

---

# Navigation of Autonomous Collaborative Robots for Visual Inspection

---

Thesis submitted by  
**Arindam Saha**

**Doctor of Philosophy (Engineering)**

Department of Information Technology  
Faculty Council of Engineering & Technology  
Jadavpur University

Kolkata, India

2024





# JADAVPUR UNIVERSITY

## KOLKATA 700032, INDIA

INDEX NO. 144/17/E

**1. Title of the Thesis:** Navigation of Autonomous Collaborative Robots for Visual Inspection.

**2. Name, Designation & Institute of the Supervisor/s:**

**Bibhas Chandra Dhara**

Professor,

Department of Information Technology,

Jadavpur University

Kolkata

**3. List of publications:**

**(a) Journal Publications:**

- 1. Arindam Saha, Bibhas Chandra Dhara, Saiyed Umer, Kulakov Yurii, Jazem Mutared Alanazi and Ahmad Ali AlZubi, “Efficient Obstacle Detection and Tracking Using RGB-D Sensor Data in Dynamic Environments for Robotic Applications”, Sensors (2022). Vol. 22, No. 17, Article No. 6537, ISSN: 1424-8220. DOI = 10.3390/s22176537. URL: <https://www.mdpi.com/1424-8220/22/17/6537>.**
- 2. Arindam Saha, Bibhas Chandra Dhara, Saiyed Umer, Ahmad Ali AlZubi, Jazem Mutared Alanazi and Kulakov Yurii, “CORB2I-SLAM: An Adaptive Collaborative Visual-Inertial SLAM for Multiple Robots”, Electronics (2022). Vol. 11, No. 18, Article No. 2814, ISSN: 2079-9292. DOI = 10.3390/electronics11182814. URL: <https://www.mdpi.com/2079-9292/11/18/2814>.**
- 3. Arindam Saha and Bibhas Chandra Dhara, “3D LiDAR-based Obstacle Detection and Tracking for Autonomous Navigation in Dynamic Environments”, International Journal of Intelligent Robotics and Applications. Vol. 8, pp. 39-60, March 2024, DOI = 10.1007/s41315-023-00302-1. URL: <https://doi.org/10.1007/s41315-023-00302-1>.**

**(b) International Conference Publications:**

4. **Arindam Saha**, Lokesh Kumar, Sarvesh Sortee and **Bibhas Chandra Dhara**, “An Autonomous Aircraft Inspection System using Collaborative Unmanned Aerial Vehicles”, In Proceeding of IEEE Aerospace Conference (AERO), Big Sky, MT, USA, 2023, pp. 1-10, doi: 10.1109/AERO55745.2023.10115655.

**4. List of Patents:** None**5. List of Presentations in International Conferences:**

1. **Arindam Saha**, Lokesh Kumar, Sarvesh Sortee and **Bibhas Chandra Dhara**, “An Autonomous Aircraft Inspection System using Collaborative Unmanned Aerial Vehicles”, In Proceeding of IEEE Aerospace Conference (AERO), Big Sky, MT, USA, 2023, pp. 1-10, doi: 10.1109/AERO55745.2023.10115655.

## “Statement of Originality”

I, **Arindam Saha**(Index No. 144/17/E), registered on **03<sup>rd</sup> April 2017**, do hereby declare that this thesis entitled, “Navigation of Autonomous Collaborative Robots for Visual Inspection” contains literature survey and original research work is done by the undersigned candidate as part of Doctoral studies.

All information in this thesis have been obtained and presented in accordance with existing academic rules and ethical conduct. I declare that, as required by these rules and conduct, I have fully cited and referred all materials and results that are not original to this work.

I also declare that I have checked this thesis as per the “Policy on Anti Plagiarism, Jadavpur University, 2019”, and the level of similarity as checked by iThenticate software is 4%.

Signature of Candidate: *Arindam Saha*

Date: *07/03/2024*


*Bohan* *07.03.2024*  
Certified by Supervisor:  
(Signature with date, seal)

**PROFESSOR**  
**Deptt. of Information Technology**  
**JADAVPUR UNIVERSITY**  
**Block -LB, Plot-8, Sector-3**  
**Salt Lake, Kolkata-700106, India**



## CERTIFICATE FROM THE SUPERVISOR

This is to certify that the thesis entitled "*Navigation of Autonomous Collaborative Robots for Visual Inspection*" submitted by *Shri. Arindam Saha*, who got his name registered on April 03, 2017 for the award of Ph.D. (Engineering) degree of Jadavpur University is absolutely based upon his own work under the supervision of *Prof. Bibhas Chandra Dhara* and that neither his thesis nor any part of the thesis has been submitted for any degree or any other academic award anywhere before.

