or histogram of gradients methods) and localizes them in the map using statistical data association and unsupervised clustering analysis.

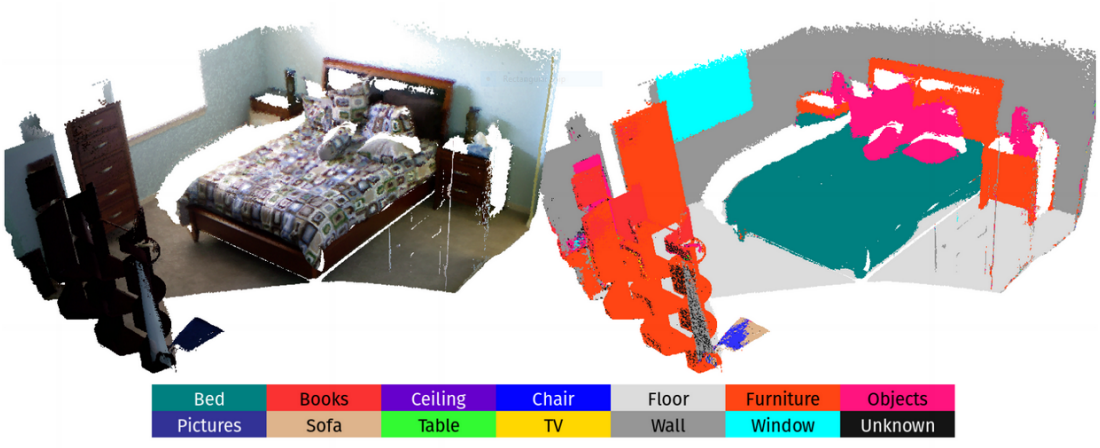


Figure 1.4: SemanticFusion result [59]

Ataer-Cansizoglu et al. [4] presented an object detection and tracking framework that jointly runs with an RGB-D SLAM system. They introduced the concept of hierarchical feature grouping which uses segments to represent object in map. Objects in this process were detected through appearance matching between the segments and existing object map. *In contrast*, we rely on object classification detection module to provide detected regions of object. They demonstrated their result to successfully grasp the objects from different viewpoints.

Grinvald et al. [36] proposed a volumetric object-centric mapping process. It uses geometric depth segmentation and object detection masks to provide consistent object labels. This information is later fused in global volumetric map. Choudhary et al. [14] proposed object discovery and modeling in SLAM where they utilized aggregated per frame segmentation matching with previous and current estimate to produce a final form. Later it gets refined and added as constraints to the map. *In contrast*, we match the depth samples of detected objects using a non-parametric statistical process. The associated samples over time gets added to the map through a clustering process.

Bowman et al. [9] introduced probabilistic data association, where semantic information was included in the optimization step. The SLAM was formulated in EM method with

additional information from classes and semantic measurements of objects. *In contrast*, our method uses SLAM in parallel to perform data association, localization of objects and later these information can be infused in SLAM.

1.2.2 Our Contributions

This dissertation addresses these aforementioned challenges by extracting and localizing objects in the scene while simultaneously localizing and building a map. No specific information about the objects is needed in advance, in contrast with recent approaches that require specific models of objects. To achieve this, we leverage deep machine learning methods available via deep neural network toolboxes e.g. Tensorflow [1], Torch [17], and Caffe [45]. However, deep learning classifiers excel at detection from a single image, and are not suited for associating or matching the same object in multiple images, such as from a video stream. Our approach tackles this problem through data association of multiple detected and classified objects. Another problem is to find the unknown number of exiting objects in the environment and the localization of these objects in the mapping process. The solutions to both these problems are initiated with no prior knowledge about any of the existing objects or its surrounding environment. The only assumption is the objects in the environment are not dynamic and the environment itself does not have dynamic elements to distort SLAM process. The environment can have moving objects but the locations of these objects cannot be estimated using our proposed method.

In this dissertation, we propose three primary contributions:

- We establish that non-parametric statistical methods can perform association of detected objects in consecutive images based on the distribution of depth data for object. Later, we use depth prediction whenever there is large motion in between two images. This helps data association remain applicable after a sudden frame loss or motion.

- We provide an unsupervised clustering process to localize objects in a map of an environment and incorporate the aforementioned non-parametric data association to reduce the complexity of the clustering process in SLAM.

- We evaluate our proposed method on public datasets and our own dataset to show the significant results. Our dataset will be made available to the public to provide ground truth data for future research in object localization in SLAM.

The proposed method in this dissertation is dependent on VSLAM and RGBD data. We leverage the pose estimation, mapping and loop closure of SLAM to achieve the outcome of this method. Figure 2.10 shows a result of our process on a RGB-D dataset, and Figure 2.1 shows the flowchart of our proposed method.

1.3 What is Loop Closure?

Loop closure (LC) is used in SLAM to indicate a return to a previously visited location and help correct drift and re-localization in the trajectory estimation [2, 88]. Successful loop closure to requires a high probability that the robot or agent will identify two identical locations and improve the overall accuracy of localization and mapping. The accuracy of loop closure detection is also crucial, as a single incorrect loop closure decision can easily disrupt the mapping process. The most difficult elements of loop closure is not just the detection of a loop in the trajectory but finding the similarity with previous locations in the map. This has been a challenging problem in SLAM, because the appearance or observations of a place or location can differ due to viewing angle, illumination, weather and dynamics of the environment. Figure 1.5 shows a before and after demonstration of a loop detection and correction in SLAM [40].

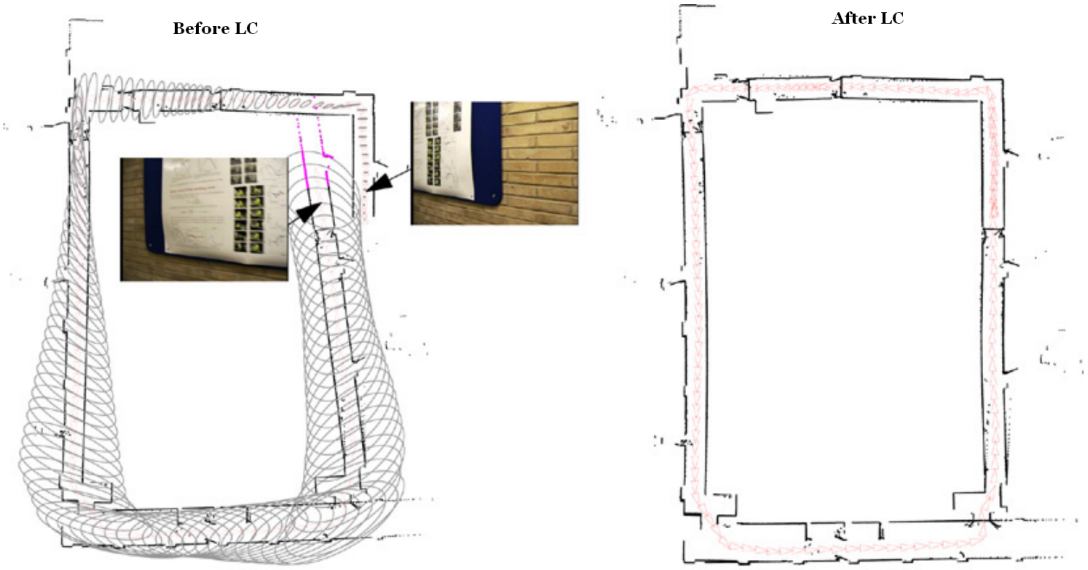


Figure 1.5: A demonstration of Loop Closure detection and correction [40].

1.3.1 Related Work

Loop closure in SLAM has already been attempted, with several established solutions such as EKF-SLAM, Monte Carlo localization, VSLAM, etc. [16, 101]. The idea behind most of these implementations is to perform loop closures using data association that matches visually salient features with current and previous observations. The same features that were used in localization and mapping can be used for loop closure detection. These methods can be applied with SLAM in online or offline fashion.

Offline methods require a database of images acquired beforehand. Kosecka et al. [47] and Ulrich et al [97] use maximum likelihood estimation with voting method to find matches in a pre-built image database of the target location. The voting method requires steep computation of pair matching for most the likely hypothesis, and the maximum likelihood is not proper for these multiple hypotheses scenario[2]. Krose et al. [49] reduced the dimension of images of the map through principal component analysis and then performed probabilistic localization on the appearance model. Ramos et al.[76] demonstrated a similar

approach combining Bayesian learning to represent places. These methods used a global feature descriptor, which is not robust against image effects such as illumination and perspective changes, rotation, scaling, etc. *In contrast*, our method relies on reinforcement learning using deep convolutional neural network to learn the features to use in loop closure detection.

Advances in computer vision led to the introduction of local feature descriptor, which is focused on different regions of the image rather the whole image. The most popular local feature descriptor in use are SIFT [57], ORB [79], SURF [6], HOG [22], etc. The visual feature based models use image as a text-based representation known as bag-of-words [105, 18]. It models the image like a text documents, where a word represents a region in the space of invariant descriptors and stores them in a tree structure for fast real-time search operations. It also requires a dictionary, which is built beforehand by clustering indistinguishable visual descriptors from a training dataset. This method has been used in [100] and [73] for localization in map and loop closure detection. Cummins et al. [19] proposed a method to estimate the similarity between two observation of the same location. This model performed loop closure with linear complexity for the number of locations in the map. Angeli et al. [2] proposed an online method of loop closure using bag-of-words and Bayesian filtering. Cumming et al. [20] introduced a probabilistic navigation based on mapping named FAB-MAP. They used bag-of-words and Recursive Bayes model to demonstrate online loop closure on a very large scale dataset consisting of 2km length path. *In contrast*, our proposed method assumes a fixed number of loop closures in the environment and uses a reinforcement learning method to learn loop closure detection at the the pre-determined locations. The existing methods relies on storing extracted features as areas of a map explored. This storage of this features can become problem if the map is large. In our proposed method, we specify the memory required for the trained loop closure policy beforehand. This way, once the model is trained, we do not need to add anymore information to the loop closure process, and it only needs sensor data to perform loop closure in the known map.

Recent advances in machine learning have introduced new methods in navigation and mapping. Convolutional neural networks have been used in place of feature descriptors. Sünderhauf et al. [90] introduced ConvNet Landmark, where combined CNN and region-based features were used to match and identify landmark proposals for loop closure, but it was computationally expensive to deploy in real-time. Chen et al. [13] generated a large place-recognition dataset with appearance changes and interpreted the problem as an image classification task. Arandjelovic et al. [3] proposed a novel place recognition framework named NetVLAD, which is trained on weakly supervised Google Street View images on geotags. Gao et al. [32] proposed a solution to loop closure using stacked denoising auto-encoder (SDA). They retrieved a similarity score for loop closure using raw images through unsupervised learning. Their method is susceptible to parameter tuning, and the method was applicable to offline setting. Mirowski et al.[62] demonstrated a deep RL algorithm in a simulated 3D environment where the agents were able perform navigation and loop closure tasks with high accuracy. This method is focused on simulated environments and has not been tested on real world environments. *In contrast*, our proposed method relies on deep CNN to extract features from the map for loop closure detection and are trained on our own simulated environment, which is designed based on indoor environment. Our CNN is not employed on the sensor data which can be expensive to extract features.

1.3.2 Our Contributions

Our contribution in this dissertation is the use of deep reinforcement learning in loop closure. Our implementation has similarity to [62] and [65], which also showcased a deep RL method to learn loop closure in a simulated environment. Our implementation is primarily focused on the loop closure solution using a learning method. We also provide an improvement over [63] to include an optimization technique for varying batch size each episode, so entropy of actions in the batch is maximized.

In this dissertation, we are proposing the following contributions,

- We show that loop closure problem can be solved using deep reinforcement learning with some assumption and prior knowledge about the environment. We demonstrate entropy maximization for sampling trajectories for sparse actions
- A simulated grid environment is demonstrated for training loop closure in a reinforcement learning framework
- We also demonstrate how our object localization module can incorporated in loop closure module.

1.4 Outline of the Dissertation

The remainder of the dissertation has been organized in two chapters to describe the two research topics. In Chapter 2, we describe data association and object localization procedure, and in Chapter 3, we describe loop closure using deep reinforcement learning.

Chapter 2 starts with problem formulation, followed by the step-by-step description of the algorithm. We begin with a mathematical description of the non-parametric statistical method, Mann-Whitney U test, in data association of classified objects. Then we provide details about how SLAM helps with this process of data association. Later, an unsupervised clustering method is demonstrated to localize the unknown number of objects in the map. Finally, experimental analysis is performed on public and our own dataset to show the efficiency and accuracy of our proposed method.

Chapter 3 starts with the problem formulation of loop closure. We describe the reinforcement learning framework and the policy optimization techniques used in the training of loop closure. We introduce our simulated grid environment and how rewards were provided to the learning module to perform loop closure. We provide details about the training procedure using the simulated environment and discuss the problems in training the on the

environment and provide solutions. Finally, we show experimental results with reward curve and accuracy of the trained model on several simulated environment.

1.5 Acknowledgments

This research has been funded by the Advanced Driver Assistance System (ADAS) group at Texas Instruments (TI) in Dallas, TX. We would also like to thank Viraj Mavani for his support and guidance in capturing the UTD dataset and helping generate the results for the project. Chapter 2 in part is a reprint of material published in:

- © 2018 IEEE. Reprinted, with permission, from A. Iqbal, N. R. Gans, Localization of Classified Objects in SLAM using Nonparametric Statistics and Clustering. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2018.

CHAPTER 2

LOCALIZATION OF CLASSIFIED OBJECT IN SLAM

2.1 Introduction

This chapter introduces the process of data association and localization of classified objects during the SLAM problem. Object detection and classification have advanced rapidly in recent years but are only applied to a single image. Data association establishes connection between two images of object classification. In our approach, we assume no prior knowledge or model about the objects in the map; therefore, we use a nonparametric statistical method to solve this data association problem. Afterwards, we merge the associated objects in SLAM and demonstrate a procedure to localize these classified objects in the SLAM established map.

We start with the formulation for the problem of data association and object localization. Both problems are an ongoing research problem for semantic SLAM and HD mapping. We describe a short summary of deep learning for object detection and explain why data association is required for object localization in SLAM. We provide a description of the elements of SLAM used in the process of localization and show how it helps both association and localization. The localization involves retrieving the unknown number of objects and each objects' corresponding location. We used an unsupervised clustering method to provide us the solution to this problem. Even though data association and clustering are independent processes, we introduce an intermittent process to enable the data association and clustering processes to interact with each other and act accordingly to benefit the SLAM process. This intermittent process can help run the method in real time. Finally, We demonstrate the results of our method on public datasets and our own dataset. We will also describe how this process can also be incorporated in our next work.

2.2 Notation

We first establish some notation used in the subsequent developments. Our proposed method is dependent on RGB-D data, which can be captured using a camera and IR sensor or a stereo camera. The following list is a summary of the notations used in this chapter:

$\mathbb{N}$	the set of natural numbers
$i \in \mathbb{N}^{n \times m \times 3}$	3-channel RGB color image
$d \in \mathbb{N}^{n \times m}$	a single channel depth image
C	the Cartesian reference frame of camera
W	the world/inertial reference frame
$P \in \mathbb{R}^3$	a 3D point co-ordinate
$P^W \in \mathbb{R}^3$	the coordinates of a 3D point in W
$P^C \in \mathbb{R}^3$	the coordinates of a 3D point in C
$t \in \mathbb{N}$	when used as a superscript, indicates the discrete time iteration that an image, estimate, measurement, etc. was made.
$R^t \in SO(3)$	rotation at the current time t in the corresponding frame
$\tau^t \in \mathbb{R}^3$	translation at the current time t in the corresponding frame
k, $l \in \mathbb{N}$	when used as a subscript, indicates a specific subset from the corresponding space
obj^t	a set of measured depth values of classified objects at time t, $obj^t = \{d_1^t, d_2^t, ..., d_k^t\}$
$\lvert \cdot \rvert$	designates cardinality of a set
$F(x)$	denote the cumulative distribution of data set x
$F(x_1) \sim F(x_2)$	indicates that two distributions are similar in the sense that data sets x_1 and x_2 come from the same original distribution.

Some examples of these notations in use are d_k^t and i_k^t, which are the sets of depth values and color values for the subset (image region) k at time t. Also, for each discrete pixel location $x \in \mathbb{R}^2$ in the image, the corresponding color value is $i(x) \in \mathbb{R}^3$ and the corresponding depth value is $d(x) \in \mathbb{R}$.

2.3 Problem Formulation

The classical localization and mapping model processes data from a sensor while moving through an unknown environment. It interprets the environment as a collection of landmarks or features. At any time instance, the landmark position and a sequence of poses of the sensor is estimated using the current and previous sensor measurement of the environment. Since our goal is to localize classified objects in the map of the environment, there are two underlying problems that need to be solved:

1. Data Association of classified objects in consecutive observations

2. Estimation of the location of the classified objects in the map

We assume that a SLAM process can estimate, at time instance t, the pose of the camera in world frame W and a map M^t, and an object detection module will provide a probability of the existence of the object in an image, i^t.

The first problem is to associate a classified object from different observations of the environment. Objects in an environment can be classified using a object detection module. This process is described in Sec. 2.4.1. Consider two sets of depth values corresponding to classified objects from time t_1 and t_2, obj^{t_1} and obj^{t_2}, as described in Sec. 2.2,

$$obj^{t_1} = \left\{d_1^{t_1}, d_2^{t_1}, ..., d_m^{t_1}\right\}$$

$$obj^{t_2} = \left\{d_1^{t_2}, d_2^{t_2}, ..., d_n^{t_2}\right\}$$

where, m and n are the number of classified objects from observations at t_1 and t_2 and $0 < t_1 < t_2 \leq t$.

If a classified object $d_1^{t_1}$ from t_1 is the same object in t_2 as $d_2^{t_2}$, then we need to associate them in these two observations. We define an association matrix $A^{t_1,t_2} \in \mathbb{R}^{m \times n}$,

$$A^{t_1,t_2} = \begin{bmatrix} p(F(d_1^{t_1}) \sim F(d_1^{t_2})) & p(F(d_1^{t_1}) \sim F(d_2^{t_2})) & \dots & p(F(d_1^{t_1}) \sim F(d_n^{t_2})) \\ \vdots & \vdots & \ddots & \vdots \\ p(F(d_m^{t_1}) \sim F(d_1^{t_2})) & p(F(d_m^{t_1}) \sim F(d_2^{t_2})) & \dots & p(F(d_m^{t_1}) \sim F(d_n^{t_2})) \end{bmatrix} \quad (2.1)$$

where element (i, j) of the matrix A^{t_1,t_2} is the probability that the i^{th} object in obj^{t_1} and j^{th} in obj^{t_2} are same. The following condition needs to be satisfied for a successful association of the i^{th} classified object in obj^{t_1} with j^{th} classified object in obj^{t_2},

$$A^{t_1,t_2}(i, j) \geq T; \quad 0 \leq j < n \tag{2.2}$$

where, T is the probability threshold for successful association. The elements of $A^{t_1,t_2}(i, j)$ satisfying (2.2) can be considered for association.

The second problem is to localize the associated objects in the map. At time t, given the a history of camera trajectory, c^{W^t}, association of classified objects in the environment, A^{t_1,t_2} and map m^t, a probable location of classified objects in the map needs to be established. This extends from our previous definition of observation model (1.3). In this dissertation, we model the location of each classified object in the map as a normal distribution

$$p(z^t|x^t, m) \sim \mathcal{N}(\mu^t_{1:k}, \sigma^t_{1:k}) \tag{2.3}$$

where, $\mu^t_{1:k}$ and $\sigma^t_{1:k}$ are the mean and variance of the location of the object at time, t in the map, m. Figure 2.1 shows the layered outline of our proposed method in this dissertation.

2.4 Algorithm Description

This section describes the algorithm step-by-step, starting with the object detection and classification, then data association of the classified objects using depth data. Next, we provide details about how SLAM is incorporated in the process to help both association and localization procedure. Also an intermittent process is introduced help run both association and clustering in parallel. A flowchart of the the algorithm is shown in Figure 2.1. The figure outlines the full process from sensor data to the final map with object information.

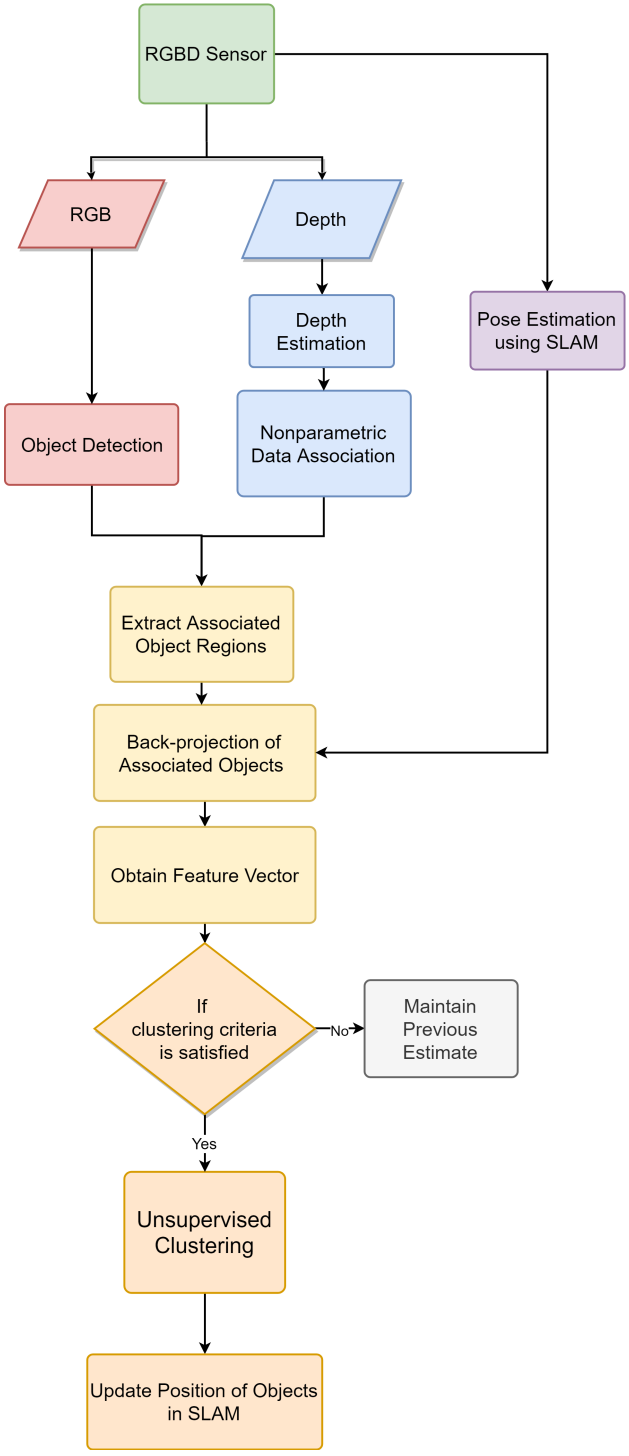


Figure 2.1: A flowchart of our proposed method for data association and localization of classified object in VSLAM.

2.4.1 Deep Learning-Based Object Detection

Object detection is the problem of recognizing that an object appears in an image and determining its location in the image, generally using a trained model. The advent of deep convolutional neural networks (e.g. [96, 33]) has led to major advancements in this field. Modern approaches can train classifiers to recognize thousands of categories of objects with an accuracy rate higher than 90%. The learning capability of CNNs is due to the multiple feature extraction stages using back propagation that can learn rich features from data. The availability of vast amount of image data and advancement in hardware capability have helped deep learning reach its current level of success. Before CNNs, image classifiers used handcrafted feature descriptors such as SIFT, SURF, Haar [99], etc. These methods did not achieve a high accuracy rate and the extracted features were limited to certain classes [72]. CNNs introduced a different architecture that follows a hierarchical design similar to how a human brain functions [77]. It has varied architectural models but generally can be described as a multiple stacked layers of convolutional filtering and pooling or sub-sampling layers, followed one or more fully connected neural network layers. The modeling of a CNN is very complex and requires a very time-intensive period to train, which can span weeks and even months. The first breakthrough results using CNNs was published by Krizhevsky et al. using Alexnet in the ImageNet LSVRC-2010 contest [48]. Subsequent years have seen modification of the CNN architecture such as VGG [87], Resnet [38], Inception [93], etc. Figure 2.2 shows results from a CNN detector on the our own datasets and other public datasets.

Our algorithm employs a detection model to retrieve the location and class of a recognized object. It is intentionally flexible in regards to the choice of object detector, as detection and classification is an active research field with frequent updates and improvements. In our experiments, we used MobileNets [43] and Faster-RCNN [78], but any detector can be used as long as they provide a class and region for the detected objects. Our models are trained

Figure 2.2: This figure shows object detection results using a CNN with region proposal network (R-CNN) on the datasets we are testing our algorithm. Classified objects are indicated by a bounding box, which features a classification label and confidence score.

on Microsoft COCO dataset [55], and our training sets contain no images of objects in our test experiment. Our model extracts the regions of detected objects in a RGB image frame, i, in the form of a bounding box (a mask could also work). Let $B_k^t \subset \mathbb{R}^2$ denote the region of an image corresponding to object k at time t, and let d_k^t denote the set of depths for all points in B_k^t. The set d_k^t is utilized in the next steps for data association, depth estimation, clustering and localization of objects in the map.

2.4.2 Non-parametric Data Association (NPDA)

Although object detection has become dramatically more accurate in recent years, association of detected objects over consecutive frames is still a challenging problem. Often, there

Figure 2.3: This figure shows detection results from two frames of a scene in TUM dataset. There are 2 chairs and 3 bottles in both image. A chair and bottle in left image does not get detected in right image. Also the bounding box region for the objects changes from frame to frame. So only relying on the object detection we cannot perform data association.

are multiple occurrences of an identical class of object detected in sequential images, and these occurrences need to be distinguished from one another from frame to frame. Additionally, objects may be occluded over multiple frames, and the object detection module can fail to detect an object or misclassify an object over a multiple frames. A data association process helps find the interconnection between these occurrences of object.