 07.03.2024

Signature of the Supervisor and Date with Official Seal

Bibhas Chandra Dhara

Professor

Department of Information Technology

Jadavpur University

Kolkata, India

**PROFESSOR**  
Deptt. of Information Technology  
**JADAVPUR UNIVERSITY**  
Block -LB, Plot-8, Sector-3  
Salt Lake, Kolkata-700106, India



# Abstract

This thesis addresses the challenge of environment perception in the field of autonomous mobile robot navigation, with a primary focus on two key aspects: dynamic obstacle estimation and collaborative localization using visual sensors.

An accurate perception with a rapid response is fundamental for any autonomous vehicle to navigate safely. In the field of dynamic obstacle estimation using visual sensors, we introduce an efficient method for obstacle estimation that utilizes only depth image of an RGB-D sensor to facilitate quick dynamic obstacle estimation. Here, we utilize another depth map representation, known as the u-depth map, for obstacle detection and propose a restricted v-depth map to improve the estimation of obstacle dimensions. Additionally, we present a novel tracking algorithm to monitor obstacles in subsequent frames until they are within the field of view.

In the realm of collaborative localization using visual sensors, we propose a client-server-based collaborative SLAM framework. Participating robot are equipped with a camera (monocular/stereo/RGB-D) and an inertial sensor for visual odometry, while a centralized server executes processor-intensive tasks such as loop closing, map merging, and global optimization. This framework employs Visual-Inertial Odometry and can adapt to use Visual Odometry in the presence of noisy inertial sensor measurements. It addresses certain disadvantages of odometry-based systems, such as erroneous pose estimation due to incorrect feature selection or losing camera track due to abrupt motion, providing a more accurate result.

Recognizing the limitations of RGB-D sensors in accurately estimating long-range obstacles, we investigate the utilization of Light Detection and Ranging (LiDAR) sensors, which offer precise environmental assessments in the form of 3D point clouds. Building upon the success of u-depth and restricted v-depth maps derived from our prior work with depth images, we extend these methodologies to LiDAR point clouds for long-range obstacle estimation. This approach obviates the necessity for certain computationally intensive modules, such as ground plane segmentation and 3D clustering, present in existing obstacle detection methods utilizing 3D LiDAR point clouds.

The evaluation of these three methods is conducted using multiple simulated data sets, self-captured data sets, open data sets, and real robot motion to establish their accuracy and effectiveness.

Furthermore, we propose a visual inspection system for pre-flight aircraft inspection using autonomous collaborative drones. This system is auto-adaptable to any aircraft model with minimal manual intervention. The drones initiate from random

locations in the vicinity of the aircraft, utilizing a novel registration algorithm to collaborate and navigate using LiDAR-Inertial measurements. The navigation algorithm establishes multilayered zones for safe navigation and employs our previously proposed obstacle estimation method using LiDAR to avoid static and dynamic obstacles. The system incorporates a low-cost RGB-D camera for inspection and defect detection. The proposed system is evaluated in a simulation with an anonymous aircraft model, demonstrating the ability to complete the inspection task within 10 minutes using two UAVs.



# Acknowledgements

Embarking on a Ph.D. journey has been one of the most challenging decisions in my life. It involves making scientific contributions to the domain, and accomplishing this without input, assistance, or encouragement from others is impractical. In many instances, I find it difficult to express my gratitude adequately due to the inherent challenges of pursuing a Ph.D.

I would like to extend my profound gratitude and indebtedness to my advisor, Prof. Bibhas Chandra Dhara, for his invaluable suggestions, encouragement, and support throughout this study. I first encountered him in my master's class on "Data Structures and Algorithms" and was influenced by his teaching style. My master's guide, Prof. Uttam Kumar Roy, recommended Prof. Bibhas Chandra Dhara to be my Ph.D. guide, and I am grateful to Prof. Uttam Kumar Roy for this recommendation. While this research work is an individual effort, it would have been impossible for me to complete this thesis without Prof. Dhara's guidance and erudite suggestions. His creative ideas and technical expertise have opened up new horizons in the realm of this research. It is not an exaggeration to express how exalted and proud I am to have him as a mentor. I am confident that his unwavering support and selfless advice will help me achieve my aspirations and bring the long-awaited success I seek.