To solve this problem, we use the the Mann-Whitney statistic [58], a non-parametric inferential statistical method for data association between two observations. The Mann-Whitney test is used in statistics to determine the likelihood that two sets of samples were selected from populations having the same distribution. We use it to determine if the set of point depths for the classified objects in different frames are adequately similar that they likely correspond to the same 3D object.

The Mann-Whitney test involves calculation of a statistic, U. To calculate U to solve the object association problem, two sets of depth values are taken from the set of observations at times t and $t-1$, obj^t and obj^{t-1}. Let these sets of depth values be d_k^t and d_l^{t-1}, where k and l indicate two objects detected at time t and $t-1$, with regions B_k^t and B_l^{t-1}. Then the

U statistic can be specified as

$$U = \left| \left\{ d_k^t(x), d_l^{t-1}(y) \right\} \right|, \qquad \text{s.t. } d_k^t(x) < d_l^{t-1}(y) \tag{2.4}$$

$$\text{and } x \in B_k^t,\ y \in B_l^{t-1} \tag{2.5}$$

The cumulative distribution functions of the two sets of depth values are $F(d_k^t)$ and $F(d_l^{t-1})$. We do not reject the null hypothesis that d_k^t and d_l^{t-1} originated from the same distribution if

$$F(d_k^t) = F(d_l^{t-1} - \Delta). \tag{2.6}$$

Here, Δ is the nonparametric confidence interval, which reflects the expected difference between d_k^t and d_l^{t-1} due to motion and noise. Let V define an ordered set of all pairwise differences between d_k^t and d_l^{t-1},

$$V = \left\{ d_k^t(x) - d_l^{t-1}(y) \right\} \quad \forall x \in B_k^t \text{ and } \forall y \in B_l^{t-1}. \tag{2.7}$$

If $pwd(q)$ is the q^{th} smallest pairwise difference of the set V, then the inequality,

$$pwd(q_a) < \Delta \leq pwd(q_b); \quad q_a < q_b \tag{2.8}$$

will hold if and only if there are at minimum q_a and no more than q_b elements of the set V that satisfy,

$$d_k^t(x) - d_l^{t-1}(y) < \Delta. \tag{2.9}$$

In other words, if a sufficient number of elements in V have differences less than our confidence interval, then the origin of d_k^t and $(d_l^{t-1} - \Delta)$ are likely to have been from the same distribution. So, the probability p for the inequalities of (2.8) and (2.9) can be estimated from the U statistics [39]. For example, for a 90% confidence interval we have

$$p\left(pwd(q_a) < \Delta \leq pwd(q_b)\right) = p\left(q_a \leq U < q_b\right) = 0.90. \tag{2.10}$$

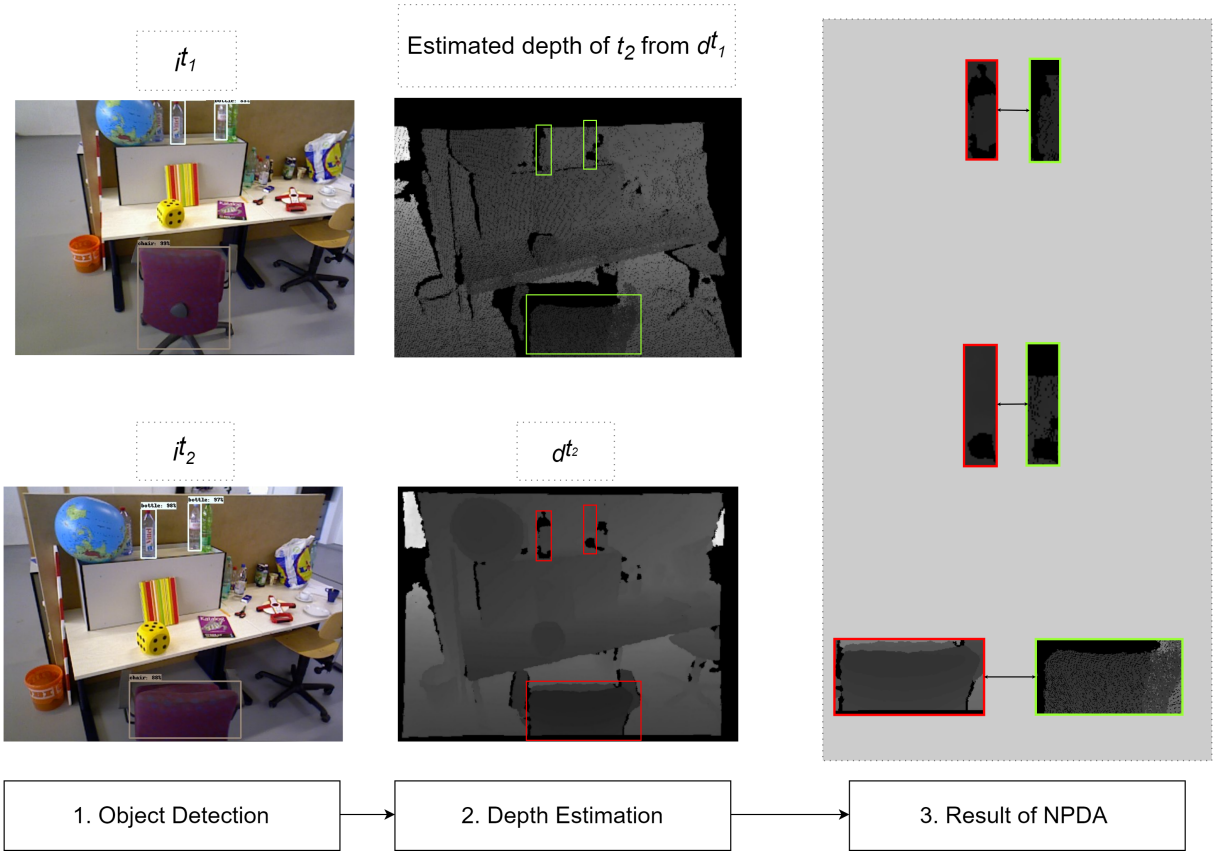


Figure 2.4: This figure shows the object detection, depth estimation and data association process. Objects are detected in i^{t_1} and i^{t_2}, and the estimated depth $d^{t_1 \to t_2}$ is calculated and to determine the likelihood that detected objects are associated.

Non-parametric statistical techniques applied on large sample-size data can be approximated using normal theory based approximation. Since sets of depth values have a large sample size, the distribution of U can be estimated using normal approximations. Moreover, the depth channels of two consecutive frames are expected to contain similar values for the same object. These similar values in $\left\{d_k^t\left(x\right), d_l^{t-1}\left(y\right)\right\}$ in such cases are called *ties*. If we have g number of tie groups and t_z denotes the number of observations in z^{th} tie group, then using normal approximation for U,

$$\begin{aligned} E(U) &= \frac{mn}{2}, \\ Var(U) &= \frac{mn(m+n+1)}{12}(1-TC). \end{aligned} \tag{2.11}$$

where $m = |d_k^t|$, $n = \left|d_l^{t-1}\right|$

and tie correction,

$$TC = \frac{\sum_{z=1}^{g}(t_z^3 - t_z)}{(m+n)((m+n)^2 - 1)}. \tag{2.12}$$

Now, the values of q_a and q_b are calculated to use in (2.10) through normal approximations. Using (2.11) and 90% confidence interval, we can approximately estimate q_a and q_b,

$$\begin{aligned} q_a &\approx E(U) - 1.645\sqrt{var(U)} \\ q_b &\approx 1 + E(U) + 1.645\sqrt{var(U)}. \end{aligned} \tag{2.13}$$

Using the estimated values of (2.13), if (2.10) is satisfied, then we cannot reject the null hypothesis. So, in retrospect d_k^t and d_l^{t-1} likely correspond to the same object when there are at least q_a and at most q_b elements that satisfy $d_k^t(x) < d_l^{t-1}(y) + \Delta$. Also, it is easy to see in (2.13) that the existence of ties will increase q_a and decrease q_b, thus giving U a lower margin to satisfy in (2.10). In general, the motion between two frames will be small if they are from consecutive frames, such that we expect ties in d_k^t and d_l^{t-1}. Therefore, no ties is also an indication of an different object. Although these are intuitive speculations, we do not derive data association based on these assumptions.

The confidence interval is the threshold for accepting the association between two sets of depth values. Since the sets of depth values used for association are discrete, the distribution of U will also be discrete. We will not be able to find q_a and q_b to precisely satisfy the confidence interval. Therefore, we vary our confidence interval from 80% to 99% for any association. In cases where multiple intervals satisfy (2.10), we use the larger confidence interval for association. This establishes the probability that the elements in (2.1) are equal to (2.10) corresponding to the specific confidence interval. Figure 2.4 shows two images i^{t_1} and i^{t_2}, which shows detected object with regions B_k^t and B_l^{t-1}, the corresponding estimated depth $d^{t_1 \to t_2}$ and d^{t_2}. An object in d^{t_2} (highlighted with a red rectangle) is checked against all the objects in $d^{t_1 \to t_2}$ (highlighted with a green rectangle) in the NPDA process to find

Algorithm 1 Nonparametric Data Association Process

1: Obtain image i^t, i^{t-1} and depth d^t, d^{t-1},
2: Detect objects using an object detector in i^t and i^{t-1}, extract region proposals for the detected objects, B^t and B^{t-1} and obtain the number of objects detected in i^{t-1}, m
3: For a object k in d^t obtained using B^t,
4: **for** $l = 0$ to m in and B^t, **do**
5: Calculate U statistic using (2.4)
6: Find the ties in $\left\{d_k^t(x), d_l^{t-1}(y)\right\}$ using (2.12)
7: Calculate $E(U)$ and $Var(U)$ using (2.11)
8: **for** confidence interval, $\Delta = 80\%$ to 99%, **do**
9: Find values of q_a and q_b for corresponding Δ,
10: **if** (2.10) is satisfied, **then**
11: then for association matrix (2.1), $p(F(d_k^t) \sim F(d_l^{t-1})) = \Delta$
12: **else**
13: break
14: **end if**
15: **end for**
16: The associated object for d_k^t is the object number, $\underset{l=0,1,\ldots,m}{argmax}\ p(F(d_k^t) \sim F(d_l^{t-1}))$ in d_l^{t-1}.
17: **end for**

the correct association. The result is of the process is also shown in Figure 2.4 in the third column. The process to perform NPDA on depth samples is expressed in further detail in Algorithm 1.

2.4.3 Depth Map Estimation for NPDA

Generally, NPDA can be applied without any modification to the depth images when there is minimal motion. But in cases when there is large motion we use a depth estimation process

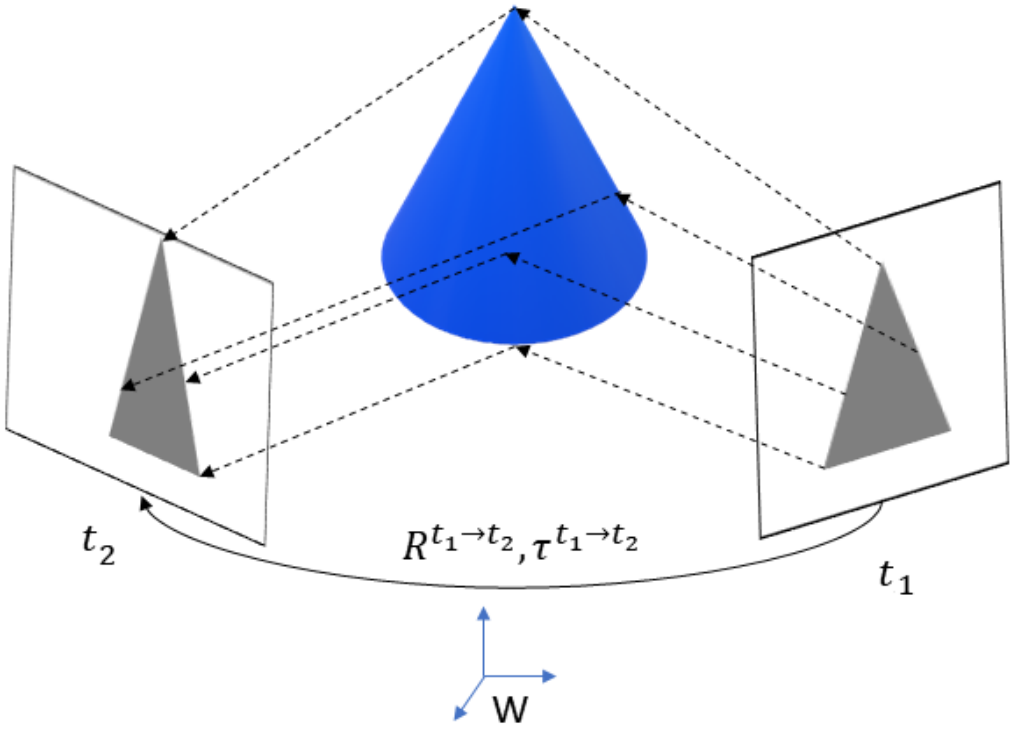


Figure 2.5: This figure shows depth estimation process for NPDA. It illustrates the depth t^1 is used to estimate depth at t_2

based on the previous observation to produce a depth image for association. This process uses the camera extrinsic and intrinsic information to generate depth from previous observation. This helps reduce the difference in depth for the two observations used for association. If we can estimate the depth at t_2 from the observation at t_1, then we can perform association in the case that motion was large between these two time instances. Let the observed motion between t_1 and t_2 be represented as rotation $R^{t_1 \to t_2}$ and translation $\tau^{t_1 \to t_2}$ retrieved from a SLAM pose estimation step. First, we back-project the depth d^{t_1} to $3D$ to obtain $P^{C^{t_1}} \subset \mathbb{R}^{3 \times N}$. It is then converted to homogeneous co-ordinate as $P^{C^{t_1}} \subset \mathbb{R}^{4 \times N}$. Next we apply the transformation obtained from SLAM pose estimation between t_1 and t_2 to obtain $P^{C^{t_1 \to t_2}} \subset \mathbb{R}^{4 \times N}$,

$$P^{C^{t_1 \to t_2}} = H^{t_1 \to t_2} \; P^{C^{t_1}}$$

where,

$$H^{t_1 \to t_2} = \begin{bmatrix} R^{t_1 \to t_2} & \tau^{t_1 \to t_2} \\ 0 & 1 \end{bmatrix}.$$

Then, $P^{C^{t_1 \to t_2}}$ is projected to the image plane at the camera position t_1 on the image plane, I^{t_1}. This projection, $proj^{I^{t_1}}_{P^{C^{t_1 \to t_2}}}$, provides the $3D$ to $2D$ correspondence between the points in W and image plane I^{t_1}. The z element of $P^{C^{t_1 \to t_2}}$ represents the predicted depth value for the corresponding I^{t_1}. This way, we retrieve the depth, $d^{t_1 \to t_2}$, for all the image points on I^{t_1}.

We perform NPDA on $d^{t_1 \to t_2}$ and d^{t_2}, as there will be less motion difference between them. Figure 2.5 shows a illustrative diagram how depth from time t_1 is utilized to estimate depth at t_2 of an object. Later, we will demonstrate in experiments how much motion can be observed in between two observations for NPDA using depth estimation. Figure 2.4 shows the estimated depth, $d^{t_1 \to t_2}$, which is estimated from d^{t_1} using the motion in between t_1 and t_2. Figure 2.6 shows the results of the depth estimation process with different sets of depth images. Generally, when the motion in between two key frames is large (more than a preset threshold), a depth estimation helps the NPDA to perform association.

2.4.4 Object Back-Projection using SLAM Pose

We have gathered information about objects in the environment and its association in previous observations. Now we are focused to infuse this information with SLAM and find the actual location of these objects in the SLAM established map. *Every SLAM process provides a map and a location of the sensor in the map.* Modern VSLAM procedures consist of two components, front-end and back-end. The front-end operates on the the raw sensor data for model estimation which may provide feature detection, association and sensor odometry. The back-end executes on the inference of the data from front-end and provide feedback for verification and correction. This primarily generates map of the global environment and

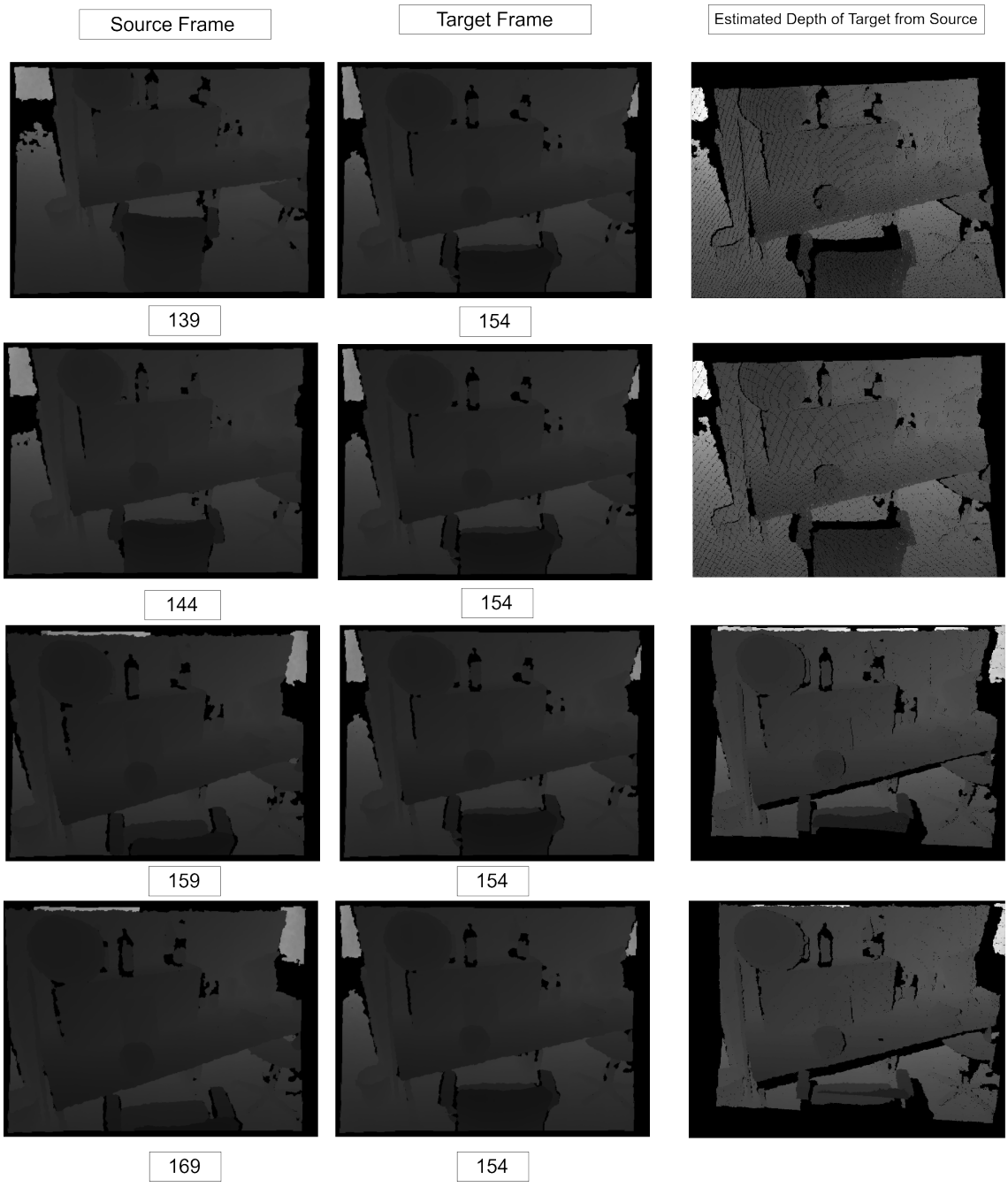


Figure 2.6: Depth estimation results on TUM dataset: The left and middle column shows the source and target image. The right column shows the estimated depth of the target image using only the source image and estimated pose from SLAM.

Figure 2.7: This figure shows SIFT matching performed in C with images from UTD Dataset.

perform any correction from loop closure detection. The front-end is generally works on camera co-ordinate frame and back-end works on the back-end. The interplay between these two produces the map and trajectory. There are many implementation of the VSLAM process. We use a feature-based localization and mapping process to extract pose and establish mapping of the environment.

In our implementation, the front-end consists of pose estimation module based on SIFT [57] feature extraction and matching process. Other implementations can use other existing feature point detectors and descriptors such as SURF, ORB, etc. The extracted feature in both frames are paired using Brute-Force matching, then classified and the associated objects in NPDA, using d^t and d^{t-1}, are back-projected in C to acquire $P_k^{C^t}$ and $P_k^{C^{t-1}}$. Next, an iterative closest point (ICP) algorithm is used to estimate the pose between t and $t-1$. The points are tracked over time to estimate motion in world co-ordinate frame W. ICP is often used for $3D$ reconstruction, robot localization, etc.

In the back-end, first we transform the current feature points $P_k^{C^t}$ to the last camera position in W to estimate $P_k^{W^t}$. Then, the tracked pool of features from $P_k^{W^t}$ are associated with the features in $P_k^{W^{t-n}}$ to estimate camera motion in W. Any unmatched feature is associated through a nearest neighbor search, only if its within a certain radius. The motion estimation

is performed using an ICP variant based on Levenberg-Marquardt (LM) optimization [41]. It is a point to point correspondence error minimization procedure. Let $P_m^{W^t}$ be the set of points in the maintained map that are candidate matches for $P_k^{W^t}$ at time t. Let $a \in P_k^{W^t}$ and $b \in P_m^{W^t}$ be a matched set of points. Then the minimization in ICP to find the camera rotation $\mathbf{R} \in SO(3)$ and translation $\tau \in \mathbb{R}^3$ composed as rigid body motion $T = \{R, \tau\}$, can be stated as,

$$\tau^t, \mathbf{R}^t = \arg\min_{\tau, \mathbf{R}} \sum_{a,b} w_k \|\mathbf{R}a + \tau - b\|^2 \tag{2.14}$$

here w_k is the weights for (a, b) pair to provide significance during LM optimization.

A classified object k in i^t is back-projected to W. This requires knowledge of its associated depth values d_k^t, the camera pose at t and the camera intrinsic parameters. If the object k has had a successful association using the NPDA process from $t - n$ to t, then all the back-projected points from $P_k^{W^{t-n}}$ to $P_k^{W^t}$ are integrated into a joint distribution to represent a single feature vector. The feature vector f for an object is expressed as,

$$f = \{X, Y, Z, S\}$$

where X: $(\mu(x_k), \sigma^2(x_k))$, Y: $(\mu(y_k), \sigma^2(y_k))$, Z: $(\mu(z_k), \sigma^2(z_k))$. The $\mu(x_k), \mu(y_k), \mu(z_k)$ are the mean and $\sigma^2(x_k), \sigma^2(y_k), \sigma^2(z_k)$ are the variance in (x, y, z) axes accordingly. S is the class of the object. The feature vector, f represents a classified object for the time duration $(t-n) \rightarrow t$. It is later used in clustering process (Sec 2.4.5) to calculate a distance metric. Figure 2.4 shows the associated objects from NPDA back-projected in W.

2.4.5 Density based Clustering of Objects

The NPDA process associates objects from the current frame to the next captured frame. However, it will fail to associate objects with obstruction and cannot distinguish if a previously seen object is observed again after a period of not being observed. Due to this, we

cannot determine the number of objects in the map solely from the NPDA process. We address this problem using a clustering process to determine the number and location of these associated objects in the map.

Clustering is the process of grouping physical or abstract objects into classes of similar objects. A cluster is a group of data which are similar, and dissimilar to data in other clusters. There are different clustering methods depending on various factors such as data space, number of clusters, hard or soft grouping, etc. Each of the existing method has its own strengths and weakness. Most of the existing methods are based on connectivity, density, subspace, centroid and hierarchy. The prominent clustering algorithm in practice are K-means, Mean Shift, Hierarchical Clustering, Density-Based Spatial Clustering, Expectation Maximization Clustering, etc [7]. K-means clustering is the most popular clustering which finds a fixed number of cluster through alternating between assignment and update. Although it is very effective it requires fixed number of cluster to perform analysis which in our case we do not have. Hard clustering method provides a specific cluster for each object in the data whereas soft clustering provide a probability of each object belonging to existing clusters. Density based clustering searches for higher density areas in the data compared to the rest of the data. Distribution based clustering is based on the concept that a cluster is generated from same distribution if there is several distinguishing distribution in the data. We opted to use a density based clustering because in data space the classified objects are concentrated in their respective locations.

Our process requires a clustering algorithm to estimate the unknown number and locations of the objects from the data of the map and NPDA. Based on these requirements, we selected an unsupervised clustering algorithm known as hierarchical density-based spatial clustering of applications with noise (HDBSCAN) [60, 61]. HDBSCAN is an extension to DBSCAN [29] to make it into a hierarchical clustering algorithm. It uses unsupervised learning to find clusters and is shown to be more robust to parameter selection and has been

shown to outperform exiting density based methods. It also requires a fairly low computational complexity compared to other existing methods [102]. It first transforms the data according to the density,then build a minimum spanning tree and construct a cluster hierarchy of connected components. Then based on minimum number of cluster size the cluster hierarchy is condensed and stable clusters are extracted from the condensed tree [60, 61].

The clustering process requires an estimate of density to group similar objects. This density estimate is usually calculated using distance between neighbors. There are many popular distance metric in use e.g. Euclidean, Manhattan, Pearson's correlation coefficient, etc. We use Bhattacharyya distance (BD) [8] to measure the distance between all the feature vectors. BD is selected because it accounts for the mean and variance in the distance calculation between two distribution. The distance metric D between two feature vector f_a and f_b is,

$$\begin{aligned} D(f_a, f_b) =& BD(f_a(X), f_b(X)) + BD(f_a(Y), f_b(Y)) \\ &+ BD(f_a(Z), f_b(Z)) + g(f_a(S), f_b(S)) \end{aligned}$$

where BD is given by

$$\begin{aligned} BD(f_a(X), f_b(X)) =& \frac{1}{4}\log\left(\frac{1}{4}\left(\frac{\sigma^2_{f_a(X)}}{\sigma^2_{f_b(X)}} + \frac{\sigma^2_{f_b(X)}}{\sigma^2_{f_a(X)}} + 2\right)\right) \\ &+ \frac{1}{4}\left(\frac{\left(\mu_{f_a(X)} - m\mu_{f_b(X)}\right)^2}{\sigma^2_{f_a(X)} + \sigma^2_{f_b(X)}}\right) \end{aligned}$$

and $g(f_a(S), f_b(S))$ is a penalty function for non-matching classes. For example, $g(\cdot,\cdot) : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ can be given by

$$g(x, y) = \begin{cases} 0 & \text{if } x = y \\ 100 & \text{else} \end{cases}.$$

2.4.6 Intermittent Clustering with NPDA

The clustering process generates a number of clusters corresponding to the distance metric, which shows a density of similar objects in the world coordinate frame W. However, the

distance metric calculation of the clustering process becomes computationally expensive as more feature vectors from observations are added over time. This makes it impractical to calculate the distance metric and train the cluster after every frame. For example, for every n new and already exiting m observations, the process will require $(m+n)*n$ additional operations for the new distance metric D. This extensive operation can be performed intermittently with the help of NPDA. Since the operation of NPDA only occurs on two frames for data association, it computational complexity requires less operations than clustering. Now we demonstrate how NPDA and clustering process can be used together to overcome this problem. NPDA is used to query the model and indicate when it is necessary to train cluster again.

The NPDA process provides association information between detected objects at the current and the previous time frame. As this association process continues, we do not use clustering to find any new objects. Instead we use the trained clusters at current time to see if it fits the current associated objects. If the currently associated objects fit to the existing clusters, then we assume that no new objects have been observed since the last clustering process. However, if the currently associated objects do not fit to any of the existing clusters, it indicates the existence of new objects in the environment. Then we need to calculate distance metric for the new observation and train the cluster again to locate the new objects in the map.

Let us denote, after time t, the set of detected objects as N. The set of associated objects from the NPDA process in the same duration is denoted M. Now, if the set of detected objects fitting in the current clusters is T, then the rest are the set of outliers, denoted O. So, by convention, $N = T \cup O$ and $M \subseteq N$. If the following condition is satisfied, we add new observations in the D and run the clustering again,

$$|M \cap T| < |M \cap O| \,. \tag{2.15}$$

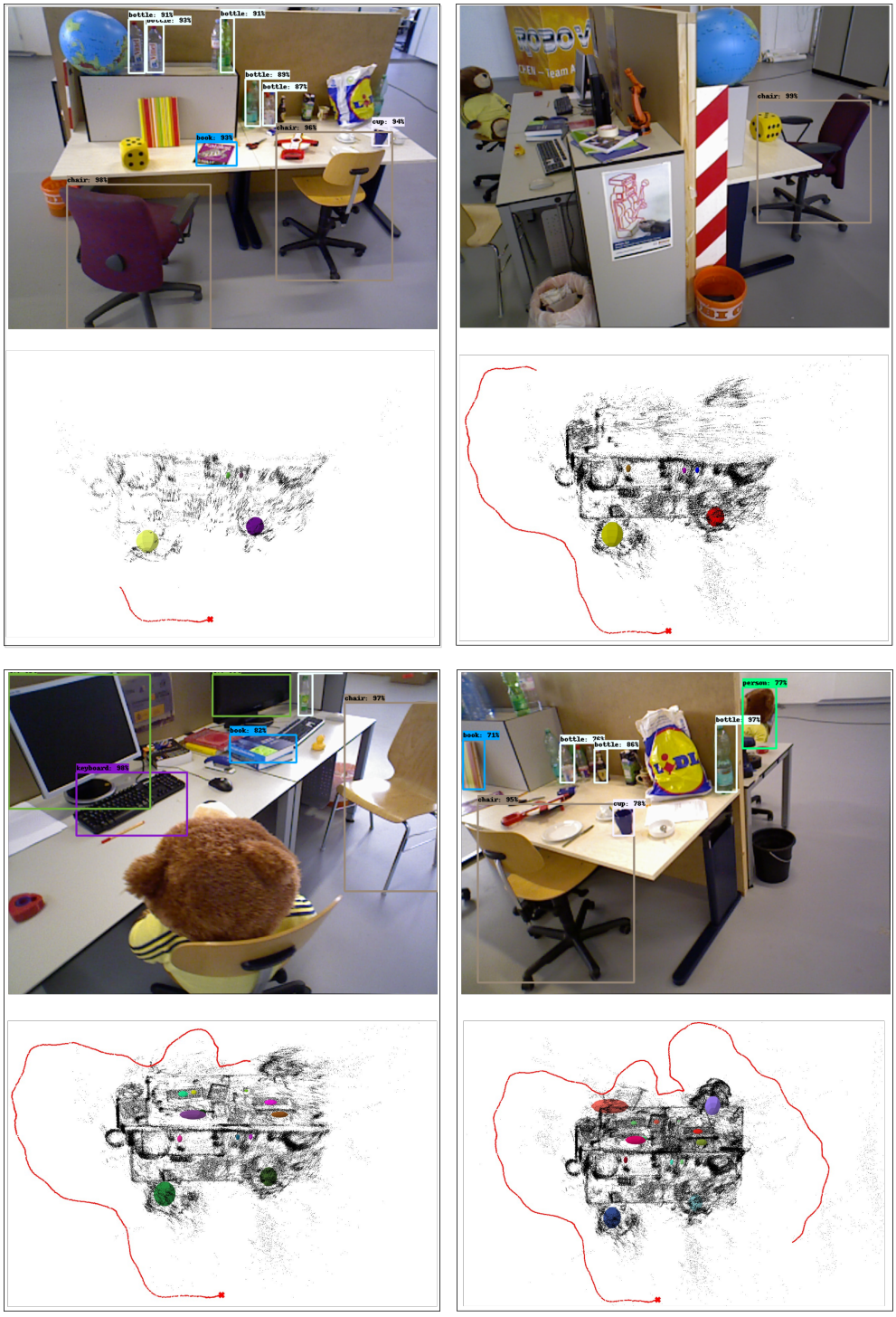


Figure 2.8: Results of the intermittent Clustering Process: The results are from a simulation on TUM. *freiburg3 long office household* dataset.

In our process, we have a threshold for the number of new objects to observe before we train the clusters. Figure 2.8 shows the intermittent clustering process in the TUM dataset. It displays four locations where clustering was performed based on (2.15) and shows the clusters as ellipsoids. Each ellipsoid constitutes a probable location and size of an object.

2.4.7 Loop Closure Correction

Loop closure is used to indicate a return to previously visited location and help correct drift in the trajectory estimation [2, 88]. In the event of a loop closure correction, the trajectory and the position of localized objects in the map needs to be updated. The clustering process is only affected by loop closure correction, as NPDA only occurs in between two observations. After a loop detection and correction from batch optimization, the poses of the agent at the previously visited places are updated, if required. The updated pose of the previous observations are used to correct the feature vectors f and run the clustering process again on this new f to find the new positions of the classified objects in the map. This way, the locations of the objects are corrected within the map along with the loop closure information.

2.5 Experimental Results

We have evaluated our algorithm on three different public datasets and our own dataset. The public datasets are from Technical University of Munich (TUM) RGB-D data set [88], Microsoft RGB-D Dataset 7-Scenes [34] and RGB-D Object Dataset [52]. Our own dataset consists of 3 routes, each through a lab environment populated by different arrangements of common objects. The object detection modules used here are the MobileNet SSD detection network and Faster-RCNN, which is trained on the Microsoft COCO dataset [55]. MobileNet is efficient for mobile and embedded vision applications, as the network itself is a light-weight deep neural network. All of our codes are implemented using Python libraries. The Tensorflow platform [1] is used for object detection.

Algorithm 2 Data Association and Localization of Classified Objects

Obtain image i^t and depth d^t,
Detect objects using an object detector in i^t, extract region proposals for the detected objects, B^t and get number of object detected, k
Perform pose estimation between i^t and i^{t-l} to obtain H^t,
if observed motion $H^{(t-l)\to t}$ is greater than threshold, **then**
    Use depth estimation to obtain $d^{(t-l)\to t}$,
end if
Perform NPDA to associate currently detected objects to the previous frame, l, to acquire the associated observation $A^{(t-l),t}$,
Back-project the currently detected and associated objects in world map again using the estimated pose in W and acquire feature vectors, f_a, $a \in \{1 \dots k\}$
if there are no cluster in map, **then**
    Find group of clusters, G_t, using the clustering process,
else
    if the feature vectors from the associated observation fits any of the current clusters, **then**
        keep the group of clusters same, $G^t = G^{(t-l)}$
    else
        if (2.15) is satisfied **then**
            Retrain the clusters
            update G^t
        end if
    end if
end if

2.5.1 Evaluation Metric

Our results were evaluated with standard performance metrics in scientific literature. We precision rate, recall rate as metric to report data association accuracy. The precision metric is defined as,

$$Precision = \frac{TP}{TP + FP}$$

where,

$$TP = \quad \text{True Positive which is correctly identified instance}$$

$$FP = \quad \text{False Positive which is incorrectly identified instance.}$$

The recall metric is defined as,

$$Precision = \frac{TP}{TP + FN}$$

where,

$$FN = \quad \text{False Negative which is incorrectly rejected instance.}$$

These metrics are used to report data association accuracy. We also use a rotation metric described in [44] to measure the difference in rotation between the two observations. This is a bi-invariant metric on $SO(3)$. Let R^{t_1} and $R^{t_2} \in SO(3)$ represent rotation observed in W for observation t_1 and t_2, then the following bi-invariant metric is used to measure the difference between them,

$$\phi(\mathbf{R^{t_1}}, \mathbf{R^{t_2}}) = \left\| log\left(\mathbf{R^{t_1}} \mathbf{R^{t_2}}^T\right) \right\|, \quad 0 \leq \phi < \pi. \tag{2.16}$$

The metric for translation, is the norm of the translation difference between the two observations.

$$\tau = \left\| \tau^1 - \tau^2 \right\|$$

We use metric ϕ and τ to report results for the NPDA with depth estimation process. This demonstrates the observed motion(translation and rotation) between two observations and their corresponding precision and recall.

Table 2.1: Precision and Recall Statistics of NPDA Process

Dataset Sequence	Precision	Recall
TUM freiburg3 teddy	95.7	75.8
TUM freiburg2 desk	92.4	68.4
TUM freiburg3 long office household	89.9	65.6
MS Scene Fire	90.5	68.6
MS Scene Office	95.2	72.3
MS Scene Pumpkin	89.3	65.4
MS Scene Chess	92.1	69.7

2.5.2 NPDA Accuracy

We report two types of results for NPDA data association accuracy. The first is the accuracy of the association process in consecutive frames. We use the precision and recall score to report the accuracy of this process. The results are from 7 data sequence from the aforementioned public data sets. NPDA was performed over 5000 frames from the public datasets. Table 2.1 shows overall precision and recall results for the corresponding data sequence. It is clear from Table 2.1 that the NPDA process shows an high precision rate. The recall rate can be higher by tuning the NPDA process, but it reduces the precision rate. We opted for a higher precision over recall rate. One of the reasons is incorrect data associations can disproportionately distort the object localization process; that is, false positives have more detrimental effect on clustering than false negatives. The clustering process involves a large number of observations where feature vectors of correct data association localize objects in the map. False positive from observations can distort this process, while false negatives do not distort the process as long as there are true positive samples.

Second, we demonstrate again the association accuracy with PR statistics for the depth estimation process. These results are for different amounts of rotation and translation between two observations. Table 2.2 summarizes the precision and recall statistics of the NPDA process with depth estimation. $\emptyset$ indicates there was no data for this combination of ϕ and

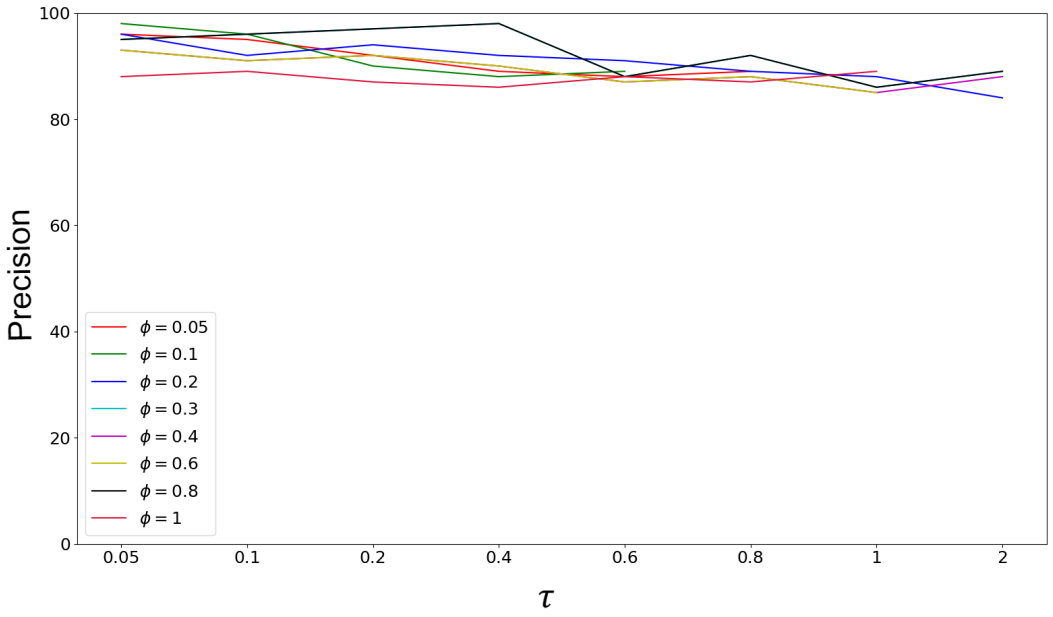
Table 2.2: Precision and Recall Statistics of NPDA with depth estimation

ϕ / $\|\tau\|$	$\phi = 0.05$	$\phi = 0.1$	$\phi = 0.2$	$\phi = 0.3$	$\phi = 0.4$	$\phi = 0.6$	$\phi = 0.8$	$\phi = 1$
$\|\tau\| = 0.05$	96\68	95\58	92\53	89\49	88\39	89\35	$\emptyset$	$\emptyset$
$\|\tau\| = 0.1$	98\65	96\60	90\52	88\41	89\36	$\emptyset$	$\emptyset$	$\emptyset$
$\|\tau\| = 0.2$	96\64	92\55	94\49	92\39	91\35	89\33	88 \29	84 \25
$\|\tau\| = 0.4$	95\65	96\58	97\53	98\45	88\36	92\28	86\29	89 \26
$\|\tau\| = 0.6$	93\58	91\45	92\43	90\37	87\31	88\26	85\21	88 \19
$\|\tau\| = 0.8$	93\54	91\40	92\36	90\33	87\28	88\22	85\20	$\emptyset$
$\|\tau\| = 1$	95\44	96\37	97\31	98\25	88\15	92\8	86\9	89 \6
$\|\tau\| = 2$	88\30	89\26	87\19	86\15	88\8	87\5	$\emptyset$	89 \6

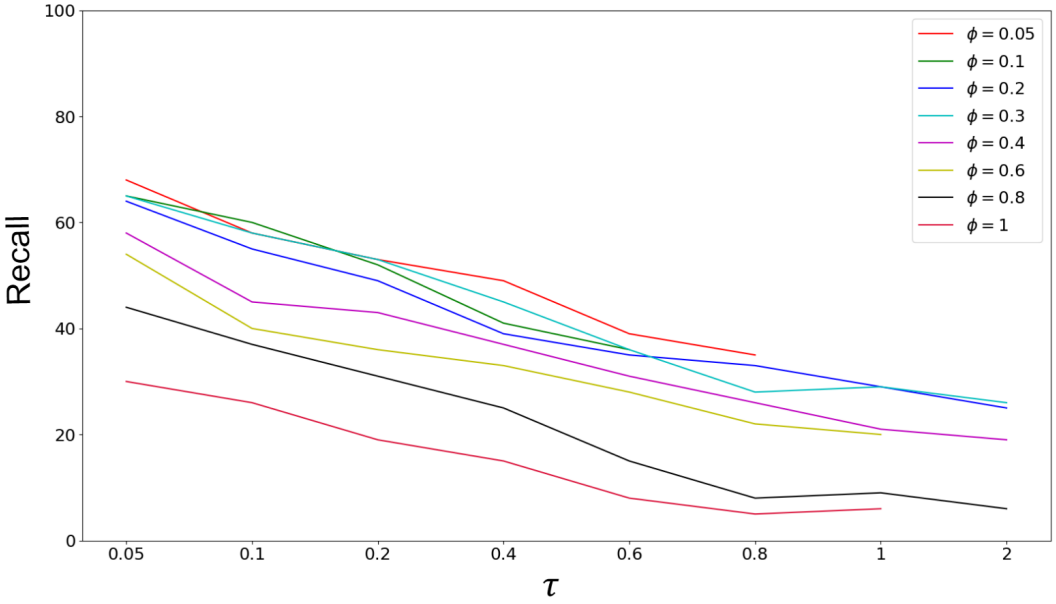
τ. Figure 2.9 also demonstrates the results of the table 2.2. It can be seen from Figure 2.9 that the precision for NPDA stays more than 80% for all cases, but the recall rate falls as more motion occurs in between observations. As stated earlier, the recall drop is due to the fact that we want to maintain a high precision for a correct data association process. Whenever there is little overlap in the camera field of view of two observations, the recall rates are low.

2.5.3 Classified Object Localization Accuracy

One challenge in evaluating our approach is that most of public datasets do not provide ground truth semantic labels of objects in the scene, nor do they provide ground truth for the location of the objects in the environment. To provide a qualitative evaluation we report two numbers, the number of unique objects detected in a dataset and the number of clusters in the map at the end of the data sequence. The number of clusters should be close to the number of detected objects to prove the effectiveness of our algorithm. However, our



(a) Precision Statistic



(b) Recall Statistic

Figure 2.9: Precision and recall statistics of depth estimation process: The results are reported with different combinations of τ and ϕ. The horizontal axis is for translation difference τ and each line is for a different rotational difference ϕ described in (2.16).

algorithm is intended to be conservative and only add clusters that are repeatedly detected, reducing false matches.

Multiple objects are generally detected in an environment during a sequence. In this proposed method, an object is associated and clustered only after it has been observed for a certain number of frames. First, we consider an object has been detected in a dataset if it is being detected by the CNN for at least 25 frames in the complete sequence. The minimum sampling size for a cluster was selected to be 50. This is intended to only add reliably detected objects to the clusters. Objects that are only briefly seen or are misclassified due to ambiguous appearance are not clustered. The numbers for classified objects and clusters in each data sequence is reported in the respected tables.

The clustering process is based on the open source HDBSCAN library [61]. The final outcome consists of a point cloud map with each object cluster represented as an ellipsoid. The center of the ellipsoid is the mean of the cluster, and the radii are the standard deviation of the cluster along the principal axes of the cluster. The final maps shown in this paper have been down-sampled using VoxelGrid filter for clarity of viewing. The red color curve indicates the trajectory of the camera or observation sensor.

TUM RGB-D dataset

We tested on four TUM sequences. The datasets are, *freiburg3 teddy*, *freiburg2 desk*, *freiburg3 long office household* and *freiburg3 long office household validation*. All of these sequences are indoor environment with walls and tables and common office objects such as book, chair, keyboard, monitor, plants, balls, etc.

We demonstrate the intermittent clustering procedure in Figure 2.8. The results are from a simulation on the TUM *freiburg3 long office household* dataset at four different times with the results of the the clustering operation, as described in Section 2.4.6. It shows camera images, and the results of the object detection are shown in bounding box regions. The

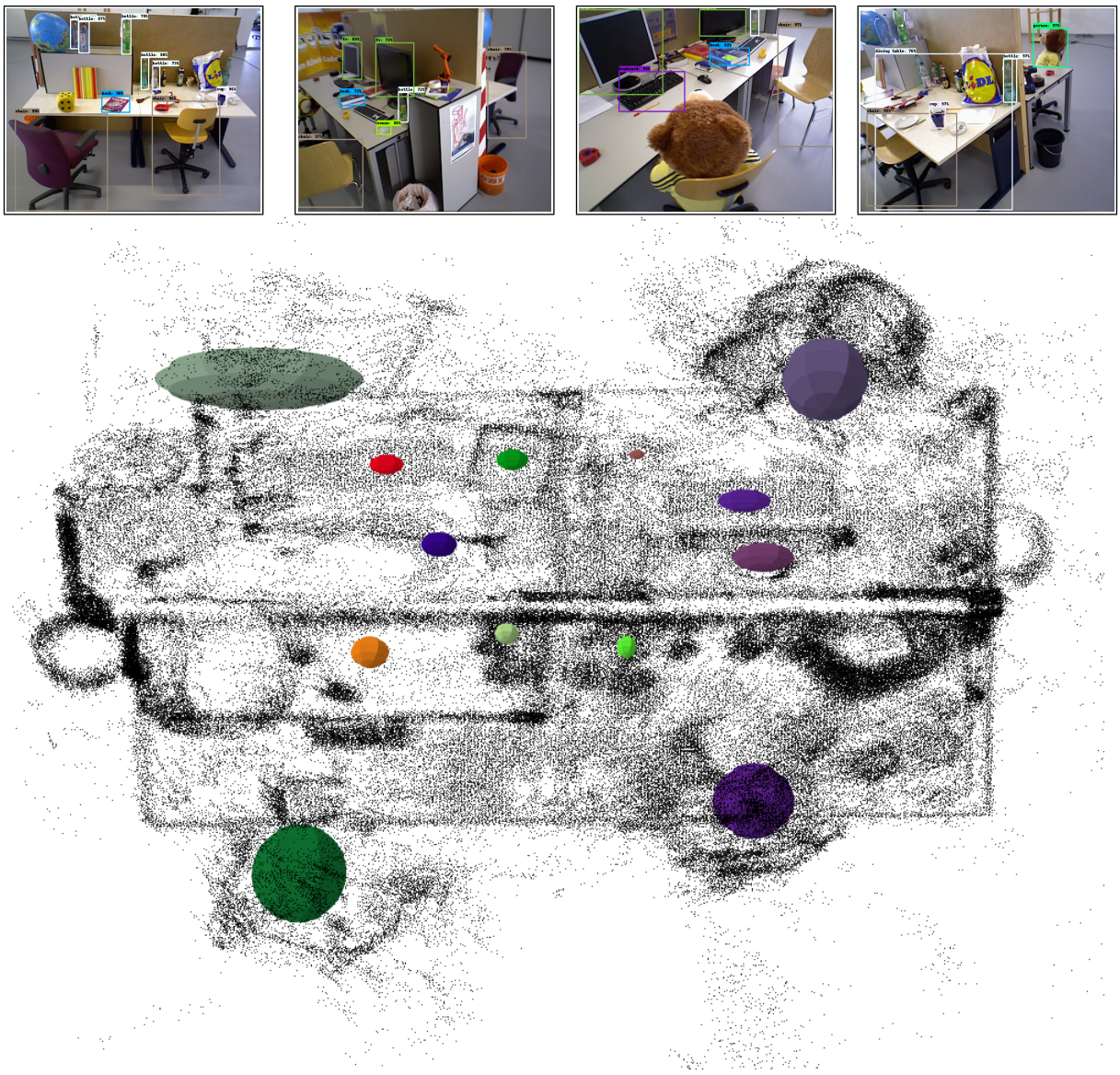


Figure 2.10: Object Localization Result on TUM Dataset, *long office household* - The ellipsoids represents the probable location and size of the objects.

bottom row shows the map for the corresponding time at the image as black points. It also shows ellipsoids representing each cluster for a classified object. Each ellipsoid constitutes a probable location and size of the object.

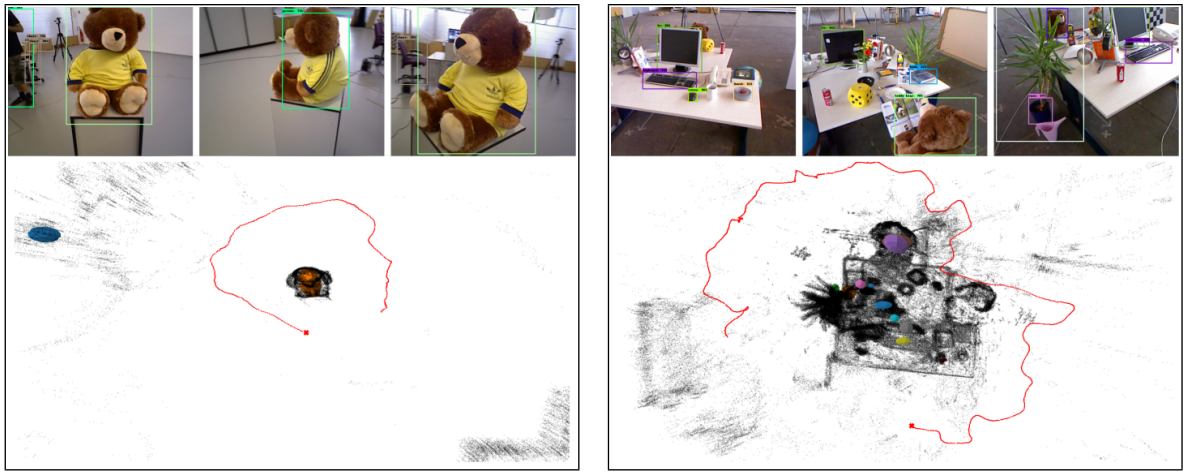
Figure 2.11a shows the results of *freiburg3 teddy*. It focuses on 1 object, a teddy bear, in the middle of a room with other objects. The classifier detected 3 objects in this sequence,

Table 2.3: Result Statistics for TUM Dataset

Dataset Sequence	Number of Classified Objects	Number of Mapped Objects
freiburg3 teddy [Figure 2.11a]	3	2
freiburg2 desk [Figure 2.11b]	15	11
freiburg3 long office household validation [Figure 2.10]	21	15
freiburg3 long office household [Figure 2.8]	21	13

and the final set of two clusters consists of a teddy and a chair. The other consistent detection was a person in the background. It was represented as a cluster, as either their motion or number of observations resulted in a distribution that was not appropriately dense to meet our criteria to constitute a cluster. *freiburg2 desk* contains several office items on a desk. The map, path, and final clusters are shown in Figure 2.11b. *freiburg3 long office household* and *freiburg3 long office household validation* have similar environments consisting of two desks divided by a wall.