I am also thankful to Dr. Saiyed Umer for his extensive collaboration on this project. His insightful interactions and meticulous attention to detail in writing the papers have significantly enhanced my presentation skills.

I express my heartfelt gratitude to two of my dearest childhood friends, Debabrata and Mithun, for their unwavering mental support during challenging times in my life. I appreciate their patience in enduring my discussions about work, even during our long-awaited vacations, and for providing their valuable opinions. Their kindness has truly proven to be my most reliable support system, and I am sincerely thankful for their enduring friendship.

I would also like to extend my gratitude to my friends and brothers, Sayan and Soumyadip, for their unwavering support and assistance throughout this lengthy journey, marked by numerous technical and non-technical discussions.

I would like to extend my heartfelt respect and gratitude to my parents and parents-in-law for their love, encouragement, and immense support in all aspects of my life. They have consistently granted me the freedom to pursue my desires, and I am genuinely thankful for their concern and patience throughout this extended period.

A special thanks go to my beloved wife, Suchismita, for her continuous support and motivation, both in positive and negative circumstances. Her unwavering encouragement has been a source of strength during this journey, and I anticipate similar support in our future endeavors.

Lastly, I express my gratitude to my daughter, Tannistha, for her unconditional love, curiosity about my work, and the support that has motivated me in various ways throughout this lengthy journey. May God bless her for all her future endeavors in life.



Arindam SAHA

September 28, 2024

# Contents

<b>Declaration of Authorship</b>	<b>v</b>
<b>Abstract</b>	<b>ix</b>
<b>Acknowledgements</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Preliminaries . . . . .	2
1.2 Autonomous Navigation of Robots . . . . .	2
1.2.1 Obstacle Estimation and Path Planning . . . . .	3
1.2.2 Visual Odometry . . . . .	6
1.2.3 Visual SLAM . . . . .	7
1.2.4 Visual-Inertial SLAM . . . . .	10
1.2.5 Collaborative VSLAM . . . . .	10
1.3 Objectives of the Thesis . . . . .	12
1.4 Contributions of the thesis . . . . .	13
1.5 Organization of the thesis . . . . .	14
<b>2 Literature Survey</b>	<b>17</b>
2.1 Obstacle Detection . . . . .	17
2.1.1 Stereo Images . . . . .	17
2.1.2 RGB-D Images . . . . .	18
2.2 VO and VSLAM . . . . .	19
2.2.1 Feature-based Methods . . . . .	20
2.2.2 Direct Methods . . . . .	21
2.3 VIO and VI-SLAM . . . . .	23
2.4 Collaborative VSLAMs and VI-SLAMs . . . . .	25
2.4.1 Distributed Collaborative Methods . . . . .	25
2.4.2 Centralized Collaborative Methods . . . . .	25
<b>3 Obstacle Detection and Tracking Using Depth Image</b>	<b>29</b>
3.1 Introduction . . . . .	29

3.2	System Architecture . . . . .	30
3.2.1	Self Localization . . . . .	30
3.2.2	Obstacle Detection . . . . .	31
3.2.2.1	Depth Map Processing . . . . .	31
3.2.2.2	Dimension Estimation . . . . .	34
3.2.3	Obstacle Tracking . . . . .	40
3.3	Experimental Results . . . . .	44
3.3.1	Datasets and Experimental Description . . . . .	44
3.3.2	Parameter Tuning . . . . .	46
3.3.3	Experiment 1: Tracking Single Dynamic Obstacle and Comparison with Ground Truth . . . . .	47
3.3.3.1	Gazebo Environment and Experimental Set-up . . . . .	47
3.3.3.2	Initialization at Full Visibility of Husky2 . . . . .	48
3.3.3.3	Initialization at Partial Visibility of Husky2 . . . . .	51
3.3.3.4	Accuracy Comparison on simulated data . . . . .	53
3.3.3.5	Accuracy of the Proposed Method on Open datasets . . . . .	55
3.3.4	Experiment 2: Tracking Single Non-Rigid Dynamic Obstacle . . . . .	59
3.3.5	Experiment 3: Tracking Multiple Non-Rigid Dynamic Obstacles . . . . .	61
3.3.5.1	Dynamic Obstacles Moving in Different Directions . . . . .	61
3.3.5.2	Multiple Dynamic Obstacles with Different Heights . . . . .	61
3.3.6	Experiment 4: Advantages of Dynamic U-Depth Thresholding . . . . .	62
3.3.7	Experiment 5: Tracking a Fast-Moving Obstacle . . . . .	65
3.3.8	Execution Time . . . . .	66
3.4	Conclusions . . . . .	68
4	<b>An Adaptive Collaborative Visual-Inertial SLAM for Multi-Agent Localization</b> . . . . .	<b>69</b>
4.1	Introduction . . . . .	69
4.2	CORB2I-SLAM Framework . . . . .	70
4.2.1	Notations . . . . .	71
4.2.2	Client Robot . . . . .	72
4.2.2.1	Odometry . . . . .	72
4.2.3	Map Structure and Optimization . . . . .	78
4.2.4	Communication Manager (Server and Client) . . . . .	79
4.2.5	Server System . . . . .	79
4.2.5.1	Client Manager . . . . .	80
4.2.5.2	Storage Manager . . . . .	81
4.2.5.3	Map Fusion . . . . .	81

4.2.6	Communication Bandwidth . . . . .	82
4.3	Experimental Results . . . . .	83
4.3.1	Experiment 1: Evaluating Single-Agent Accuracy . . . . .	84
4.3.2	Experiment 2: Evaluating Map Fusion Accuracy for Homogeneous Camera . . . . .	85
4.3.3	Experiment 3: Evaluating Map Fusion Accuracy for Heterogeneous Camera . . . . .	87
4.3.4	Experiment 4: Evaluating Map Fusion in Real Autonomy . . . . .	89
4.3.5	Execution Time . . . . .	89
4.4	Conclusions . . . . .	89
<b>5</b>	<b>Obstacle Detection and Tracking using 3D LiDAR</b>	<b>91</b>
5.1	Introduction . . . . .	91
5.2	Literature Survey . . . . .	93
5.3	System Design . . . . .	95
5.3.1	Obstacle Detection and Estimation . . . . .	96
5.3.1.1	Virtual Camera Coordinate Representation . . . . .	96
5.3.1.2	U-depth Representation . . . . .	98
5.3.1.3	Restricted V-depth Representation . . . . .	102
5.3.1.4	3D Dimension Estimation . . . . .	105
5.3.1.5	3D Occupancy Map . . . . .	106
5.3.2	Obstacle Tracking . . . . .	108
5.3.3	Dynamic Obstacle Estimation . . . . .	109
5.4	Experimental Results . . . . .	110
5.4.1	Experiment 1: Evaluation on Simulated Dataset . . . . .	111
5.4.2	Experiment 2: Evaluation on the KITTI datasets . . . . .	114
5.4.2.1	Quantitative Evaluation . . . . .	114
5.4.3	Experiment 3: Evaluation with Real-Time Robot Motion . . . . .	118
5.4.4	Experiment 4: Evaluation on Aerial LiDAR Open Dataset . . . . .	122
5.4.5	Execution Time . . . . .	122
5.5	Conclusions . . . . .	125
<b>6</b>	<b>Autonomous Aircraft Inspection using Collaborative UAVs</b>	<b>127</b>
6.1	Introduction . . . . .	127
6.2	Background Study . . . . .	128
6.3	System Design . . . . .	129
6.3.1	Operational Flow . . . . .	131
6.3.2	Simulated Environment . . . . .	132

6.3.3	Template Generation . . . . .	134
6.3.4	Navigation . . . . .	136
6.3.4.1	Global Planning . . . . .	136
6.3.4.2	Local Planning . . . . .	137
6.3.5	Registration . . . . .	137
6.3.6	Object Detection . . . . .	144
6.3.7	Object Inspection . . . . .	147
6.3.8	Communication Module . . . . .	147
6.4	Experimental Results . . . . .	148
6.4.1	Navigation and Path Following . . . . .	148
6.4.2	Registration Accuracy . . . . .	149
6.4.3	Measurement Validation . . . . .	150
6.4.4	Execution Time . . . . .	151
6.5	Conclusions . . . . .	152
<b>7</b>	<b>Conclusions and Future Works</b>	<b>153</b>
7.1	Contributions . . . . .	153
7.2	Future Research Directions . . . . .	155
<b>A</b>	<b>3D Reconstruction from 2D Images</b>	<b>157</b>
A.1	Pinhole Camera Model . . . . .	157
A.2	Structure From Motion . . . . .	162
A.2.1	Feature Estimation and Matching . . . . .	162
A.2.2	Epipolar Geometry and Essential Matrix Estimation . . . . .	163
A.2.3	Fundamental Matrix Estimation . . . . .	165
A.2.4	Camera Pose Estimation . . . . .	168
A.2.5	Optimization Using Bundle Adjustment . . . . .	168
	<b>Bibliography</b>	<b>171</b>