The results, including final clusters for *freiburg3 long office household validation* are shown in Figure 2.10. The results for all TUM datasets are consolidated in Table 2.3. We are able to localize about 65% of the unique objects in the map. The number of localized objects depends on the number of observations that contribute to the feature vector in the clustering process. If the object does not have enough observations then it may not be recognized as a cluster. This demonstrates a strength of our approach, as mobile objects and difficult to classify objects are not added. Also the shape of the cluster ellipsoid depends on how the object was observed. In Figure 2.11a, the object was observed from all angles so it has an ellipsoid similar to the object size and location. In cases where an object is partially observed, the ellipsoid will only resemble the observed data. Overall, the localized entities shown in Figure2.11a,2.10,2.11b visually resembles the location and size of the existing objects in the map.



(a) TUM: freiburg3 teddy (b) TUM: freiburg2 desk

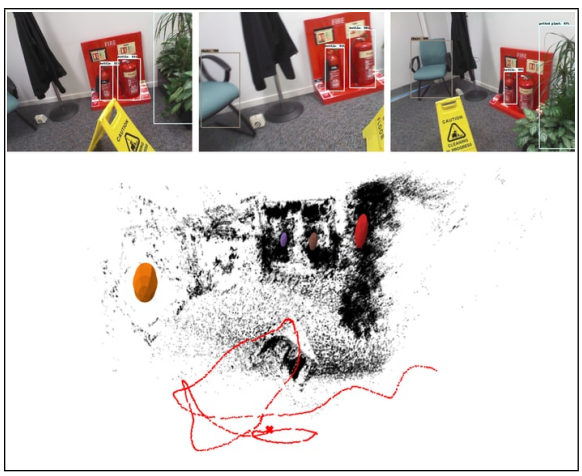
Figure 2.11: Object Localization Result on TUM Dataset

Table 2.4: Result Statistics for MS RGB-D Dataset 7 Scenes

Dataset Sequence	Number of Classified Objects	Number of Mapped Objects
Scene Fire [2.12a]	5	4
Scene Office [2.12b]	14	12
Scene Pumpkin [2.12c]	6	5
Scene Chess [2.12c]	6	5

RGB-D Dataset 7-Scenes

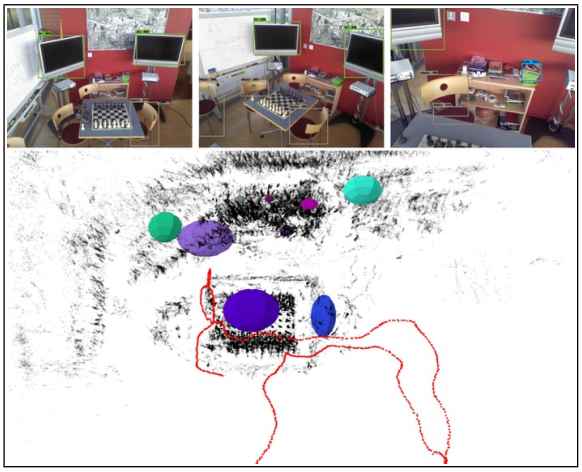
The RGB-D Dataset 7-Scenes from Microsoft consists of 7 different environment, each with its own different sequences. Figure 2.12a - 2.12d shows the results of our simulation for the 4 scenes, and outcomes are reported in Table. 2.4. Each of the sequence has a unique objects (such as chessboard, pumpkin, fire hydrants and, etc) with other commonplace object in the office environment. We were able to localize all of these unique objects in the map. Table. 2.4 shows overall we clustered about 75% of the classified objects in the environment.



(a) Scene: Fire

(b) Scene: Office

(c) Scene: Pumpkin

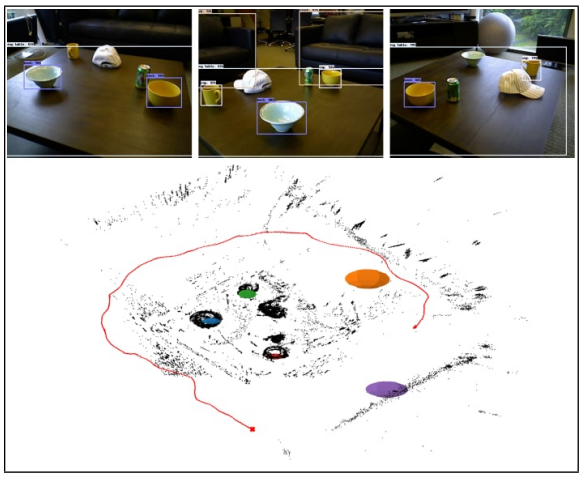


(d) Scene: Chess

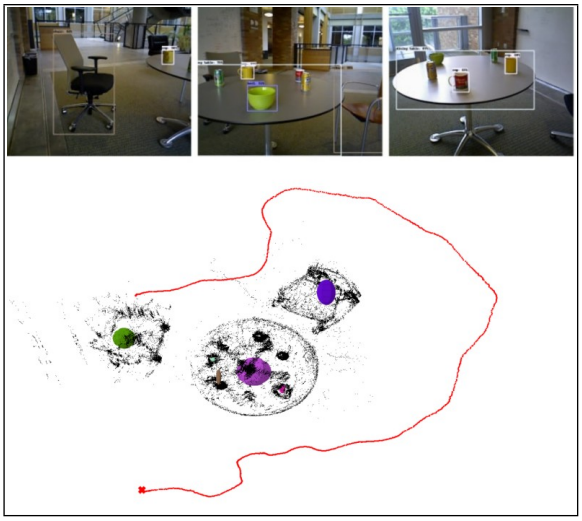
Figure 2.12: Object Localization Result on Microsoft RGB-D 7 scenes Dataset

RGB-D Object Dataset

The RGB-D Scenes Dataset v2 consists of 14 scenes that contain multiple pieces of furniture and different objects such as cereal boxes, coffee mugs, bowls, caps, and soda cans. This dataset was introduced by Kevin Lai [52, 51]. Lai demonstrated unsupervised feature learning with this dataset. On all of the sequences, small objects are placed on a flat surface, and the camera moved around the objects. Due to the object centered view and camera observations of the objects were attained from almost all angles, we chose this dataset for object

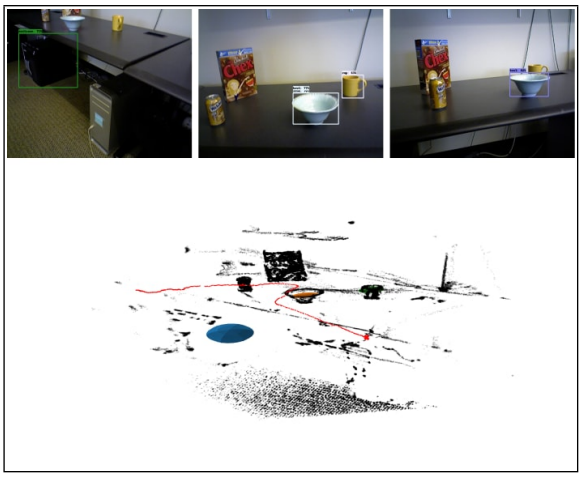


(a) RGB-D Scenes v2: scene 01



(b) RGB-D Scenes v2: scene 06

(c) RGB-D Scenes v2: scene 10



(d) RGB-D Scenes v2: scene 13

Figure 2.13: Object Localization Result on Washington RGB-D Scenes Dataset

localization. We tested on sequences 1, 6, 10, 13, 14. The results for these sequences are shown in Figure 2.13a-2.13d, and Table 2.5 shows the reported results. The results indicate that for these sequences, the majority of detected objects were localized in the map. The classified objects with observations from multiple viewpoints are localized in the map. It shows the localized objects with proportional size ellipsoid. This represents we were able to locate the objects on the correct location with its correct corresponding size.

Table 2.5: Result Statistics for RGB-D Scenes Dataset

Dataset Sequence	Number of Classified Objects	Number of Mapped Objects
Scene 1 [Figure 2.13a]	7	5
Scene 6 [Figure 2.13b]	9	7
Scene 10 [Figure 2.13c]	8	6
Scene 13 [Figure 2.13d]	4	3
Scene 14	5	4

UTD Dataset

The goal of this dissertation is to demonstrate the process of data association and localization of classified objects in SLAM. The public datasets we tested on are primarily used for SLAM and 3D scene or object reconstruction, These dataset does not include the ground truth information about the location, shape, orientation, and class of the existing objects in it. So, we introduce a dataset to compare object localization accuracy. We have captured three data sequence using a Kinect v2 RGB-D sensor moving through a lab environment with rearrange-able walls and a variety of objects. The datasets included regular everyday objects such chairs, mugs, cups, toys, bottles, etc. These objects can be easily classified using any traditional CNN-based model. In all of the sequences objects are placed at scattered locations in the environment. Figures 2.14 and 2.15 show images from the datasets along with the results. The three setup is as follows,

- Scene 1: It contains no wall and has objects scattered all around the map.
- Scene 2: It has a wall at the center and the objects are placed around this wall. The camera moves around the center wall and also creates a loop in its trajectory.
- Scene 3: It has two wall positioned at center and objects are placed similar to sequence 2. The camera movement in this sequence follows an 8 shape pattern and has two loops in the trajectory.

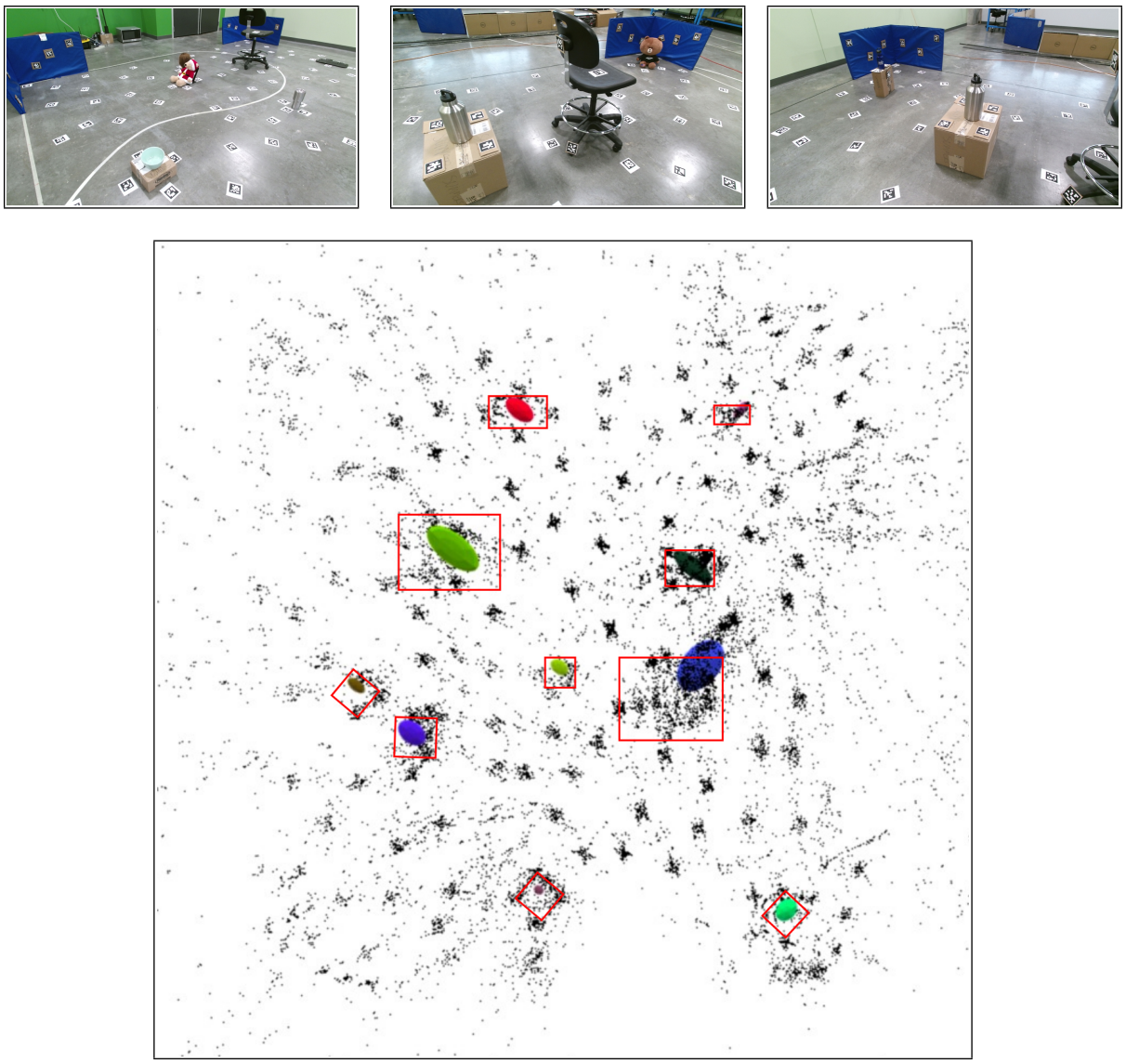


Figure 2.14: The layout of the Scene 1 of UTD Dataset and the mapping results with ellipsoids representing the objects: Object are placed at several locations with tag marks around them in the images. The tag markers can provide a region of the objects in the map similar to a bounding box in object detection.

Table 2.6: *IoU* Statistics of different object class for UTD Dataset

Sequence	Object Class									
	chair 1	chair 2	Bowl	teddy 1	teddy 2	head	keyboard	bottle 1	bottle 2	bottle blue
Scene 1	0.52	0.42	0.75	0.68	0.68	0.48	0.48	0.55	0.48	0.47
Scene 2	0.48	0.52	∅	0.53	∅	0.59	0.55	0.51	∅	0.42
Scene 3	0.51	0.56	0.6	0.47	0.55	0.55	0.45	0.44	0.44	0.53

We placed detectable planar markers in a rectangle around each object to estimate its location in the map. In Figure 2.14, it can be seen that the environment also has tag markers placed on the floor and around the objects. Each one of the marker is a square identifier that has a wide black border and a inner binary matrix. This design of the matrix represents a number that acts as an unique ID for the marker. By properly placing the markers in an environment, it can be used to estimate pose and map. We used Marker Mapper [68] and UcoSLAM [69] to obtain locations of the markers, which we compare against our results. The marker IDs around each of the object were recorded, and these markers will provide a region where an object is located in the map. This allows us to report object localization results using the Intersection over Union (IoU) metric. IoU is defined as,

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}.$$

Let A be the region on the ground plane of the rectangle defined by the markers for an object on the map, and let B be the region of a rectangle that bounds the projection of the cluster ellipse onto the ground plane.

$$IoU = \frac{A \cap B}{A \cup B}.$$

Figures 2.14 and 2.15 show the datasets with captured images, a generated map and our results with bounding box regions for the corresponding object. Table 2.7 shows the results of our proposed method on the datasets. All the datasets consisted of regular objects such

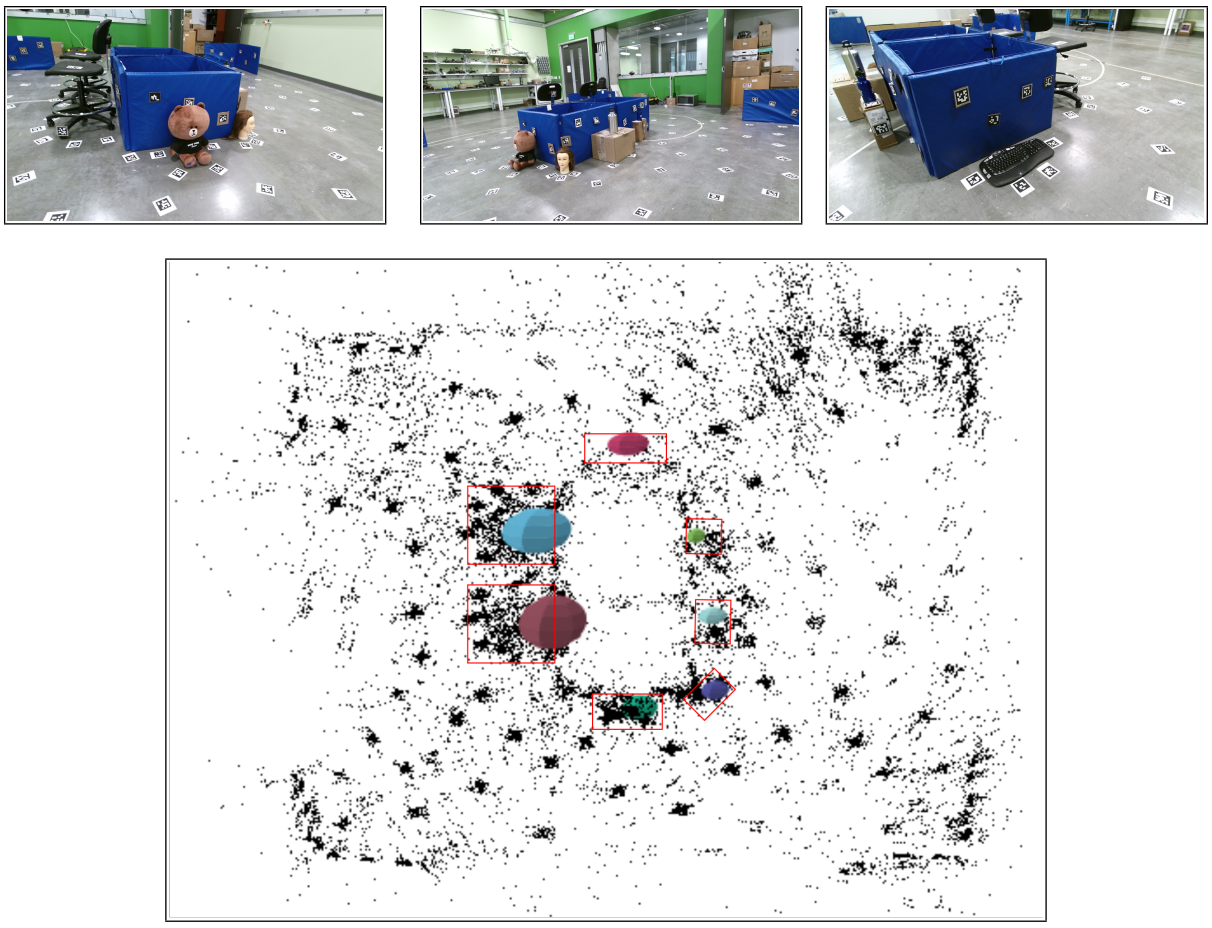


Figure 2.15: The layout of the Scene 2 of UTD Dataset and the mapping results with ellipsoids representing the objects: Object are placed at several locations with tag marks around them in the images. The tag markers can provide a region of the objects in the map similar to a bounding box in object detection.

Table 2.7: Result Statistics for UTD Dataset

Dataset Sequence	Number of Classified Objects	Number of Mapped Objects
Scene 1 [Figure 2.14]	7	7
Scene 2 [Figure 2.15]	10	10
Scene 3	10	9

as a chair, a keyboard, cups, etc. It also has duplicates of the same class objects in the map to show how our method can distinguish between them. The results show that we are able to localize almost all of the objects in the map. The IoU results show the ellipsoid for the classified objects lie within the marked region. It gets an average about 50% overlap with the marked region. A process with 0.5 IoU results are generally considered a good detector.

2.5.4 Comparisons

We now compare our results with other related works in areas such as semantic SLAM, object localization in SLAM, etc. The methods we compare in this section have used their own captured sequence and public datasets. The datasets we use in our evaluation and the compared methods here are not same, but they do share similar characteristics such as indoor office environment, objects, etc. The public datasets some use do not include depth data that our data association method needs, so we cannot run a comparison on these datasets. Also, some of the reported statistics are not similar to our evaluation metric. We will provide a comparison to our method in cases where the metrics are same and convert to our metrics if possible. In general, we define success rate of the proposed mapping process as

$$\text{Success Rate} = \frac{\text{Number of mapped object}}{\text{Number of existing objects in the map}}.$$

In the case of average success rate, we take the average of all the results on different datasets reported on the method.

Mu et al. [67] proposed a non-parametric graph SLAM (NP-Graph) in which they demonstrated association of object detection and localization in SLAM. The results were demonstrated on an office environment and a simulated environment. The results are based on their own datasets and for comparison, we are reporting our results on similar environment that includes office space. Table 2.8 shows the comparison results between our method and

NP-Graph. The average success rate of NP-Graph was 93.87% in their datasets. Our proposed method on public dataset has an average success rate of 75% and on the UTD dataset showed an average success rate of 96%.

Table 2.8: Comparison between NP-Graph and our method

NP-Graph

Dataset	Number of Objects	Number of Mapped Objects
Office	36	31
Simulated Environment	15	15

Our Method

Dataset	Number of Objects	Number of Mapped Objects
TUM fr3 office	22	15
Washington office	20	14
MS office	8	6

Sünderhauf et al. [89] demonstrated a object-oriented semantic mapping with a SLAM solution. Their experiments were performed in indoor environments, which included a single desk, a larger office, a kitchen, etc. Their results were reported as true positive detection (successful mapping of an object), missed objects (unsuccessful mapping of an object) and false detection (mapping an object although it was not there in reality). We use these numbers to report the results in our evaluation metric, the number of existing objects in the map (true positive detection+missed objects) and the number of mapped objects (true positive detection) in the map. The dataset used for comparison are not same so we compare our results based on similar environments. Table 2.9 shows the statistical results of the proposed method by Sünderhauf et al. and the similar data sequences in our proposed method. *In comparison,* Sünderhauf et al. demonstrated an average success rate of 71.68%, and our proposed method on similar datasets has an success rate of 71.62%.

Salas Moreno et al [80] proposed SLAM++, which showed its results in a large common room cluttered with chairs and table. The room contained 35 objects of 5 different classes.

Table 2.9: Comparison between proposed method by Sünderhauf et al. and our method

Meaningful Maps With Object-Oriented Semantic Mapping

Dataset	Number of Objects	Number of Mapped Objects
Lab	60	43
Desk	16	15
Kitchen	4	4
Office	33	19

Our Method

Dataset	Number of Objects	Number of Mapped Objects
TUM fr3 office	22	15
TUM fr2 desk	15	11
Washington office	20	14
MS office	8	6
RGBD scene 6	9	7

Their proposed method mapped 34 objects out of 35 in the environment. The method requires 3D model database of objects and the objects to sit on the ground plane. *In comparison,* we are able to map 26 out of 27 objects in our own 3 data sequences.

Long et al. [56] provided results for semantic segmentation using CNN on datasets such as NYUDv2, SIFT Flow, etc. Their results were presented with mean intersection over union (IoU) metric and our results are reported as IoU. Mean IoU of the image is calculated by taking the IoU of each class and averaging them. The IoU metric used in our method is calculated on the final localization results on the map. Table 2.10 shows the statistical results of semantic segmentation and our proposed method. The semantic segmentation shows mean IoU of 0.34 and 0.39 and, *in comparison*, our proposed method on our dataset demonstrates better IoU statistics, which ranges from $0.42 - 0.68$ with mean of 0.52. Note that the objective and outcomes of [56] are focused on segmentation of classes from a single image. We do not intend to report a direct comparison of our results, but intend to show that our IoU results are in the range, or better, of what is considered successful for the IoU metric.

Table 2.10: Comparison between Semantic Segmentation [56] and our method

Semantic segmentation using FCN-16s

Dataset	mean IoU
NYUv2	0.34
Sift Flow	0.395

Our Method

Dataset	IoU range
UTD dataset	0.42-0.68

2.6 Conclusions

In this chapter, we address the problem of localization of classified objects during SLAM tasks. The number of objects is not known in this method, and the approach can be employed with most classifiers, such as CNNs. The objects are first extracted by classifiers from images, and their location in the map are established using a novel employment of data association and clustering processes. We demonstrated that non-parametric statistics can successfully solve the data association problem between classified objects using depth data. When there is significant motion in between two observations, we are able to associate objects using a depth estimation process.

Our method can easily incorporate RGB-D based SLAM systems and contributes to the filed by including semantics information to the environment through the labeling of detected objects. Experiments on the three public datasets showed our methods were successfully able to determine the probable location and size of objects, modeled as ellipsoids, in the environment. Also we have reported the precision and recall statistics for the NPDA procedure. It possesses a high precision rate, even when motion is introduced. Although the recall drops as motion is introduced in between observation, a higher precision for will ensure that the localization process is robust. We also captured our own dataset and showed IoU statistics using marker tags on our datasets. Our goal of the work is to demonstrate the accuracy of

the object localization and not the registration accuracy of SLAM. Our diversified experimental results are focused on demonstrating the capability of finding these classified objects in the map. In future, we expect the procedure to fully integrate with a SLAM process and demonstrate results on a large scale city environment.

CHAPTER 3

LOOP CLOSURE THROUGH REINFORCEMENT LEARNING

3.1 Introduction

In this chapter, we present a solution to an essential element of SLAM, loop closure. Loop closure is the problem of correctly asserting that an agent has returned to a previously visited area. A SLAM solution without loop closure is essentially just the estimation of odometry. Loop closure was introduced because SLAM odometry suffers from drift over time, leading to accumulated error in the pose and map estimation process. Odometry will only explore the environment through mapping, as shown in Figure 3.1, and is unable to re-localize itself in a previously visited location. Loop closure helps SLAM to identify the intersection of the path in the environment and recover any error by indicating matching locations in its trajectory. SLAM fixes the errors in the map and trajectory through batch optimization using this information. Generally, loop closure has been performed using visual features in VSLAM, but we are proposing a novel solution to loop closure. We demonstrate that loop closure can be solved using a reinforcement learning (RL) algorithm. The information of the location of classified objects added to a map by the previous Chapter 2 can be used as features to help solve the loop closure problem.

We start by describing the mathematical problem behind loop closure and show how it can be solved using a RL algorithm. We show the formulation for loop closure and describe how we incorporate it in RL based on a few assumptions. We will discuss how these changes provide advantage over traditional methods. We provide an introduction to RL and its application in deep machine learning methods. We will then describe the simulated environment for the training of loop closure. The important details of this environment, such as observation, action, reward, structure, etc are discussed in detail, with examples. We provide description of the interaction of the RL-agent with environment, through which

the RL algorithm trains the agent to learn how to perform loop closure. Next, we describe the training procedure and the model used in the procedure. We will describe how we handle an entropy issue with simulated environment for loop closure. Lastly, we demonstrate our results on several different simulated environment scenarios.

3.2 Notation and Assumptions

We use the notation $\mathbb{N} := 1, 2, ..., n$ to represent a set of natural numbers. We use capital letters for random variables and lower case letters for values of random variables and scalar functions. A list of notations used are mentioned below:

s	states or observations from the simulated environment
S	a set of states or observations
t	discrete time steps of an episode
T	final time step of an episode
π	policy of the RL-agent
θ	parameter vector of a network network
π_θ	policy parameterized by θ
A	set of actions the agent can perform in the environment
a_t	an action taken by policy, π at time t, $a_t \in A$
$\pi(s)$	probability of taking an action under policy π in state s
$\pi(a\|s)$	probability of taking an action a under policy π in state s
r_t	reward received at time t following policy, π
γ	discount factor or the penalty to uncertainty of future rewards; $0 < \gamma < 1$
$V(s)$	state-value function that measures the expected return of state s
$\mathbb{E}$	expected value of a random variable

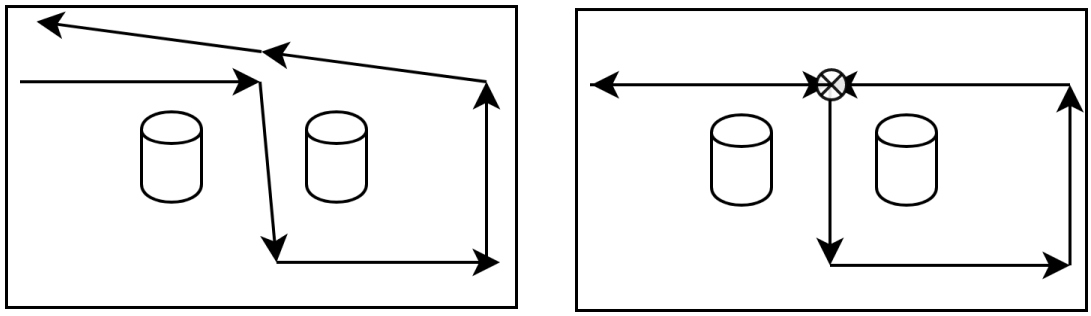


Figure 3.1: The left image is an illustration of odometry only, and the right image shows an illustration with loop closure and odometry.

3.3 Problem Formulation

A robot or a mobile agent moves in an environment and gathers observations from the sensor over time. Whenever the current observation has a sufficient similarity with any of the previous observations, then we can indicate a loop closure to perform error correction in SLAM. We formulate the loop closure problem in a probabilistic Bayesian filtering framework [2]. The overall environment can be modeled as a discrete set of locations or unique observations, where each observations can be described as a distribution over sensor measurements. For a current observation s_t the loop closure problem can be formulated as,

$$j = \quad \underset{i=0,\ldots,t-c}{\arg\max}\, p(LC_t = i | S_t) \tag{3.1}$$

where,

- LC_t is a random variable representing loop closure hypothesis at current time t.
- The event $LC_t = i$ represents that the current position at t matches with the i^{th} (previous) observation
- $S_t = [s_0, s_1, \ldots, s_{t-c}]$ is the history of previous states. The states from t to $t-c$ (where $c \in \mathbb{N}$ is a preset threshold) are not considered for loop closure hypothesis, as they are expected to be similar to current states and will result in repeated loop closure to recent positions.

- if $j = 0$, then it indicates a no loop closure situation.

This formulation requires a full posterior, $p(LC_t|S_t)$ for all $t = 0, 1, ..., t-c$. In general, most implementations of loop closure follow Bayes' rules and assume the Markov property. This assumption decomposes (3.1) as

$$j = \underset{i=0,...,t-c}{\arg\max}\, \zeta\; p(s_t|LC_t = i)\; p(LC_t = i|S_{t-1}) \tag{3.2}$$

where, ζ is a normalization term. This way, the full posterior at t is achieved by the *belief*, $p(LC_t = i|S_{t-1})$ and *likelihood*, $p(s_t|LC_t = i)$.

Our approach to solving this classical problem of loop closure is through training a model by RL. Our solution is dependent upon the locations rather than on the time and all possible previous locations. We have some assumptions on solving this problem:

- The full map of the environment is known to the *model* beforehand as a simulated model
- The number of loop closure locations are predefined

Because of these assumptions, the formulation of the loop closure problem's dependence on previous observations is changed to include only the locations in the map. We demonstrate a similar scenario in a generic graph slam solution. Figure 3.2 shows three scenarios, ground truth trajectory, odometry estimate and loop closure correction based on our assumptions. It shows that if loop closure is performed only at the two predefined locations, we can still recover the approximately correct solution. We can compare this formulation of loop closure to an offline loop closure, which uses stored images of the environment to obtain a bag-of-words dictionary and uses it in SLAM to perform loop closure. Similarly, we are assuming loop closure has a knowledge about the environment only to match the current observations to a previously trained environment. Note that the assumption of knowledge of the grid map for training the RL model is not the same as having a fully established map provided. The

predefined loop closure locations can be set to the intersections of the possible trajectories in the known map or the corner locations in the structure of the map. These locations are provided to the RL model to provide rewards to train

Based on these assumption, the (3.1) and (3.2) is changed accordingly,

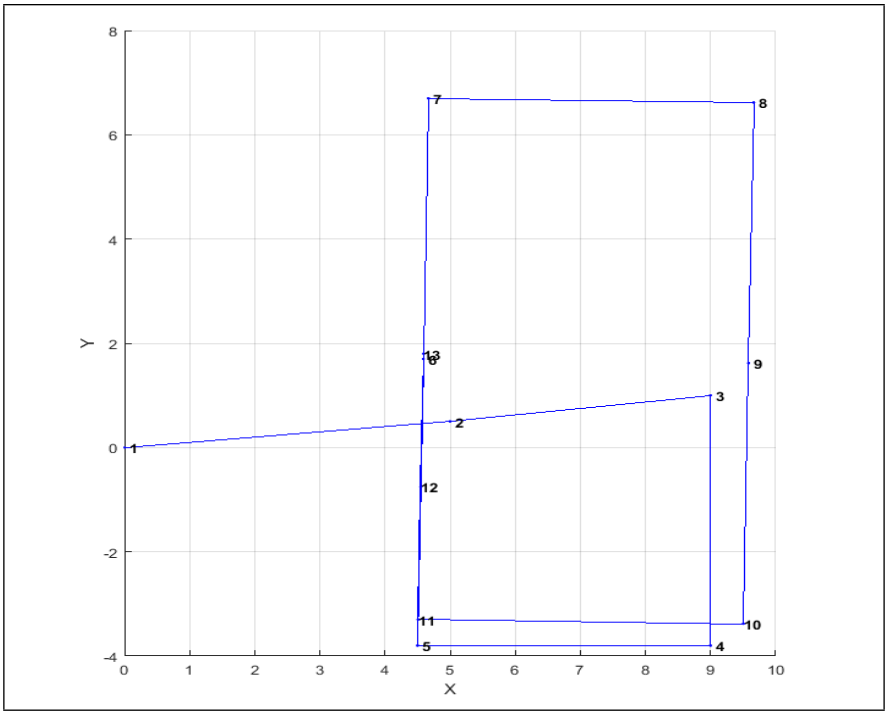
$$\begin{aligned} j &= \arg\max_{l=0,\ldots,n} p(LC_t = l|S_t) \\ j &= \arg\max_{l=0,\ldots,n} \zeta\ p(s_t|LC_t = l)\ p(LC_t = l|S_{t-1}) \end{aligned} \tag{3.3}$$

where, l is the index of possible loop closure locations and n is total number of loop closure locations.

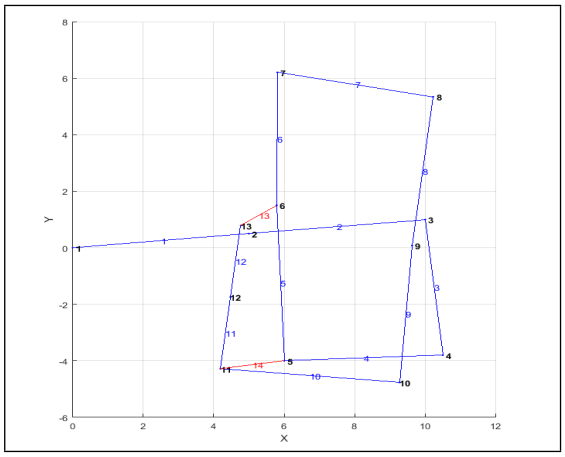
Our solution is to train a policy with RL, which outputs the probability, $p(LC_t = l|S_t)$ or $p(LC_t = l|S_{t-1})$. In this way, LC_t is the action taken by an RL agent, and S_t or S_{t-1} is the observation provided to the RL agent. We will start with the description of the RL framework and then provide training details about the policy to solve the loop closure problem.

3.4 Reinforcement Learning

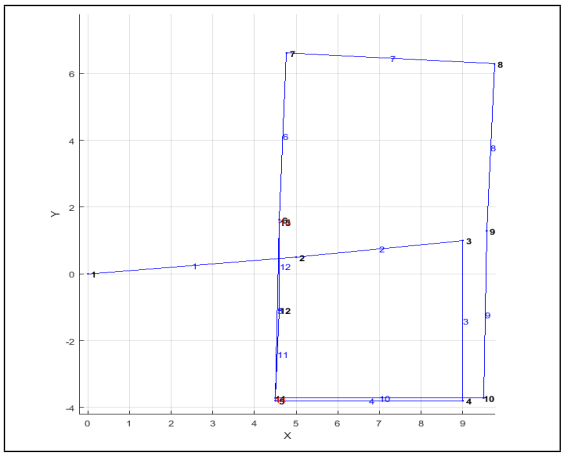
Reinforcement learning is one of the major branches of machine learning. It is a goal driven learning process that seeks to maximize a complex goal. It is distinct from supervised and unsupervised learning in that it depends on interaction with the environment or model and receives a reward as feedback. Supervised learning requires a labeled training dataset, and it then approximates a mapping between the label and data. The classifiers in our object localization are an example of supervised learning. Unsupervised learning is also provided with a training dataset, but there are no labels to the data. This method of self learning develops distinction or inferences in datasets. Our clustering method for object localization is an example of unsupervised learning.



(a) Ground Truth Trajectory



(b) Odometry Estimate



(c) Loop closure output using our assumptions

Figure 3.2: This figure shows three scenarios to demonstrate a result of loop closure based on our assumptions. The top picture shows the ground truth trajectory. In the bottom left, the odometry estimate is shown which has drift error in trajectory. The bottom right images shows loop closures with only 2 locations (indicated in red) and the corrected trajectory after optimization.

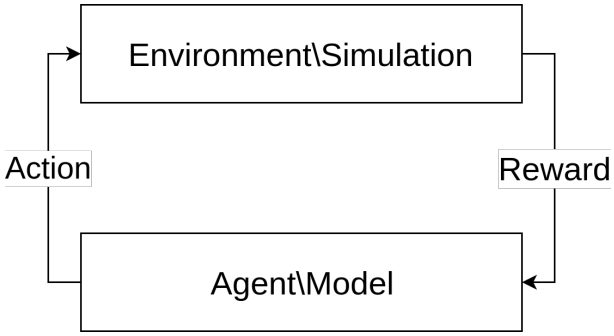


Figure 3.3: A general RL procedure

The basic components of an RL process are a policy, an agent, action and a reward. Sometimes, a model of the training environment is given. The goal of a RL algorithm is to train an agent to make correct decisions when it encounters different states. RL is generally a cyclical process. The process starts with an agent and environment. At each time step t, an RL agent receives a state or observation, s_t, from the environment and takes an action, a_t. Based on the action, the agent receives a reward, r_t, and moves to the next state, s_{t+1}. This interaction with the environment goes on until the agent reaches a final or end state. The RL algorithm attempts to fully maximize the total reward, R received during this period through trial and error process. Figure 3.3 illustrates a typical RL algorithm process.

3.4.1 Markov Decision Process

A Markov Decision Process (MDP) is a discrete stochastic process that describes the interaction of an RL agent with the environment. All states of a MDP follow the Markov property, meaning future and past states are conditionally independent, given the knowledge of the present state. A MDP in RL generally consists of five elements, $\{S, A, Pr, R, \gamma\}$. Pr is the transition probability distribution that describes the probability of receiving a reward r and transition into a state s' based on current state-action pair $\{s, a\}$.

This dissertation is focused on the episodic nature of RL with a finite MDP. The interaction with the environment is divided into sequences or trajectories with states, action and

reward. These sequences or trajectories typically follow the format

$$\underbrace{s_0 \rightarrow a_0 \rightarrow r_0}_{sequence\ 0} \rightarrow \underbrace{s_1 \rightarrow a_1 \rightarrow r_1}_{sequence\ 1} \rightarrow ... \rightarrow \underbrace{s_T \rightarrow a_T \rightarrow r_T}_{sequence\ T}. \tag{3.4}$$

3.4.2 Partial Observability

The environment for our loop closure provides the RL agent only a partial observation that may include noisy and limited information about the current state. This means we only have observations of a portion of the environment rather than a full observation. In this case, the RL agent needs to accumulate the information from previous steps to train and optimize a policy. Usually this is done through a set of previous observations, O. At any time step t, if y is the partial observable state, then $O_t = \{y_0, a_0, y_1, a1, ..., y_{t-1}, a_{t-1}, y_t\}$. Also, an observation model can also help update the belief of the RL agent. In general, a partially observable Markov decision process (POMDP) is not much different from a fully observed one, as we can use O_t to derive an approximately fully observed state. Most real world scenarios are designed as POMDP.

3.4.3 RL Policy

The RL algorithm trains the agent using trajectories stated in (3.4) to find a policy, π. This will map the states s to the probabilities of selecting an action $a \in A$. In RL, a policy can be deterministic, $\pi(s)$ or stochastic, $\pi(a|s)$. Deterministic policies are mostly used in deterministic environments that have no uncertainty involved in the states or the actions. Examples of such cases may be chess, pong, solaris, etc. Stochastic policies are mostly used whenever there is an uncertainty involved in the states or actions. It outputs a probability over all the possible action on a state. Most real-world scenario and POMDPs require a stochastic policy to train an agent. In this dissertation, we are interested to find a stochastic policy for loop closure.

The stochastic policy $\pi(a|s)$ is the probability of taking an action a while being in state s. A policy gets updated in every iteration of training to maximize the overall return. A policy can be evaluated using *state-value function* or *action-value function*. A state-value function, denoted as $\upsilon_\pi(s)$, is the expected return if starting at state s and following policy π thereafter

$$\begin{aligned} \upsilon_\pi(s) \doteq & \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} | s \right] \\ \upsilon_\pi(s) = & \sum_a \pi(a|s) \sum_{s_{t+1,r}} p(s_{t+1}, r|s, a)[r + \gamma \upsilon(s_{t+1})] \end{aligned} \tag{3.5}$$

$\upsilon_\pi(s)$ evaluates how much can be achieved from the current state. The action-value function, denoted as q_π, is the expected return if starting at state, s, performing an action a and following policy, π afterwards

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} | s, a \right]. \tag{3.6}$$

A solution to an RL problem is a policy that provides the highest return in the training sequence. A RL policy gets updated after every training period, so there is one policy in the run that is better than all others. This policy is known as the optimal policy, denoted as π_*. So the *optimal state-value function* is defined as,

$$\upsilon_*(s) \doteq \max_\pi \upsilon_\pi(s), \tag{3.7}$$

and the *optimal action-value function* is defined as,

$$q_*(s, a) \doteq \max_\pi q_\pi(s, a) \tag{3.8}$$

3.4.4 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is the subset of of RL methods that use deep neural network as a function approximation for the policy. Deep neural networks perform better as function approximators for nonlinear cases compared to other existing methods. The success

of deep learning in speech [21] and object recognition [48] has paved the way for it to be integrated in RL. Most initial success in RL came from linear or tabular functions, but the first breakthrough using deep learning came from [64]. Mnih et al. demonstrated that DRL is able to play Atari games using a variant of Q-learning. He used the game screen as a input to a CNN and showed results on par with human performance. [85] trained a RL framework on the game of Go that defeated an European Go champion and achieved a 99.9% winning rate against other players. They introduced a novel method of combining Monte Carlo simulation with value and policy network to train itself from supervised and self-play data. Levine, et al. demonstrated an end-to-end training using CNNs and guided policy search from images to control a robot's arm or motor [54]. Their experiments showed it was able to perform several complex manipulation skills, such as grasping different shaped objects. There have been multiple algorithms proposed by [37], [81], and their variants [98], and improvements been introduced in the last few years. Several open-source platforms have been introduced to explore algorithms for RL. OpenAI has demonstrated several improvement of DRL and introduced numerous simulated environments to test RL algorithms. Recent advancement has been shown to overcome some of the drawbacks of traditional RL, such as learning to learn [30], demonstrated learning adaptability to new task without a reward feedback, etc. Recent advancement has mostly focused on simulated environments rather than on real-world dataset. The success and promise of DRL in these simulations can be extended to real world robotics application with efficient and accurate design of the simulation. In this dissertation, we demonstrate one of the classical robotics problem, loop closure in DRL and showcase how it can be applied to a indoor environment.

3.4.5 Policy Optimization Techniques

There are many existing methods to optimize a stochastic policy in RL. Policy-based optimization methods are different from value-based methods such as TD learning and Q-learning. Policy-based optimization methods view RL as numerical optimization problem in

which the expected return is optimized against the policy parameters. There are generally three ways to optimize a policy:

- *Policy Iteration* methods optimize the policy by alternating between estimating υ_π and improving π. These method may converge in relatively few iterations, as they always moves toward improving the policy. However, for large systems of equations, it becomes computationally expensive [91].

- *Derivative free optimization* (DFO) treats the problem as a block box function, which in turn provides better policy parameters. Cross-Entropy methods and co-variance matrix adaptation are a few DFO algorithms. These are vulnerable to variance reduction and converging to local optima [94].

- *Policy Gradient* methods model and optimize π directly with a parameterized function with respect to θ. It is one of the most popular method in practice and provides better results on large training datasets.

An in-depth discussion of these methods is outside the scope of this dissertation. So, we will describe here a policy optimization technique related to our proposed method. We use a gradient-based stochastic policy optimization method that operates on a discrete action space. Policy gradient methods can generally be grouped into two categories, on-policy and off-policy. On-policy primarily interacts with a *target policy*, π. It follows and obtains samples from a target policy, which is the same policy the RL is being trained on. On the other hand, off-policy has a second policy named *behavior policy*. In this setting, the agent trains the *target policy* and selects actions based on the *behavior policy*. REINFORCE is a Monte Carlo on-policy gradient method that collects episodic trajectories of policy, π and uses them to update policy parameters. A baseline value is subtracted from the return to reduce the variance. Actor-Critic is a another popular method, in which a model is split

into two. The two components are policy model and value function. The actor or policy decides which action a to take in a state s, and the critic evaluates the performed action using a value-based function. We use actor-critic in the implementation of the RL algorithm. Actor-critic methods have certain advantages over other methods such as

- This method can acquire multiple episodes of sample trajectory and then optimize based on all sample trajectories, whereas previous methods were based on a single run of an episode. Using multiple parallel trajectories helps incorporate various samples in one batch of episode.

- The use of two separate modules enables to have two different updates where the critic helps the actor by fixing its wrong actions

Our parameterized stochastic policy is denoted as $\pi_\theta(a|s)$. Here, $\theta \in \mathbb{R}^n$ is the parameters of a neural network. The policy is improved using a performance measure. It is usually defined as

$$\begin{aligned} \mathcal{J}(\theta) &\doteq \upsilon_{\pi_\theta}(s_0) \\ \mathcal{J}(\theta) &= \sum_{s \in S} \eta(s) \upsilon_{\pi_\theta}(s) \\ \mathcal{J}(\theta) &= \sum_{s \in S} \eta^{\pi}(s) \sum_{a \in A} q_\pi(s,a) \Delta\pi(a|s) \end{aligned} \tag{3.9}$$

where, s_0 is the initial starting state, and $\eta(s) = \sum_{k=0}^{\infty} Pr(s_0 \rightarrow s, k, \pi)$ is the on-policy distribution under π. Equation (3.9) can be expanded using a derivation from [91],

$$\begin{aligned} \nabla\mathcal{J}(\theta) \doteq & \nabla\upsilon_{\pi_\theta}(s_0) \\ = & \sum_{s \in S} \eta(s) \sum_{a \in A} \pi_\theta(a|s) q_\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\ = & \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla_\theta \ln \pi_\theta(a|S_t) \right]. \end{aligned} \tag{3.10}$$

So, the policy can be optimized using,

$$\theta_{k+1} = \theta_k + \alpha \nabla \mathcal{J}(\theta) \tag{3.11}$$

where, α is the learning rate and k is iteration number in the optimization. There are numerous policy gradient theorems based on (3.10). The gradient is computed over the sampled trajectory, based on the policy π_θ. Since the sampled trajectory can take different actions depending on the states, the variance between the samples can be high. Introducing a bias or baseline in (3.10) helps to reduce the variance. A good overview of mitigating such problems has been proposed and discussed in detail by Schulman et al. [83]. If the sample size is very large (meaning if $|S| \to \infty$), then the variance can be low. However, the gradient for a large sample size data is computationally expensive and not feasible. Our method also encounters this issue in sampling the correct actions in the trajectory of an episode. We address this with entropy maximization, which is discussed in section 3.6.

The policy gradient theorem with a function approximation has been proven for on-policy methods for stochastic policies by Sutton [92] and for deterministic policies by Silver et al.[86]. In recent works, policy gradient has been proven to work for off-policy, as shown by Degris et al.[23]. He proposed an algorithm named Off-PAC (Off-Policy Actor-Critic) and proved the convergence in the standard off-policy setting. There is a justified proof by Degris et al. [23] on how off-policy can guarantee policy improvement and reach an optimum policy. We shortly summarize this in this dissertation. Off-policy introduces a *behavior policy*, b. The objective function with b can be (3.9),

$$\mathcal{J}(\theta) = \sum_{s \in S} \eta^b(s) \sum_{a \in A} q_\pi(s,a) \nabla \pi(a|s) \tag{3.12}$$

and its gradient,

$$\nabla \mathcal{J}(\theta) = \sum_{s \in S} \eta^b(s) \sum_{a \in A} \left[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \right]. \tag{3.13}$$

Since the term on the right, $\nabla q_\pi(s,a)$, is hard to estimate for off-policy, it is omitted in most algorithms, and without it, policy has been shown to still guarantee improvement over [23].

So (3.13) can now be rewritten as an expectation

$$\begin{aligned}\nabla \mathcal{J}(\theta) =& \mathbb{E}_b \left[\sum_{a \in A} q_\pi(S_t, a) \nabla \pi(a|S_t) \right] \\ =& \mathbb{E}_b \left[\sum_{a \in A} b(a|S_t) \frac{\pi(a|S_t)}{b(a|S_t)} q_\pi(S_t, a) \frac{\nabla \pi(a|s)}{\pi(a|s)} \right] \\ =& \mathbb{E}_b \left[\frac{\pi(a|S_t)}{b(a|S_t)} q_\pi(S_t, a) \nabla \ \ln \pi(a|s) \right]\end{aligned} \tag{3.14}$$

where, $\frac{\pi(a|s)}{b(a|s)}$ is the importance weight of *target policy* to *behavior policy*. This forms the basis of off-policy gradient methods.