# List of Figures

1.1	An example of obstacle detection: (a) An RGB image captured in a static environment, (b) Detected obstacles are highlighted with yellow bounding boxes. . . . .	3
1.2	Three non-consecutive snapshots from the PTB [7] dataset, where (a)–(c) display the ground truth of object tracking, specifically the toy bear, marked with red bounding boxes, in contrast, (d)–(f) illustrate the likely outcomes of obstacle detection, marked by boxes in red, yellow, and green. . . . .	4
1.3	A sample path to reach the destination for the previous example in Fig. 1.1. . . . .	5
1.4	A pictorial view of incremental camera pose estimation in VO. . . . .	7
1.5	ORB-SLAM2: A VSLAM Framework. . . . .	9
1.6	An IMU integrated version of ORB-SLAM2. . . . .	11
3.1	Block diagram of the proposed system for robust dynamic obstacle detection and tracking using RGB-D camera sensor data. . . . .	30
3.2	Illustration of obstacles in the u-depth map: (a) A sample RGB snapshot in Gazebo [95], (b) Corresponding depth image with lower intensity representing smaller depths, and (c) Corresponding u-depth map, where the white patch at a smaller row index indicates the closer obstacle. . . . .	33
3.3	Estimating obstacle width from u-depth map: (a) Snapshot from the OpenLORIS-Scene market data sequence [58], (b) Corresponding u-depth map, (c) Corresponding thresholded u-depth map after a closing operation. . . . .	34
3.4	A sample thresholded u-depth map with a segmented obstacle and the corresponding bounding box. . . . .	34
3.5	Height estimation of an obstacle using VbOAD [4]: (a) A snapshot captured with D435i along with the estimated height of the obstacle (the person), (b) Corresponding thresholded u-depth map. . . . .	35
3.6	Obstacle height calculation using the proposed restricted v-depth map: (a) The same snapshot from Fig. 3.3(a), (b) Corresponding v-depth map, (c) Corresponding thresholded v-depth map, (d) Corresponding proposed thresholded restricted v-depth map. . . . .	36

3.7	Estimation of the height of an obstacle using the proposed restricted v-depth map: (a) The same image used in Fig. 3.5 and the result of the proposed technique, (b) Result of the thresholded restricted v-depth map, (c) The magnified view of the thresholded restricted v-depth map shows the portion of the roof as a separate obstacle. . . . .	38
3.8	A sample magnified, thresholded restricted v-depth map with a segmented obstacle within a bounding box. . . . .	38
3.9	(a) An RGB snapshot with detected obstacles in the OpenLORIS-Scene market data [58], (b) Corresponding Rviz [99] visualization with annotated dimensions. . . . .	40
3.10	A sample pictorial view of the selected points in $\Phi_A$ for obstacle A: (a) Red dots represent the points present in $\Phi_A$ , (b) Vector formation using the points in $\Phi_A$ . . . . .	41
3.11	Snapshots of Gazebo [95] environments: (a–h) are time-sampled snapshots depicting the movements of Husky1 (white) and Husky2 (yellow). . . . .	49
3.12	Comparison of tracking a dynamic obstacle (Husky2), where all RGB-based algorithms (top six rows) are initialized when Husky2 becomes completely visible (first column). VbOAD and the proposed method are initialized when Husky2 is partially visible. . . . .	50
3.13	Comparison among the mentioned tracking algorithms with a dynamic obstacle (Husky2). All algorithms are initialized when Husky2 becomes partially visible (first column). . . . .	52
3.14	Comparative analysis of the deviation between the estimated and actual relative distance of Husky2 from Husky1 in the camera coordinate frame $C$ . . . . .	53
3.15	The proposed method consistently estimates Husky2's closest distance, which is always smaller than the relative distances at any given time. . . . .	54
3.16	Absolute positional estimation of Husky2 in the world coordinate frame by the proposed method compared to the GT of the experiment, where all algorithms are initialized when Husky2 is partially visible. . . . .	55
3.17	RGB and corresponding depth snapshots from the PTB data set [7]: (a),(b): bear_front, (c),(d): child_no1, (e),(f): face_occ5, (g),(h): new_ex_occ4, and (i),(j): zcup_move_1. The RGB images display our obstacle estimation in green rectangular boxes, with ground truth annotations represented by red rectangular boxes. . . . .	56
3.18	$ACC$ plot of our estimation for (a) bear_front, (b) child_no1, and (c) new_ex_occ4 video sequences from the PTB dataset [7]. An $ACC = 1$ indicates best estimation. . . . .	57