Policy gradient methods have been extended from (3.10) and (3.14) to various expression to handle issues with bias, variance and convergence [82, 23]. This form can be generalized to the following expression,

$$\hat{g} = \mathbb{E} \left[\sum_{t=0}^{T} \psi_\pi \nabla_\theta \ \ln \pi(a_t|s_t) \right] \tag{3.15}$$

where, $\hat{g}$ is the gradient estimator for policy gradient method, ψ_π is the advantage function estimator and the expectation is taken over a number of batch samples. In this dissertation, we incorporate both trust region policy optimization [82] and proximal policy optimization method (PPO) [84] to find the loop closure policy. A full detail of these method is out of scope for this dissertation. So we describe the objective function used in training the policy. We aim to optimize the objective function as proposed by PPO using batch samples with the following,

$$min \left(\frac{\pi_{\theta_{new}}(a|s)}{\pi_{\theta_{old}}(a|s)} \ \psi_{\pi_{\theta_{old}}}, \ g(\epsilon, \psi_{\pi_{\theta_{old}}}) \right) \tag{3.16}$$

where,

$$g(\epsilon, \psi_{\pi_{\theta_{old}}}) = \begin{cases} (1+\epsilon)\psi_{\pi_{\theta_{old}}} \text{ if } \psi_{\pi_{\theta_{old}}} \geq 0 \\ (1-\epsilon)\psi_{\pi_{\theta_{old}}} \text{ if } \psi_{\pi_{\theta_{old}}} < 0 \end{cases}. \tag{3.17}$$

Here, the subscript *new* and *old* indicate the parameters of network after and before an optimization, respectively, and ϵ is a small parameter that is the limitation of the update of

new and old policy. The advantage function estimator is,

$$\psi_{\pi_\theta} = \gamma^{T-t}V(s_T) - V(s_t) + \sum_{i=t}^{T} \gamma^{i-t} r_i \tag{3.18}$$

where T is the duration of the trajectory from an episode of the training environment. Since our method involves an actor and a critic, the actor is based on the policy optimization method we just discussed, and the critic is a value function approximation parameterized by ψ. This is similar to the target policy with the difference being in off-policy it approximates the policy and the critic approximates the value function. We describe the process in the training procedure of Section 3.6.

The same components that essentially fuel the RL framework also create problems for it to be directly applied to a real world application. One of the reasons are the actions taken by the RL agent may involve hazardous situations with the environment and object. Therefore, a simulated environment is widely used in RL to avoid such situations during the training process. The method we propose here is trained in a simulated environment. We will first showcase a simulated environment for the RL agent and go through the details about the training procedure. Later, we demonstrate how this can extended to a real world dataset.

3.5 Simulated Grid World for Training Loop Closure

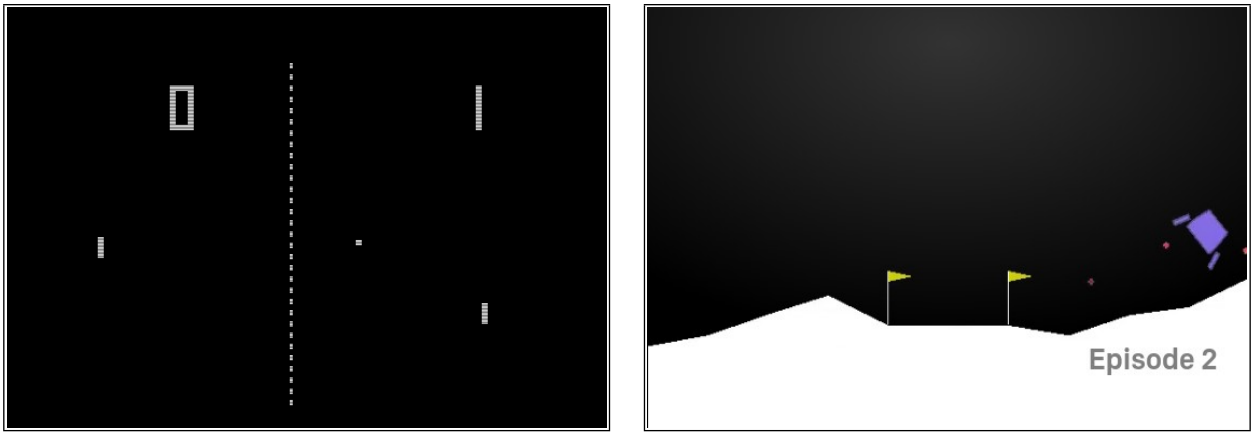
The learning process of RL is modeled to provide a reward signal for appropriate actions and provide a penalty or terminate the process when the actions are unwanted or incorrect. During training, the action taken by the agent can incur damage to the agent and/or the environment if it is performed in a real-world environment, such as collisions, grasping of an object, etc. Also, model-free RL requires a large volume of data to train and produce satisfactory results, as well as data to validate the trained policy. This amount of data is harder and takes much longer to obtain from a real-world environment compared to a simulated environment. For these reasons, a simulated environment is often easier than a

real-world environment to train an RL. Therefore, we designed a simulated environment for loop closure scenarios for a RL algorithm. Most recent advancements in RL have been trained and demonstrated using simulated environments. In the beginning, most RL algorithms were tested in a 2D grid-based environment. Recently, the environment has included complex tasks such as memory, 3D perception and navigation. OpenAI has a vast collection of simulated environment such as CartPole, BipedalWalker, LunarLander, etc. to test RL efficiency. Mnih et al. [64] at Deepmind demonstrated their RL algorithm on five Atari 2600 Game. Figure 3.4 shows a few of these simulated environments that were tested for RL efficiency in recent papers.

We demonstrate in this dissertation that the loop closure problem is solvable using a simulated grid world. Currently this grid is restricted to a two-dimensional representation. The agent will be allowed to move and observe in this grid environment, so it can be trained based on these observations. The agent referred here is taking the loop closure detection action. To achieve this, there are a few elements that need to established: environment structure, the movement of the agent in the grid world, and the observation of the agent. We describe this in the following section.

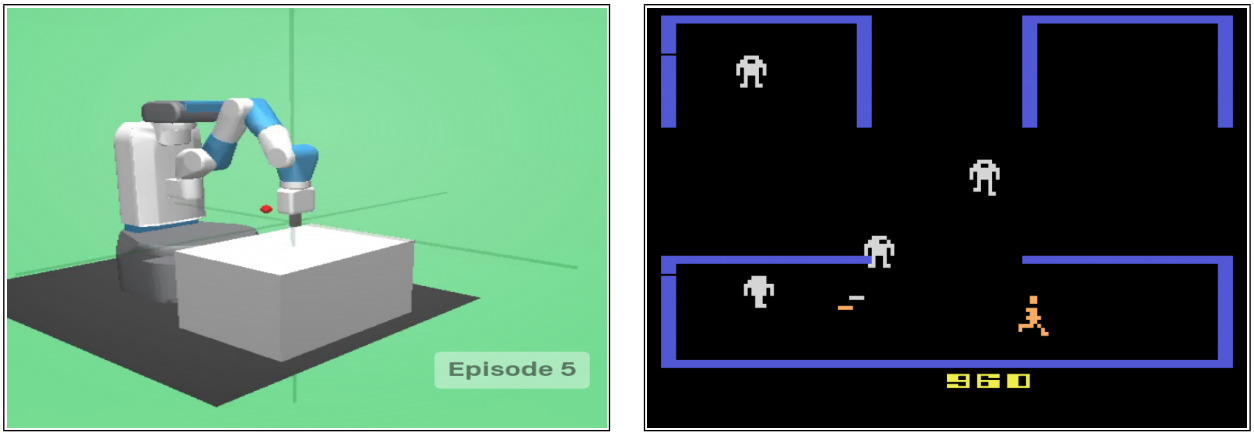
3.5.1 Grid World Structure

We will denote the grid world environment as E, which is a $N \times M$ square grid. This size should be chosen to be similar to the environment the RL-agent is trying perform loop closure. It is currently designed to represent an indoor environment. As such, each square in E will be empty space, occupied by the agent, occupied by feature blocks, or occupied by walls and/or obstacles. A feature block may represent an object, a feature point or landmark in the real world environment. In the current configuration, the grid contains multiple feature blocks placed at the walls in the environment. Each feature block is distinguishable from each other, similar to the associated classified objects discussed in Chapter 2. For visualization,



(a) ATARI Pong

(b) OpenAI Luner Lander



(c) Robotic Grasp

(d) ATARI Berzerk

Figure 3.4: Several different simulated environments have been successfully used in recent DRL.

every feature block is given a different color to distinguish it from others. A feature block can occupy a single or multiple squares in E. An agent is allowed to move through the feature blocks existing on the environment. Figure 3.5 shows a few possible configuration of E. The wall or obstacle square prevents motion through the position of the block. It is used to represent any wall or non interacting blocks in the environment. They are represented with white color when visualizing E. The white blocks in Figure 3.5 for the configurations represent walls.

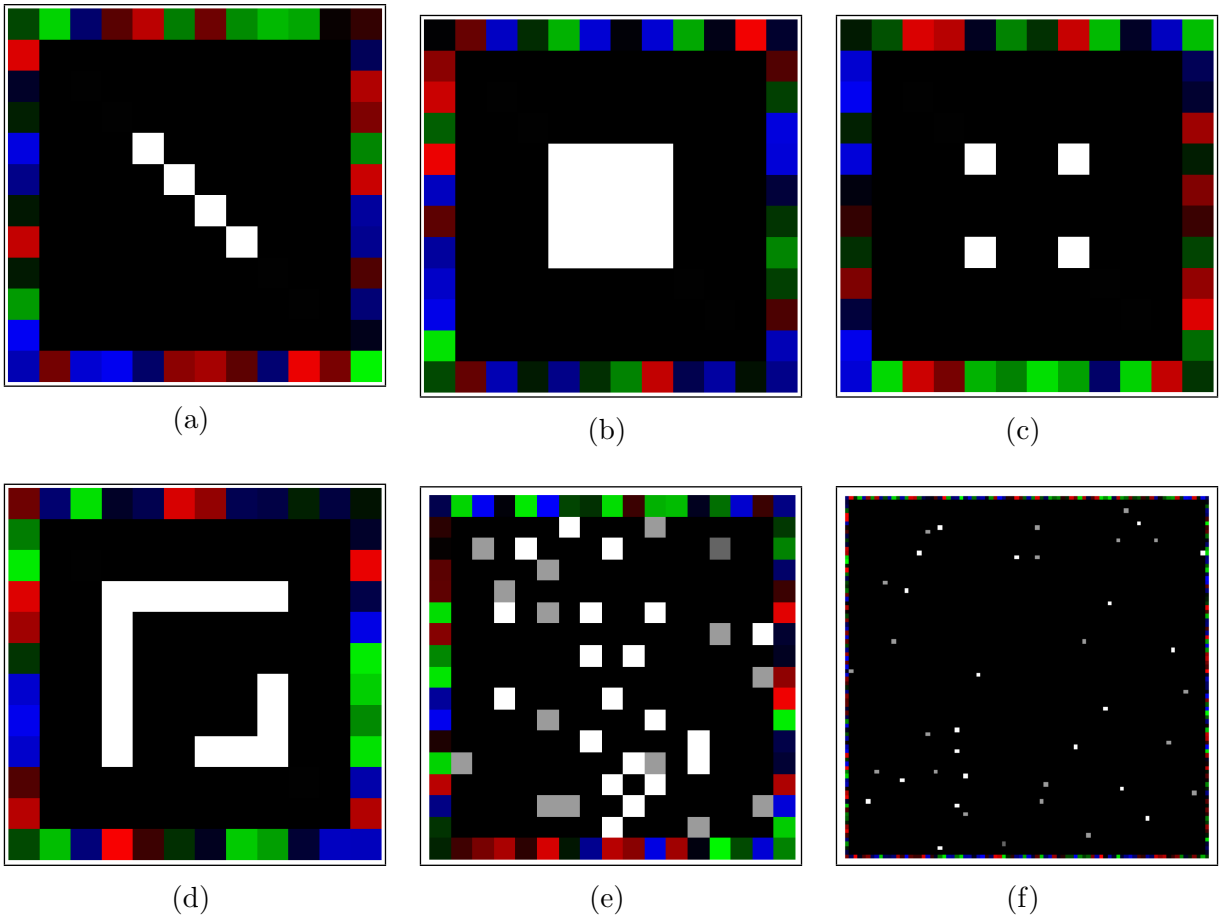


Figure 3.5: The figure shows six different grid configuration of sizes 10×10 (a-d), 15×15 (e) and 84×84 (f). The features block are placed around the walls, and the white blocks are the obstacles in the environment.

3.5.2 RL Agent for Loop Closure

The agent is a movable block in the grid E and takes actions based on the trained policy. The movement, direction and step size of the RL-agent in the grid environment is constrained. The agent is allowed to move only one or two blocks at a time. The movement and the direction of the agent at any time can be one of four ways: up, down, left and right. The direction and movement the agent is generated from a random distribution. The random distribution generates a different combination of direction and movement for the agent at each time, which allows the reinforcement learning agent to generate a different trajectory for

each training period. In practice, the random numbers are sampled from a uniform, discrete distribution. We always start with a different seed for the random number generator in our simulation, so one episode of simulation is not similar to the other episode. Figure 3.6 shows the observations of the agent from sequential movement and its heading indicated by an arrow in E.

3.5.3 Observation Space for Loop Closure Grid

The RL-agent estimates all relevant information about the current state of the environment over time necessary to fulfill the task successfully. The movement of the agent in E and the outcome from the actions of the agent is transcribed to policy through the observation space. The observation of E at each step is in the topological form or top down view of E. At each step in the simulation, the agent generates an observation based on its current "cone of vision". The "cone of vision" is a 60° angle of vision (30° to the left and right) that extends as an imaginary cone. The cone of vision is used here to maintain similarity with an image sensor or human perception. The observation of the agent at any time includes the feature and obstacle blocks in the the full extension of "cone of vision". Figure 3.6 shows four sequential observations from the simulations. It shows the features and obstacles in the "cone of vision" of the agent at each step.

3.5.4 Action Space for loop closure Grid

The RL algorithm trains the policy to take appropriate actions to maximize rewards. The actions of the RL agent determines if the RL algorithm can optimize the reward for the policy. The action space for the RL agent can be continuous or discrete. If the action is not discrete, it is difficult to select an action from the huge continuous action space [53]. The time complexity of the action selection in continuous space is much higher compared to the a discretized action space. We opted to use a discrete action space, as we do not need a large

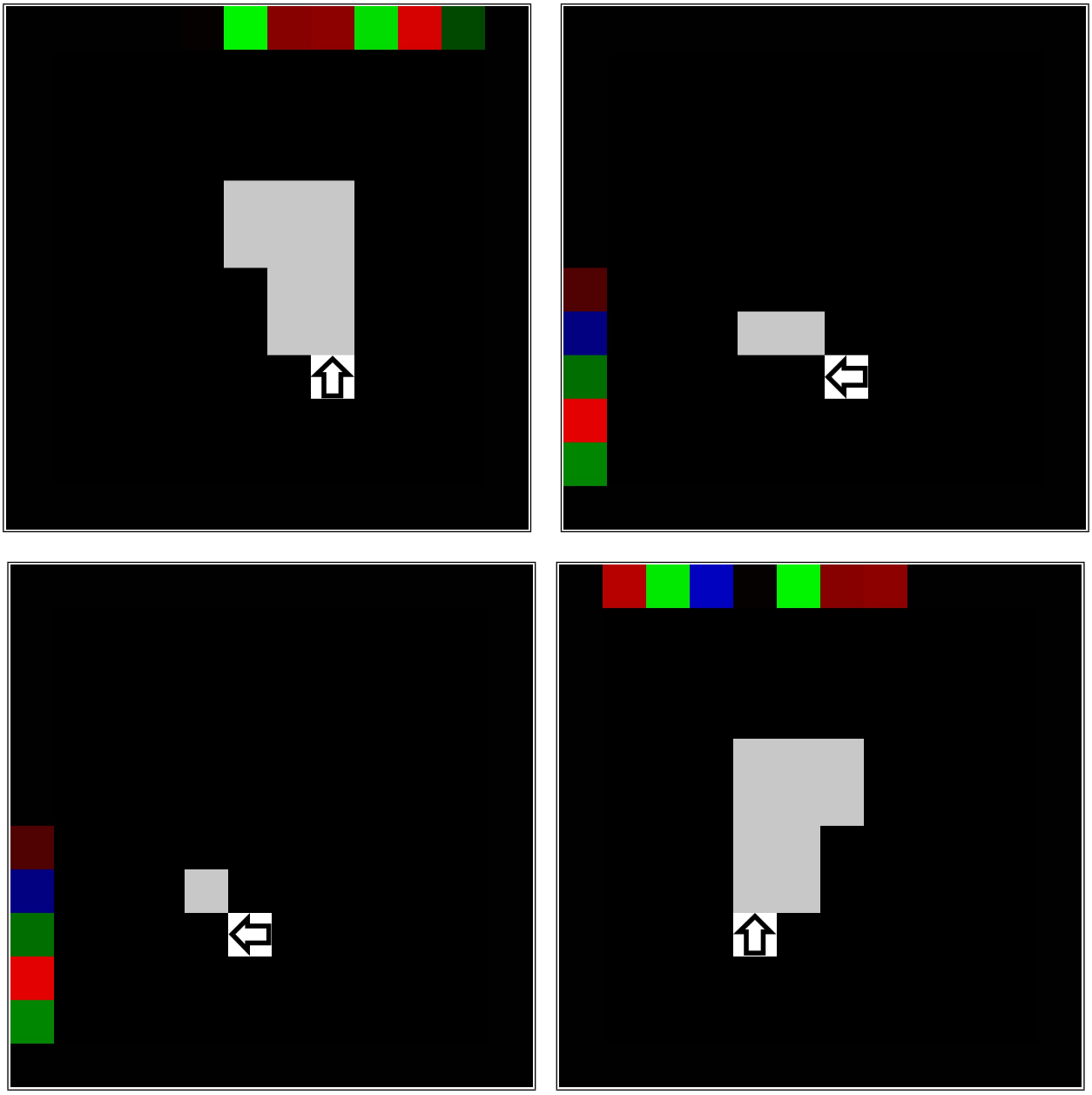


Figure 3.6: The figure show four observations the agent provides to the policy. The arrow in the agent block shows its current direction of view and the observation is generated using the "cone of vision" from the direction and position.

number of action set at each step, and each action itself is independent from others. Even though most real-world actions are continuous, such as steering of a car, in our case, we are trying to perform loop closure at the predefined locations in the simulation. The number of these locations are discrete, so we opted to use a discrete action space. The number of actions is equal to the number of loop closure locations in E plus a no loop closure action. Let $\{i, j\}$ indicate the current position of the RL agent in E and $l^t \in \mathbb{N}^{N \times M}$ be the location history for E at time t, such that $l^t_{i,j}$ indicates the number of times the agent has visited position $\{i, j\}$. Then, the action space for the RL agent can be specified as

$$A_t = \begin{cases} \text{Loop closure action}, & a = i \times N + j; a \in \mathbb{Z}^+; \quad a < N \times M; \quad \text{if } l^t_{ij} > 0 \\ \text{No loop closure action}, & a = 0\ ; \qquad\qquad\qquad\qquad\quad \text{otherwise} \end{cases}. \tag{3.19}$$

When the RL-agent is at location $\{i, j\}$ in E then, for loop closure for the appropriate action, $a = i \times N + j$. Figure 3.7 shows the loop closure locations in grey, and its appropriate action in the corresponding blocks. For example, the position at $\{2, 2\}$ the loop closure action should be, $a = 2 \times 10 + 2 = 22$. At all other location in black, the appropriate action would be 0.

3.5.5 Reward function for Loop Closure Environment

A reward function specifies the positive or negative feedback for the current state or state-action pair resulting from following a policy. The main purpose of any RL is to maximize the cumulative reward received by the RL agent over the full trajectory. In general, a reward is provided every time the agent has taken an action in the environment. If a reward is not provided, or a zero reward is provided, then the policy will not be able to evaluate the action taken by the agent. Without a proper reward function, RL may not be able to learn anything or find convergence in its duration of training. So our agent takes either a no loop closure or a loop closure action in every step of the environment, and a reward is provided based on the current state of the environment.

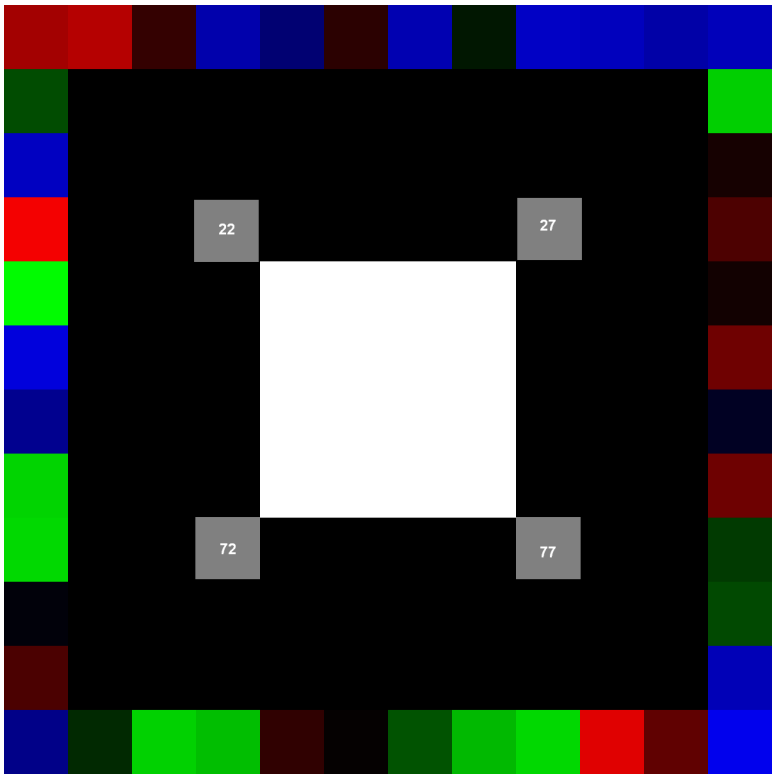


Figure 3.7: The figure shows the locations of the loop closure as grey blocks with numbers.

The reward function is the primary source of information for improving the policy, π_θ. The modeling of the reward is based on what we are trying to teach the agent and what the agent can learn over time. The reward needs to clearly indicate to the policy what the environment considers a positive reward and what it penalizes, so the agent can distinguish between the two cases easily. The distribution of reward provided in the environment can be sparse or dense. Sparse rewards are provided where the environment is usually designed to perform only one or two tasks such as hitting a ball, picking an object, etc. Dense rewards are designed for environment where multiple tasks are needed to solve a problem such as folding a paper or box. Both rewards are given in the form of positive or negative discrete number. Since sparse rewards are returned in a relatively low number of states or observations, they can be provided as a large number, so the total reward for taking the actions corresponding to the sparse reward will be larger in comparison to other rewards.

This way, the policy will be able to optimize to take the actions that provides the sparse rewards. In robotics applications, a sparser reward is more applicable than a denser reward, as it is easier to maximize a sparser reward using a RL algorithm. For example, in the OpenAI environment "CartPole-v1", a reward is provided to the agent when it can successfully keep the pole upright and ends the episode when the pole falls. "FetchPickAndPlace-v1" provided rewards when a robotic arm can pick a box and move it to its desired goal location, which is randomly set at the start of training.

Since the goal of our RL agent is to perform loop closure, we designed the reward function to give a reward whenever the agent performs a correct action and penalty for performing an incorrect action. As discussed earlier in Section 3.3 and in (3.3), we have a fixed set of loop closure locations in the environment. The locations of loop closure are primarily the intersecting locations in the trajectory of the agent. Since we assume the map of the environment is known to the agent, the locations the agent can visit in the map are also known. The intersections of all possible trajectories of the agent in the simulated environment can be estimated, and the pre-determined loop closure locations can include these intersecting locations. However in our training, we have manually set the loop closure locations in the environment. For example, in Figure 3.5a, the four corners of the middle wall were included in the loop closure location.

We keep a history of the locations the agent has traversed over the full period of the current episode. When the current location of the agent has already been visited, and it is in the set of loop closure locations, then a loop closure is the correct action. When checking for loop closure at time t, the location history from the start to previous time $t - c$ is used, as discussed in the loop closure problem formulation in Section 3.3. This is to prevent immediate last positions becoming candidates for loop closure in the motion of agent. The training episode is terminated, and a heavy penalty is given in the following cases

- If no loop closure is performed for more than four loop closure locations. This is intended to provide heavy penalty for missing loop closure positions consecutively.

- If loop closure is performed multiple times consecutively at no loop closure positions. This is intended to stop performing loop closure repeatedly.

Given that the grid size of E is $N \times M$, then after every movement of the agent the location history, l_{ij}^t is updated,

$$l_{ij}^t = l_{ij}^{t-1} + 1, \text{if the agent at time } t \text{ is at } (i,j) \text{ of } E.$$

We define a linear map function $f(l_{ij}) : l_{ij}^t \to A$, where $A \in \mathbb{N}$ and X is the set of predefined loop closure locations, $X \subseteq Y$. The reward function is defined as

$$r_t = \begin{cases} +100 & a_t \in X & \& l_{ij}^t > 0 \\ +10 & a_t = 0 & \& \quad f(l_{ij}^t) \notin X \\ -100 & a_t \neq f(l_{ij}^t) & \& \quad f(l_{ij}^t) \in X \end{cases} .$$

Based on the locations of the loop closure, the reward for correct and incorrect action can be sparse in the trajectory, meaning the loop closure actions are only encountered in a few locations. To make sure the loop closure actions are included in the trajectory of the training, we also perform an entropy maximization for including the loop closure actions. This will be discussed in detail in the training procedure and experiments.

Figure 3.7 shows four loop closure position in E highlighted as grey block with numbers indicating a correct loop closure action. When these position are visited again in the path of the agent and it takes loop closure action, it receives positive reward.

3.6 Training Procedure

We have discussed earlier policy gradient methods to train on the loop closure environment. We are using a modified version of the proximal policy optimization technique, as described in section 3.4.5. The method is implemented using an actor-critic algorithm, where the policy is the actor, and the critic is the value function that suggests the policy updates. We represent

the policy with convolutional neural networks (CNNs) followed by a long short-term memory (LSTM) cell. The CNNs extract features for the policy to take loop closure action, and the LSTM module is added handle memory as the policy needs to store information about the loop closure locations.

The actor-critic mode can be achieved in two different ways, based on network design and the type of synchronization. We used a synchronized actor and critic mode, meaning all the actors in the environment will finish their simulation before an update from the critic. Both the actor (policy) and critic (value approximator) can be trained using a shared and dual networks. In the shared network design, the output the network provides values for both policies. On the other hand, it can follow a dual network design, where both actor and critic have their own network. The design of a dual network is simple and much more stable compared to the shared network.

Figure 3.8 shows the architectural overview of our network using dual network design. The architecture is composed 3 convolutional layers and an LSTMCell. We have tested our algorithm on different size grids, and we are reporting our results on 10×10, 15×15 and a 84×84 size grid. The observations are generated with 84×84 resolution. Since we have variable sizes of grids, we up-sampled the observations to match the input resolution. A larger grid size environment may require more layers in the architecture and different kernel size. The training process uses a synchronous actor critic method. Multiple models of the architecture shown in Figure 3.8 are employed in parallel to capture the data. A step by step process of training with parallel actors is given by Algorithm 3.

The training was completed on 6 different simulated environments. For each different environment, there were on at least 500 different episodes of training completed. For each set of these episode, the agent started from a new position to allow exploration. The motion of the robot was generated from random distributions using a different seed for each episode. This makes sure the policy has different observations in the episodes to learn and does not over-fit on certain set of features.

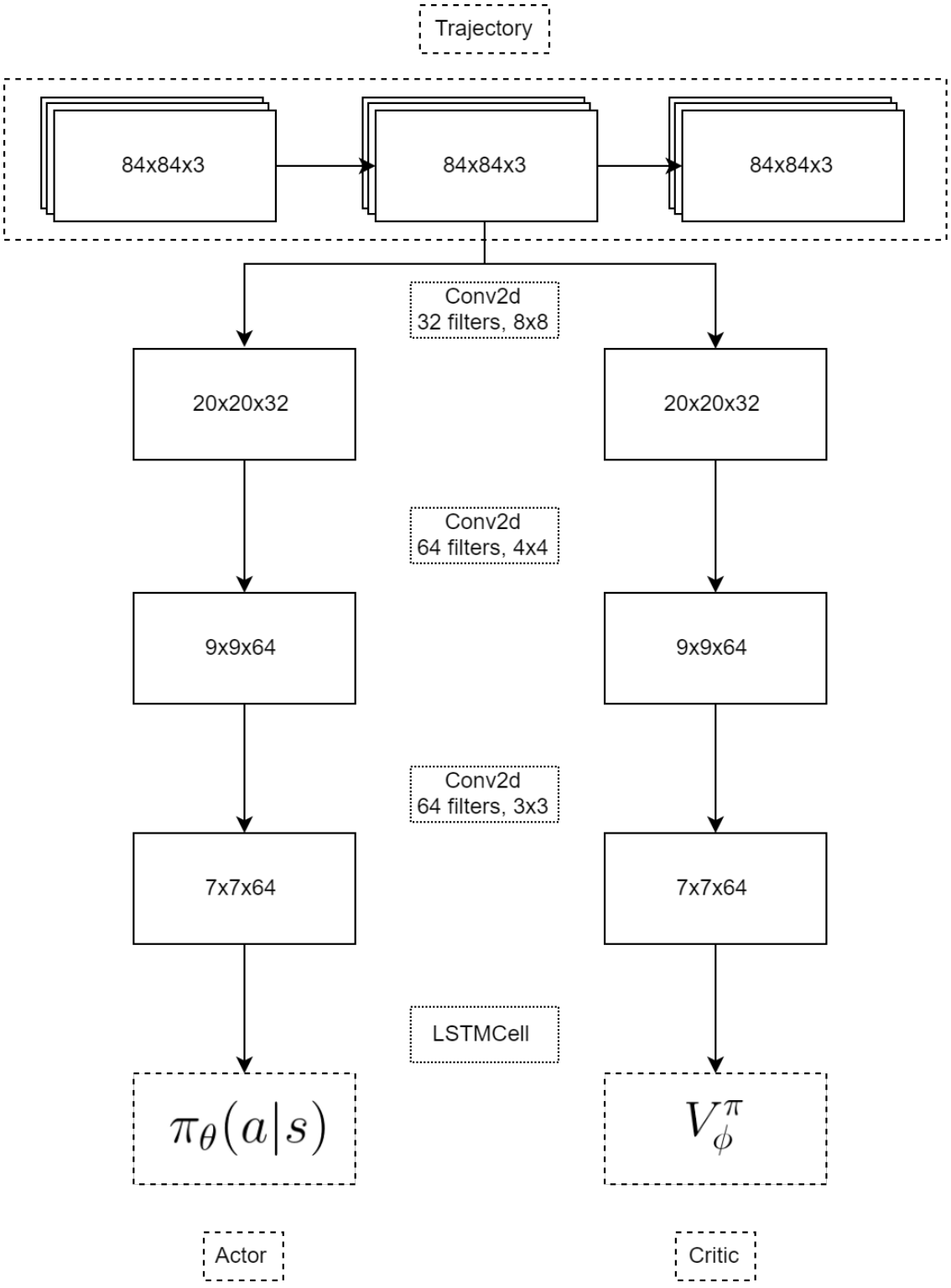


Figure 3.8: Architecture Overview of the policy

Algorithm 3 Training Procedure for the Loop Closure RL Policy

Initialize policy and value function parameters, θ and ϕ accordingly at random,
Initialize loop closure grid environment,E with predefined set of loop closure locations,
for actor, $k = 1, 2, ..., N$ **do**
  for $t = 1$ to $b_1 < T < b_2$ **do**
    obtain a state from E, s_t
    take an action, $a_t \sim \pi_\theta(a|s)$,
    obtain next state, s_{t+1} corresponding reward, r_t
    check entropy using (3.18)
  end for
  compute reward to-go, $\hat{R}_t = \sum_{t'=t}^{T} r(s_{t'}, a_{t'}, s_{t'+1})$
  compute advantage estimate for the trajectory, $\hat{A}_t$ using υ_ϕ^π
end for
compute gradient using (3.16) and update the policy π_θ via stochastic gradient ascent using Adam [46]
find value function parameters by minimizing the following error, $\sum_{k=0}^{N} \sum_{t=0}^{T} (\upsilon_\phi(s_t) - \hat{R}_t)^2$
update ϕ and θ

In the simulated environment, we discussed that the actions for loop closure are sparsely populated. The loop closure scenario occurs when the agent has returned to a position it has visited earlier. However, in the simulated environment there is a large number of steps taken before a correct loop closure action has occurred. In our case, the agent has to return to a position after a certain period of time (this is to avoid frequent loop closures), and the location also has to be a pre-determined loop closure location. Figure 3.9 shows how the percentage of possible loop closure actions varies with pre-determined loop closure locations and the size of the grid. It shows that for different grid size environment from 20×20 to 80×80, and varying the pre-determined loop closure locations from 10% to 25% of the total

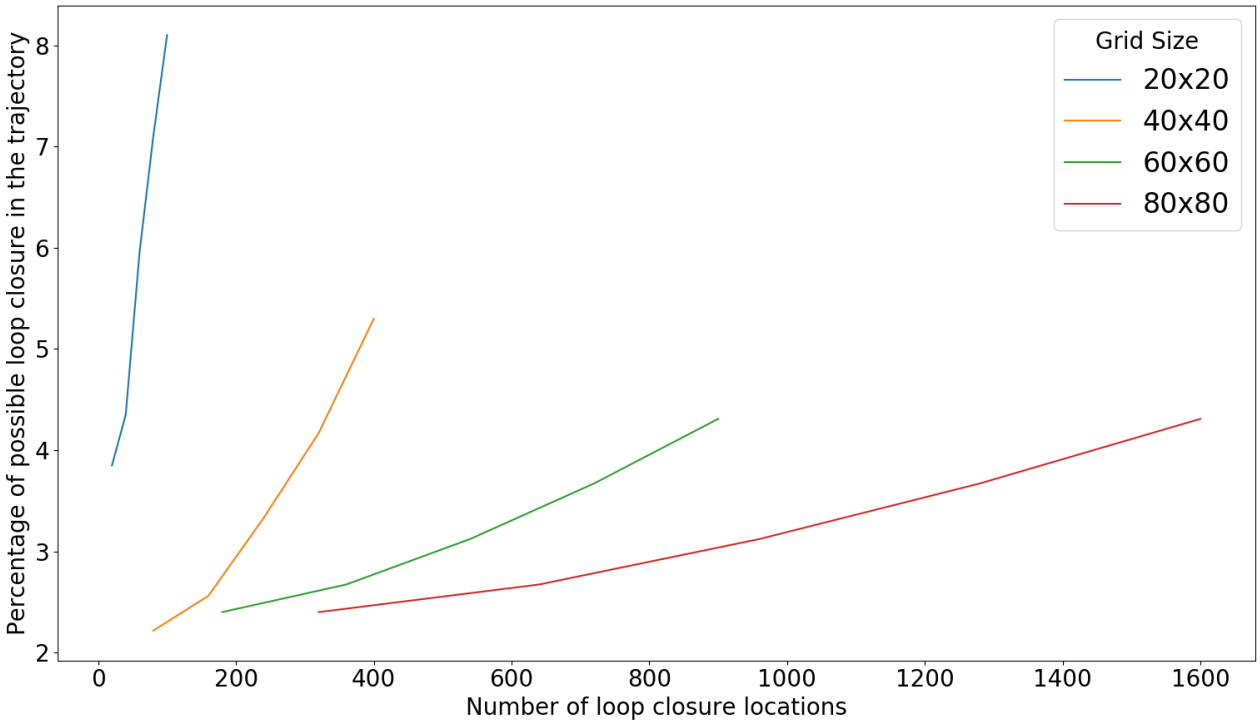


Figure 3.9: The percentage of possible loop closures in the trajectory of different grid size environment. It shows how the percentage of loop closures changes as we increase grid size and number of loop closure locations increase.

size of the environment. The percentage of loop closures to the total steps taken for the same trajectory remains below 10%. So it is evident from the figure that as the size of E increases, there is less chance of observing loop closure action in the trajectory.

In scenarios where most of the actions are no loop closure, if we use a fixed-length trajectory we run risk of missing loop closure actions from the sampled trajectory. In this case, our policy will only learn to perform a no loop closure action correctly and may never learn any loop closure detection. To solve this, we use an entropy optimization on the loop closure action distribution to make sure the loop closure actions are included in the trajectory of the episodes used for training.

We use a variable-length trajectory for each actor. If the previous entropy of actions of a sampled trajectory is $H(Tr_1)$ and the current entropy is $H(Tr_2)$, then we optimize based

on the size of the sample as

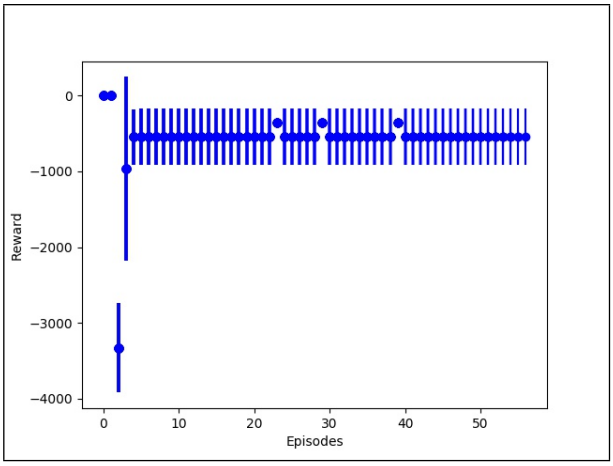
$$\begin{aligned} \underset{n}{\text{maximize}} \quad & H(Tr_1(n)) - H(Tr_2(n)) \\ \text{subject to} \quad & b_1 < n < b_2 \end{aligned} \tag{3.20}$$

where, b_1 and b_1 are the minimum and maximum batch size for a trajectory. b_2 is determined based on the hardware the training is running. This way, we capture trajectories of the agent with loop closure actions while maintaining the sample batch size in between a range.

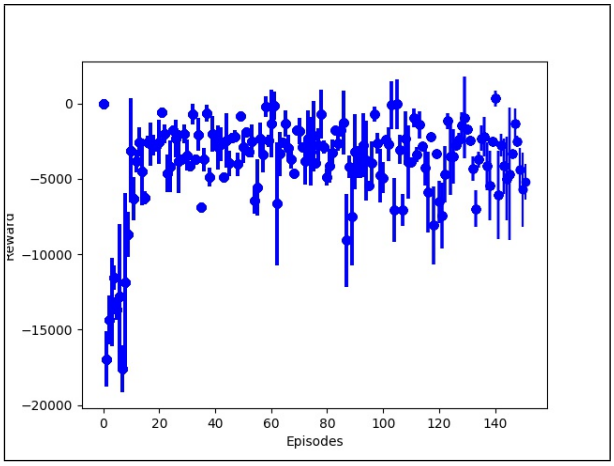
3.7 Experiments and Results

We tested the algorithm with 6 different simulated environments shown in Figure 3.5. Since the loop closure positions need to be fixed in our process, we choose several positions in each of the environments before the training started. We will describe the results using two evaluation processes. First, we showcase the learning or reward curve for the algorithm. We consider the average reward per episode in the training as the score metric. This result is demonstrated as a reward score against all the episodes involved in the training. Second, after a training has converged for an environment we evaluate it again to obtain the accuracy of loop closure actions in the environment.

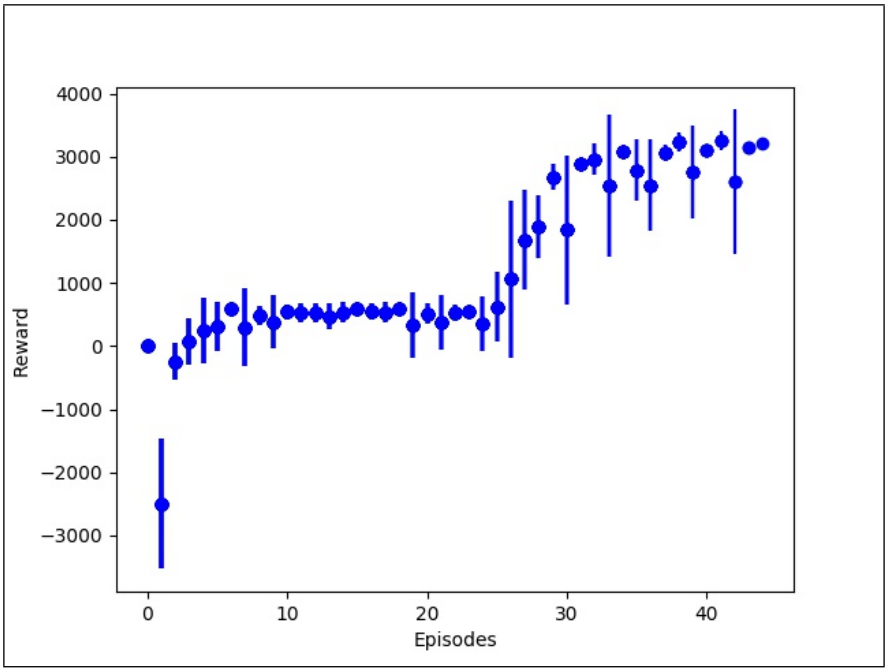
First, we performed analysis on the entropy of the samples and how it affects the training of a policy in DRL. Figure 3.10a and 3.10b shows a reward curve for a trained policy without any entropy maximization of the samples. In Figure 3.10a, the reward for the agent never turns positive, and it converges to a value that indicates that it only learn to perform no loop closure action. We increased the batch size so entropy will be increased as more actions are included in the trajectory. The reward curve for this is shown in Figure 3.10b, which is evidence of the policy again failing to achieve satisfactory reward and convergence over time. Finally we demonstrate again this same setup with entropy maximization in Figure 3.10c. The reward curve does achieve positive reward where the previous two methods failed.



(a) Batch Size 512



(b) Batch Size 2048



(c) Entropy Dependent Batch Size

Figure 3.10: Reward curves with different sample sizes: The top left image is with a small sample size of of 512 and the top right is with a larger sample size 2048. The bottom image with sample size determined by the entropy maximization step.

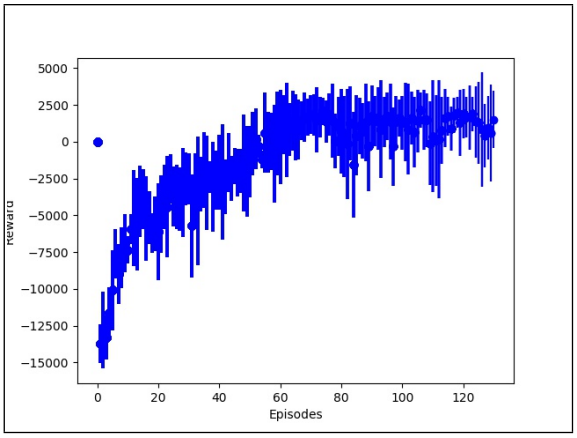
These reward curve results shows the effect of sample efficiency for RL. By using entropy, we are able to find samples that can help the RL algorithm to learn faster.

We demonstrate again the rewards curve for all six environment. Figure 3.11 shows the reward curve in the training on the six environment. Each time, reward starts negative as the parameters are random, but over time the policy is able to learn and achieve positive reward. It can seen from the reward curve that as more loop closure locations are added in the environment the longer period it takes for the algorithm to train. Sequence 6 showed a drop in the reward after reaching a convergence. Since the environment starts every new episode from a random position, the agent may have explored new observations that it was not trained on, resulting in incorrect actions and negative reward. It was able learn based on those actions and quickly reach convergence again. It is evident from all the rewards that the agent has learned to perform loop closure as required by the environment, and policy has reached convergence.

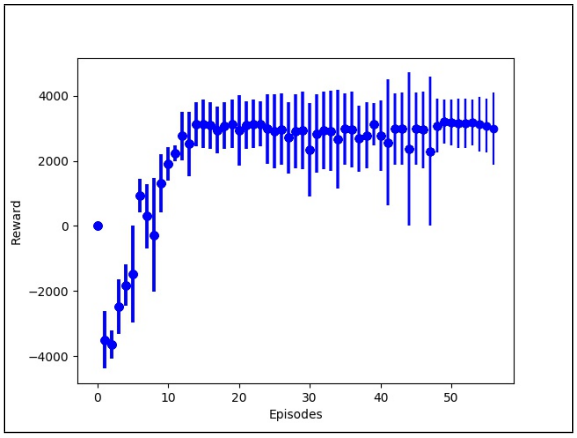
The Table 3.1 summarizes the results on the six simulated environment from a trained policy. The accuracy metric is based on the following formula,

$$Accuracy = \frac{\text{Number of correct loop closure action in trajectory}}{\text{Number of loop closure actions in the trajectory}}.$$

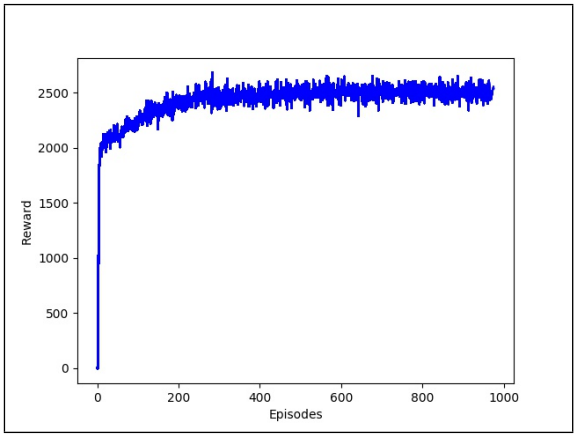
The policy is trained on each of the environments and after the policy has converged, we run the trained policy on each of the environment again to gather the loop closure accuracy results. In this step, we start the agent at a different point in the environment and run the simulation until all the loop closure locations has been tested. The movement of the agent is random in the environment, and the test environment does not terminate as it would in the training period. This step is completed for 20 times for each of the environments to obtain the loop closure results. On average, we are able to perform loop closure with 80% accuracy in all the environments. The cases of missed loop closure is due to observations that do not contain enough information to perform loop closure action. Also, the policy provides



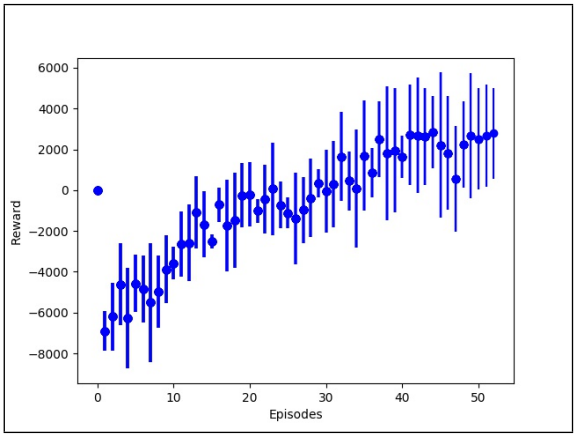
(a) Sequence 1



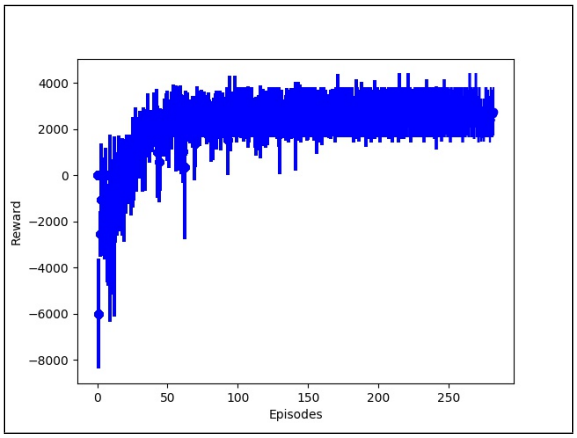
(b) Sequence 2



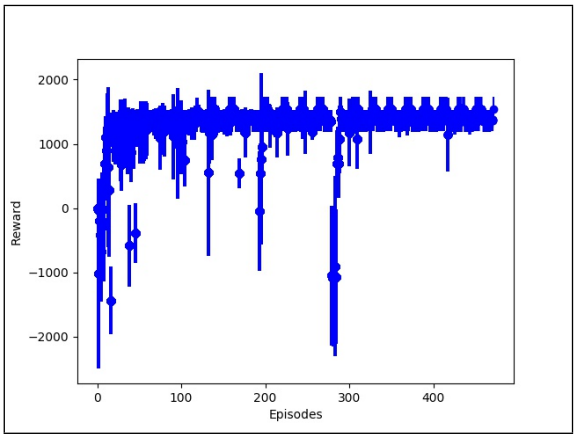
(c) Sequence 3



(d) Sequence 4



(e) Sequence 5



(f) Sequence 6

Figure 3.11: This figure shows the reward curves for the six environments we trained our RL policy.

Table 3.1: Loop closure detection statistics in the simulated environment

Sequence	Number of loop closure locations	Precision and Recall of loop closure detection
sequence 1{10×10}	2	77.8/63.5
sequence 2{10×10}	4	78.2/71.2
sequence 3{10×10}	5	82.5/63.1
sequence 4{10×10}	8	85.6/68.7
sequence 5{15×15}	14	76.4/64.4
sequence 6{84×84}	20	83.1/70.6

a probability of loop closure actions that has to be more than threshold to be considered a loop closure. This is on par with existing loop closure methods based on probabilistic frameworks. The accuracy of most loop closure methods discussed in Section 1.3.1 lie in the range 80% to 90%. Angeli et al [2] demonstrated correct loop closure accuracy 80% using visual features and histogram information. FabMap [20] showed a loop closure precision of 85% to 90% with recall range from 65% to 50%.

3.7.1 Comparisons

We compare results of our loop closure detection algorithm with other existing popular algorithms. Angeli et al. [2] demonstrated loop closure with a bag-of-words method and reported loop closure results as "%TP", i.e. the percentage of loop closures correctly detected. Table 3.2 shows the statistics on their own captured sequence at indoor and outdoor environments. "SIFT + H" indicates SIFT features are used in combination with histograms.

Cummins et al. [20] proposed FAB-MAP and showed loop closure detection on two datasets with precision and recall statistics. The two datasets are City Centre and New College Dataset. Both datasets are based on outdoor environments and were captured from a robot driving few kilometers. Figure 3.12 shows the precision and recall curve on these datasets. They are able to achieve 80% precision with 52% recall on City Centre and 87%

Table 3.2: Loop closure detection result statistics of proposed method by Angeli et al. [2]

Sequence	%TP
Indoor SIFT + H	80
Outdoor SIFT + H	71

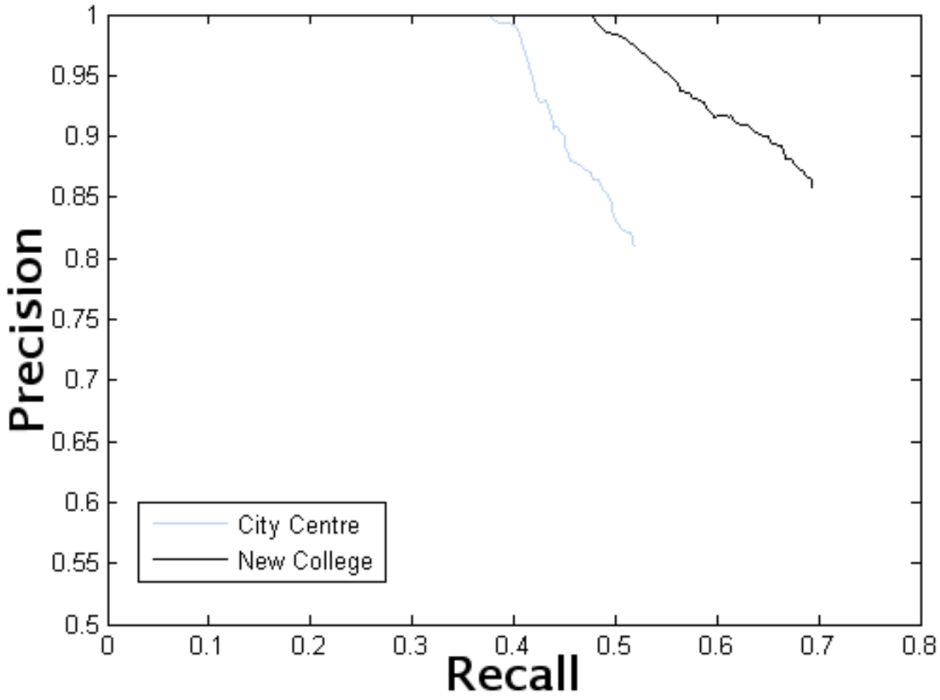


Figure 3.12: FAB-MAP [20] Precision-Recall curves for the City Centre and New College datasets

precision with 68% recall on New College. Increasing the precision any higher decreases the recall. 100% precision rate is demonstrated with a low recall rate of less than 50%.

In comparison, our proposed method on average demonstrated 80% precision with 67% recall on our simulated environment. This shows our proposed method is on par with the existing loop closure detection algorithms. Although our experiments are completed on simulated environments and the comparisons mentioned here are tested on real environment, our method can be extended to real world environments by combining our classified object

localization process. We discuss the process of translating real world environments in our simulated grid world in the next section.

3.8 Simulated Environment to Real World Transformation

So far in this chapter, we have demonstrated a simulated environment for loop closure scenarios. In the previous chapter, we demonstrated a process for the localization of classified objects in the environment. We now demonstrate how these two methods can be combined to transform the simulated environment to real world application.

Recently, semi-autonomous driving has been demonstrated using simulated environments. There exist several simulation software packages that mimic real-world environments using advanced graphics. Alexey et al. [25] introduced CARLA to support the development, training, and validation of level 3 to 4 autonomous driving. This simulation generates data close to what a driver would have seen while driving. Zoox, a leading autonomous driving start-up company, demonstrated their toolbox where sensor data is captured from the real world and then these data get translated into their simulated environment. This simulated environment is partially built beforehand, and others are generated from real-time data. In this simulated environment, the autonomous actions are taken based on the translated data. Figure 3.13 shows such cases where data from the real world is used to generate a simulated environment.

We intend to generate a similar scenario. A real-world dataset is first translated to a simulated grid environment. Then, we train our RL-agent on it to learn to perform loop closure. Later, once we have a trained policy, we can translate the real world data to the simulated environment and perform loop closure using the trained policy. Figure 3.14a shows our results using the object localization process on one of the TUM dataset, and Figure 3.14b shows the same map built using our simulated environment with 40×40 grid size.

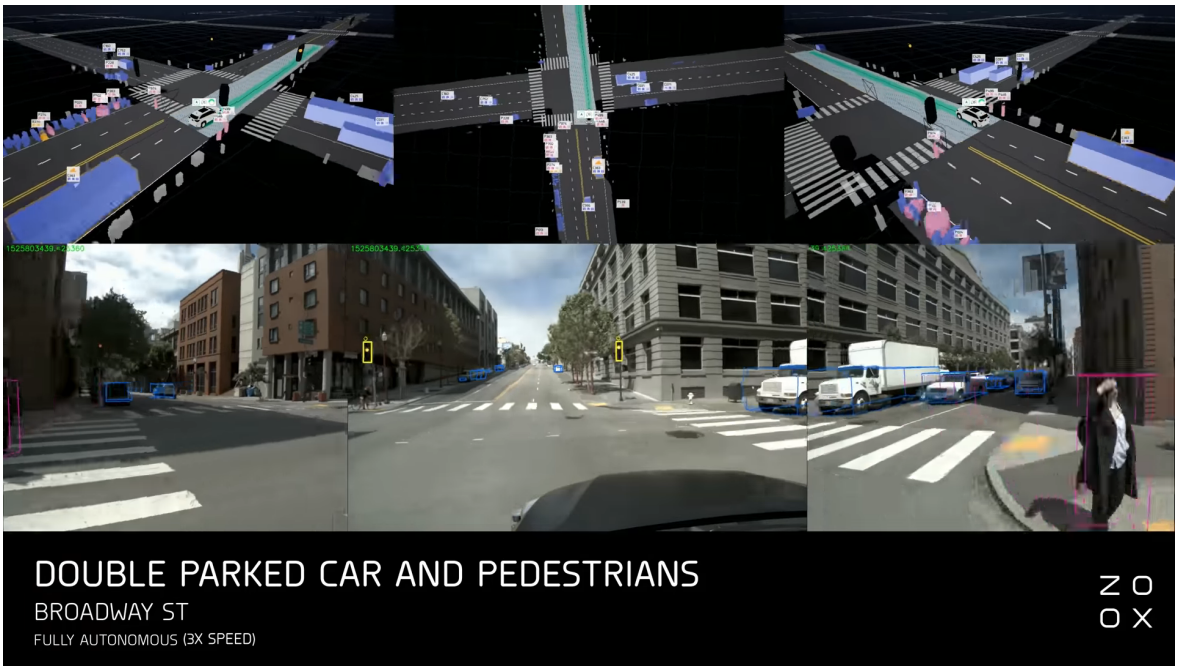
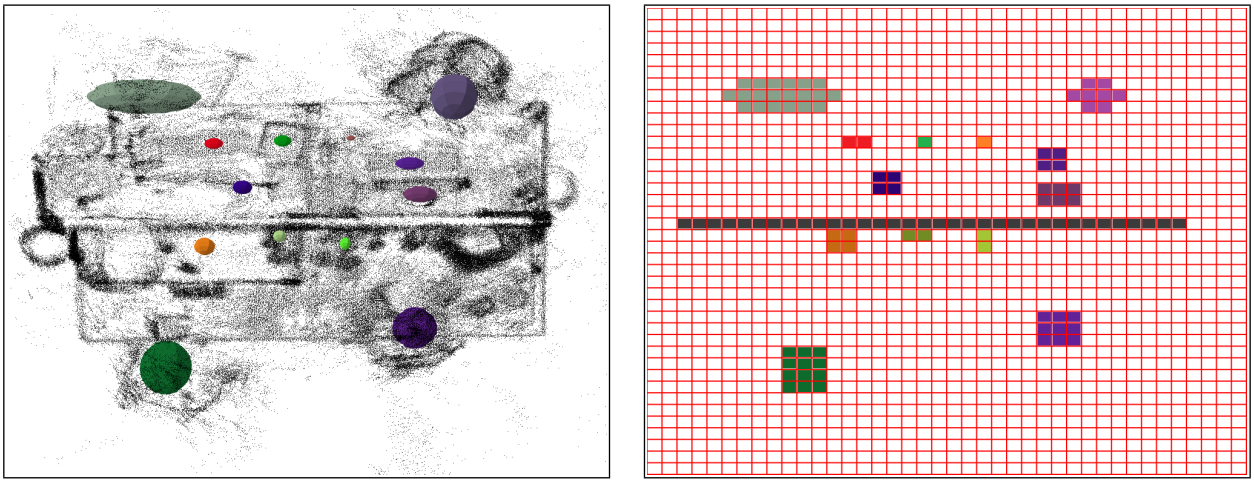


Figure 3.13: This figure shows a demonstration from the Zoox autonmous driving platform. It shows the data from a real world environment translated to a prebuilt map, and current detected object used to update the map.



(a) Results from Chapter 2 on a TUM sequence

(b) Simulated grid on the map of TUM sequence

Figure 3.14: A result of our object localization result on left image. The locations of the localized objects are translated to the simulated grid environment on the right image. Each object is represented with a different color block.

3.9 Conclusion

We presented a solution to the loop closure problem in SLAM using deep reinforcement learning. We showed the formulation of loop closure can be accommodated with certain assumptions. A simulated environment was demonstrated to train loop closure with features and walls. We also showed how this simulated environment can be designed similarly to an indoor environment. We provided a detailed procedure for training the simulated environment. An entropy issue was encountered with action distribution in the sampling of trajectory, and we provided a solution by maximizing entropy of action by varying the sampling of the trajectory. We demonstrated our results on several simulated environments with reward curve and statistics about the success rate of LC. The results indicate that we were successful in detecting loop closure locations in varied simulated environments. Our method demonstrates a novel solution to loop closure that does not require image storage, rather it requires a graphical model of the environment.

CHAPTER 4

SUMMARY AND CONCLUSION

Localization and mapping of an environment is an important autonomous robotics task. Traditional mapping processes include only sensor data in the accrued map. Additional information about the environment can be obtained through further processing, such as object detection and classification. The general robotics mapping and localization process, SLAM does not incorporate these information in the map. In this dissertation, we demonstrated a process to include classified objects in the mapping and localization process. The existence of the objects are obtained using a pre-trained CNN module. One of the major challenge in this process is data association. We provide a solution to this using a non-parametric statistical method, the Mann Whitney U-test. This process requires the corresponding depth data of the classified objects. We also added a depth estimation process to handle large motion in the association process. Later, these classified and associated objects are localized in the map using an unsupervised clustering process. Experiments were completed on public and our own dataset to demonstrate results. We demonstrated that the classified objects can be localized within the map with 50% IoU with the ground truth location. The metric showed our strong accuracy in localizing existing objects in the map. Our results and process showed how traditional map building process can leverage the object classification module to fuse semantic information. These additional information can help extend a robot to explore environment with much more detailed information and help perform other task such as finding a class of object in a unknown environment. Currently, the location of the classified object is added to the map. In the future, the extracted information about the classified objects can be incorporated in the SLAM graph model as a node with constraints.

Another problem we discussed in this dissertation is loop closure. We discussed how loop closure is generally performed using visual features and proposed a different solution to loop closure through reinforcement learning. We demonstrated our solution using a simulated

environment. The simulated environment is a grid-based, two-dimensional representation of a map. It includes elements such as feature blocks and walls to maintain a similarity to an indoor environment. The environment can be generated in various sizes with different combination of features and walls. In this simulation, the RL agent received rewards for correctly detecting loop closure and optimizing over these accumulated rewards in the environment the agent learned to perform loop closure at different locations. We also provided a solution to inefficient sampling of data in training. Sampling data with a fixed length can miss the actions. We introduced entropy maximization to sample data at the desired actions intended for training. Our tests on several different simulations have shown the agent has learned to perform loop closure at the required locations. Our training method can achieve convergence with different simulated environments and perform loop closure with 80% accuracy on all the test environments. Our future goal is to extend the current work on loop closure to real world environment. Both contributions proposed in this dissertation can be combined into a single process to perform object localization and then use localization information to help loop closure detection.

BIBLIOGRAPHY

[1] Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

[2] Angeli, A., D. Filliat, S. Doncieux, and J. Meyer (2008, Oct). Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics 24*(5), 1027–1037.

[3] Arandjelovic, R., P. Gronat, A. Torii, T. Pajdla, and J. Sivic (2016). Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5297–5307.

[4] Ataer-Cansizoglu, E. and Y. Taguchi (2016). Object detection and tracking in rgb-d slam via hierarchical feature grouping. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4164–4171. IEEE.

[5] Bailey, T. and H. Durrant-Whyte (2006). Simultaneous localisation and mapping (slam) part 2: State of the art. *Robotics and Automation Magazine*.

[6] Bay, H., T. Tuytelaars, and L. Van Gool (2006). Surf: Speeded up robust features. In *European conference on computer vision*, pp. 404–417. Springer.

[7] Berkhin, P. et al. A survey of clustering data mining techniques. *Grouping multidimensional data 25*, 71.

[8] Bhattacharyya, A. (1943). On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 99–109.

[9] Bowman, S. L., N. Atanasov, K. Daniilidis, and G. J. Pappas (2017, May). Probabilistic data association for semantic slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1722–1729.

[10] Cadena, C., L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard (2016, Dec). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics 32*(6), 1309–1332.

[11] Castle, R. O., D. J. Gawley, G. Klein, and D. W. Murray (2007). Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Proc. International Conference on Robotics and Automation*, pp. 4102–4107.

[12] Castle, R. O., G. Klein, and D. W. Murray (2010). Combining monoslam with object recognition for scene augmentation using a wearable camera. *Image and Vision Computing 28*(11), 1548–1556.

[13] Chen, Z., A. Jacobson, N. Sünderhauf, B. Upcroft, L. Liu, C. Shen, I. Reid, and M. Milford (2017). Deep learning features at scale for visual place recognition. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3223–3230. IEEE.

[14] Choudhary, S., A. J. Trevor, H. I. Christensen, and F. Dellaert (2014). Slam with object discovery, modeling and mapping. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1018–1025. IEEE.

[15] Civera, J., D. D. Gálvez-López, L. Riazuelo, J. D. Tardós, and J. Montiel (2011, Sept). Towards semantic slam using a monocular camera. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1277–1284.

[16] Clemente, L. A., A. J. Davison, I. D. Reid, J. Neira, and J. D. Tardós. Mapping large loops with a single hand-held camera.

[17] Collobert, R., S. Bengio, and J. Mariéthoz (2002). Torch: a modular machine learning software library. Technical report, Idiap.

[18] Csurka, G., C. Dance, L. Fan, J. Willamowski, and C. Bray (2004). Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, Volume 1, pp. 1–2. Prague.

[19] Cummins, M. and P. Newman (2007). Probabilistic appearance based navigation and loop closing. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 2042–2048. IEEE.

[20] Cummins, M. and P. Newman (2008). Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research 27*(6), 647–665.

[21] Dahl, G. E., D. Yu, L. Deng, and A. Acero (2011). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing 20*(1), 30–42.

[22] Dalal, N. and B. Triggs (2005). Histograms of oriented gradients for human detection.

[23] Degris, T., M. White, and R. S. Sutton (2012). Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*.

[24] Dellaert, F. (2012). Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology.

[25] Dosovitskiy, A., G. Ros, F. Codevilla, A. Lopez, and V. Koltun (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16.

[26] Durrant-Whyte, H. and T. Bailey (2006, June). Simultaneous localization and mapping: part I. *IEEE Robotics Automation Magazine 13*(2), 99–110.

[27] Ekvall, S., P. Jensfelt, and D. Kragic (2006, Oct). Integrating active mobile robot object recognition and slam in natural environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5792–5797.

[28] Engel, J., T. Schöps, and D. Cremers (2014). Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pp. 834–849. Springer.

[29] Ester, M., H.-P. Kriegel, J. Sander, X. Xu, et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, Volume 96, pp. 226–231.

[30] Finn, C., P. Abbeel, and S. Levine (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org.

[31] Frintrop, S. and P. Jensfelt (2008, Oct). Attentional landmarks and active gaze control for visual slam. *IEEE Transactions on Robotics 24*(5), 1054–1065.

[32] Gao, X. and T. Zhang (2017). Unsupervised learning to detect loops using deep neural networks for visual slam system. *Autonomous robots 41*(1), 1–18.

[33] Girshick, R., J. Donahue, T. Darrell, and J. Malik (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587.

[34] Glocker, B., S. Izadi, J. Shotton, and A. Criminisi (2013). Real-time rgb-d camera relocalization. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, pp. 173–179. IEEE.

[35] Goncalves, L., E. di Bernardo, D. Benson, M. Svedman, J. Ostrowski, N. Karlsson, and P. Pirjanian (2005, April). A visual front-end for simultaneous localization and mapping. In *IEEE International Conference on Robotics and Automation*, pp. 44–49.

[36] Grinvald, M., F. Furrer, T. Novkovic, J. J. Chung, C. Cadena, R. Siegwart, and J. Nieto (2019, July). Volumetric instance-aware semantic mapping and 3d object discovery. *IEEE Robotics and Automation Letters 4*(3), 3037–3044.

[37] Hausknecht, M. and P. Stone (2015). Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

[38] He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

[39] Higgins, J. J. (2003). Introduction to modern nonparametric statistics.

[40] Ho, K. L. and P. Newman (2006). Loop closure detection in slam by combining visual and spatial appearance. *Robotics and Autonomous Systems 54*(9), 740–749.

[41] Holz, D., A. E. Ichim, F. Tombari, R. B. Rusu, and S. Behnke (2015, Dec). Registration with the point cloud library: A modular framework for aligning in 3-d. *IEEE Robotics Automation Magazine 22*(4), 110–124.

[42] Howard, A., L. E. Parker, and G. S. Sukhatme (2006). The sdr experience: Experiments with a large-scale heterogeneous mobile robot team. In *Experimental Robotics IX*, pp. 121–130. Springer.

[43] Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[44] Huynh, D. Q. (2009). Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision 35*(2), 155–164.

[45] Jia, Y., E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678. ACM.

[46] Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[47] Košecká, J., F. Li, and X. Yang (2005). Global localization and relative positioning based on scale-invariant keypoints. *Robotics and Autonomous Systems 52*(1), 27–38.

[48] Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

[49] Kröse, B. J., N. Vlassis, R. Bunschoten, and Y. Motomura (2001). A probabilistic model for appearance-based robot localization. *Image and Vision Computing 19*(6), 381–391.

[50] Kümmerle, R., G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard (2011). g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613. IEEE.

[51] Lai, K., L. Bo, and D. Fox (2014). Unsupervised feature learning for 3d scene labeling. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3050–3057. IEEE.

[52] Lai, K., L. Bo, X. Ren, and D. Fox (2011). A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1817–1824. IEEE.

[53] Lee, K., S.-A. Kim, J. Choi, and S.-W. Lee (2018). Deep reinforcement learning in continuous action spaces: a case study in the game of simulated curling. In *International Conference on Machine Learning*, pp. 2943–2952.

[54] Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research 17*(1), 1334–1373.

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

[56] Long, J., E. Shelhamer, and T. Darrell (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

[57] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*(2), 91–110.

[58] Mann, H. B. and D. R. Whitney (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 50–60.

[59] McCormac, J., A. Handa, A. Davison, and S. Leutenegger (2017). Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *IEEE International Conference on Robotics and Automation*, pp. 4628–4635. IEEE.

[60] McInnes, L. and J. Healy (2017). Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 33–42. IEEE.

[61] McInnes, L., J. Healy, and S. Astels (2017). hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software 2*(11), 205.

[62] Mirowski, P., R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. (2016). Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

[63] Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937.

[64] Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[65] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). Human-level control through deep reinforcement learning. *Nature 518*(7540), 529.

[66] Mu, B., S. Y. Liu, L. Paull, J. Leonard, and J. P. How (2016a, Oct). Slam with objects using a nonparametric pose graph. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4602–4609.

[67] Mu, B., S.-Y. Liu, L. Paull, J. Leonard, and J. P. How (2016b). Slam with objects using a nonparametric pose graph. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4602–4609. IEEE.

[68] Munoz-Salinas, R., M. J. Marin-Jimenez, E. Yeguas-Bolivar, and R. Medina-Carnicer (2018). Mapping and localization from planar markers. *Pattern Recognition 73*, 158–171.

[69] Munoz-Salinas, R. and R. Medina-Carnicer (2019). Ucoslam: Simultaneous localization and mapping by fusion of keypoints and squared planar markers. *arXiv preprint arXiv:1902.03729*.

[70] Mur-Artal, R. and J. D. Tardós (2015). Probabilistic semi-dense mapping from highly accurate feature-based monocular slam. In *Robotics: Science and Systems*.

[71] Mur-Artal, R. and J. D. Tardós (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*.

[72] Nanni, L., S. Ghidoni, and S. Brahnam (2017). Handcrafted vs. non-handcrafted features for computer vision classification. *Pattern Recognition 71*, 158–172.

[73] Newman, P., D. Cole, and K. Ho (2006). Outdoor slam using visual appearance and laser ranging. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1180–1187. IEEE.

[74] Nicholson, L., M. Milford, and N. Sünderhauf (2018). Quadricslam: Dual quadrics from object detections as landmarks in object-oriented slam. *IEEE Robotics and Automation Letters 4*(1), 1–8.

[75] Pillai, S. and J. Leonard (2015). Monocular slam supported object recognition. *arXiv preprint arXiv:1506.01732*.

[76] Ramos, F., B. Upcroft, S. Kumar, and H. Durrant-Whyte (2012). A bayesian approach for place recognition. *Robotics and Autonomous Systems 60*(4), 487–497.

[77] Rawat, W. and Z. Wang (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation 29*(9), 2352–2449.

[78] Ren, S., K. He, R. Girshick, and J. Sun (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99.

[79] Rublee, E., V. Rabaud, K. Konolige, and G. R. Bradski (2011). Orb: An efficient alternative to sift or surf. In *ICCV*, Volume 11, pp. 2. Citeseer.

[80] Salas-Moreno, R. F., R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison (2013, June). Slam++: Simultaneous localisation and mapping at the level of objects. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1352–1359.

[81] Schaul, T., D. Horgan, K. Gregor, and D. Silver (2015). Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320.

[82] Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015). Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897.

[83] Schulman, J., P. Moritz, S. Levine, M. Jordan, and P. Abbeel (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[84] Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[85] Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature 529*(7587), 484.

[86] Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller (2014). Deterministic policy gradient algorithms.

[87] Simonyan, K. and A. Zisserman (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[88] Sturm, J., N. Engelhard, F. Endres, W. Burgard, and D. Cremers (2012, Oct). A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580.

[89] Sünderhauf, N., T. T. Pham, Y. Latif, M. Milford, and I. Reid (2017). Meaningful maps with object-oriented semantic mapping. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5079–5085. IEEE.

[90] Sünderhauf, N., S. Shirazi, A. Jacobson, F. Dayoub, E. Pepperell, B. Upcroft, and M. Milford (2015). Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. *Proceedings of Robotics: Science and Systems XII*.

[91] Sutton, R. *Introduction to reinforcement learning*, Volume 2.

[92] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning 3*(1), 9–44.

[93] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

[94] Szita, I. and A. Lörincz (2006). Learning tetris using the noisy cross-entropy method. *Neural computation 18*(12), 2936–2941.

[95] Thrun, S. et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium 1*(1-35), 1.

[96] Uijlings, J. R., K. E. Van De Sande, T. Gevers, and A. W. Smeulders (2013). Selective search for object recognition. *International journal of computer vision 104*(2), 154–171.

[97] Ulrich, I. and I. Nourbakhsh (2000). Appearance-based place recognition for topological localization. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, Volume 2, pp. 1023–1029. Ieee.

[98] Van Hasselt, H., A. Guez, and D. Silver (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

[99] Viola, P., M. Jones, et al. Rapid object detection using a boosted cascade of simple features.

[100] Wang, J., H. Zha, and R. Cipolla (2006). Coarse-to-fine vision-based localization by indexing scale-invariant features. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 36*(2), 413–422.

[101] Williams, B., P. Smith, and I. Reid (2007). Automatic relocalisation for a single-camera simultaneous localisation and mapping system. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 2784–2790. IEEE.

[102] Xu, D. and Y. Tian (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science 2*(2), 165–193.

[103] Zender, H., O. M. Mozos, P. Jensfelt, G.-J. Kruijff, and W. Burgard (2008). Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems 56*(6), 493–502.

[104] Zhang, J. and S. Singh. Loam: Lidar odometry and mapping in real-time.

[105] Zhang, Y., R. Jin, and Z.-H. Zhou (2010). Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics 1*(1-4), 43–52.

BIOGRAPHICAL SKETCH

Asif Iqbal is a PhD candidate in the Electrical Engineering program and a research assistant in the SeRViCE Lab at The University of Texas at Dallas, TX, USA. He received his MS degree in Electrical Engineering from The University of Texas at Dallas. He received his BS degree in Electrical and Electronics Engineering from the Bangladesh University of Engineering and Technology in 2011. His research interests include robotics, reinforcement learning and computer vision.

CURRICULUM VITAE

Asif Iqbal

November, 2019

Contact Information:

Department of Electrical Engineering
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080-3021, U.S.A.

Voice: (972) 883-4724
Fax: (972) 883-2349
Email: axi122230@utdallas.edu

Educational History:

B.S., Electrical and Electronics Engineering,
Bangladesh University of Engineering and Technology, 2006-2011
M.S. and PhD, Electrical Engineering,
The University of Texas at Dallas, 2013-2019

Classified Object Localization in SLAM and Loop Closure through Reinforcement Learning
PhD Dissertation
Electrical Engineering, The University of Texas at Dallas
Advisor: Dr. Nicholas R. Gans

Development of an automatic vehicle license plate detection and recognition system for Bangladesh
Undergraduate Thesis
Electrical and Electronics Engineering, Bangladesh University of Engineering and Technology
Advisor: Dr. M.S. Rahman

Employment History:

Software Engineer, Samsung Research Bangladesh - Dhaka, 2011 – 2012
Perceptual Computing Software Engineer Intern, Intel - Chandler, AZ, Jan 2016 – August 2016
Graduate Intern, Texas Instruments - Dallas, TX, May 2018 – August 2018

Graduate Coursework:

Digital Signal Processing I and II
Linear Systems
Pattern Recognition
Speech and Speaker Recognition
Convex Optimization
Digital Image Processing
Random Processes
Video Analytics
Multimodal Signal Processing
Speech and Speaker Recognition
Dynamics of Complex Network and Systems

Publications:

A. Iqbal, N. Gans, 'Data Association and Localization of Classified Objects in Visual SLAM', In review for *Journal of Intelligent & Robotic Systems*

A. Iqbal, N. Gans, 'Loop Closure through Reinforcement Learning', Journal in Preparation

A. Iqbal, N. Gans, 'Localization of Classified Objects in SLAM using Nonparametric Statistics and Clustering', in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

P Agrawal, A Iqbal, B Russell, MK Hazrati, V Kashyap, F Akhbari, 'PCE-SLAM: A Real-time Simultaneous Localization and Mapping using LIDAR data', *In IEEE Intelligent Vehicles Symposium (IV)*, 2017 (pp. 1752-1757),

A. Iqbal, C. Busso, N. Gans, 'Adjacent Vehicle Collision Warning System using Image Sensor and Inertial Measurement Unit', *In the Proceedings of the ACM International Conference on Multimodal Interaction*, 2015, pp. 291-298.

Y. Li, A. Iqbal and N. Gans, 'Multiple Lane Boundary Detection Using A Combination of Low-Level Image Features', *In IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, 2014 (pp. 1682-1687).

N.A. Siddique, A. Iqbal, F. Mahmud, M.S. Rahman, Development of an Automatic Vehicle License Plate Detection and Recognition System for Bangladesh, In *IEEE International Conference on Informatics, Electronics & Vision (ICIEV)*, 2012 (pp. 688-693)