3.19	Results of our proposed obstacle tracking system for the (a) ball10_wild, (b) cube03_indoor, (c) duck03_wild, (d) hand01_indoor, (e) human02_indoor, (f) pot_indoor, (g) squirrel_wild, and (h) suitcase_indoor video sequences from the DepthTrack dataset [52]. . . . .	58
3.20	Comparison of the proposed method with the VbOAD [4] algorithm in the presence of a dynamic obstacle that changes its shape and size abruptly during motion: (a) Tracking output of VbOAD, (b) Tracking output of the proposed method. . . . .	60
3.21	Performance of the proposed tracking algorithm with two moving obstacles alongside multiple static obstacles: (a–d): Time-sampled snapshots displaying the results using our proposed method. . . . .	62
3.22	Comparison of the proposed method with the VbOAD [4] algorithm on the market sequence of the Open LORIS-Scene dataset [58] with multiple dynamic obstacles of different sizes: (a) (i–vi) Time-sampled snapshots with the output of VbOAD, (b) (i–vi) Time-sampled snapshots with the output of our proposed method. . . . .	63
3.23	The effectiveness of the proposed dynamic thresholding compared to fixed thresholding as presented in VbOAD [4] on the market sequence of the Open LORIS-Scene dataset [58]: (a) Example 1, (b) Example 2. . . . .	64
3.24	Performance of the proposed tracking algorithm with a very fast-moving obstacle: (a–e): Time-sampled snapshots displaying the results using our proposed method. . . . .	65
3.25	Comparison of minimum, median, average, and maximum execution times for single obstacle tracking. . . . .	66
3.26	Comparison of continuous execution time for single obstacle tracking. . . . .	67
3.27	Box plot of the execution time for single obstacle tracking, as illustrated in Fig. 3.26. . . . .	67
4.1	CORB2I-SLAM architecture. . . . .	70
4.2	Internal modular design of a client in CORB2I-SLAM. . . . .	72
4.3	Improved accuracy with pose-consistency evaluation: (a) GPS ground truth path for Malaga 07 dataset [116] on Google Maps, (b) Map is generated using ORB-SLAM2 [15], (c) Merged camera track with our pose-consistency evaluation, where white and green denote two individual camera tracks, (d–e) Magnified views showcasing map fusion accuracy at location P3. . . . .	76
4.4	Information flow after camera tracking failure. . . . .	78
4.5	Internal modular design of the Server in CORB2I-SLAM. . . . .	80

4.6	Snapshots of the CORB2I-SLAM map fusion experiment on EuRoC sequences [120]: (a) Map fusion among multiple agents, (b) Map fusion among sub-maps of a single agent and multiple maps of multiple agents.	86
4.7	Test on real autonomy: Each column shows the camera view and corresponding map, (a, c) display camera views of <i>Client</i> <sub>1</sub> , (e, g) show camera views of <i>Client</i> <sub>2</sub> , (b, d) illustrate the camera trajectories of <i>Client</i> <sub>1</sub> , (f) depicts the camera trajectories of both <i>Client</i> <sub>1</sub> and <i>Client</i> <sub>2</sub> before fusion, (h) presents the entire fused map of <i>Client</i> <sub>1</sub> and <i>Client</i> <sub>2</sub> .	88
4.8	Server execution time with respect to the number of KFs to be processed. The values are tested on EuRoC MH01 to MH04 sequences with an average of 5 executions.	90
5.1	Block diagram of the proposed system for dynamic obstacle detection and tracking using 3D LiDAR.	94
5.2	(a) Visualization of coordinate frames (base_link and Velodyne) on a Husky robot [102] in Rviz [99], (b) Isolated display of coordinate frames without the robot for clarity, with red, green, and blue arms representing the X, Y, and Z axes, respectively.	95
5.3	Transformation from the Velodyne coordinate frame $L$ to the virtual camera coordinate frame $C_v$ : (a) The Velodyne coordinate frame $L$ , (b) An intermediate coordinate frame achieved with $+90^\circ$ rotation along the X-axis of $L$ , (c) The virtual camera coordinate frame $C_v$ is obtained by rotating the intermediate coordinate frame $-90^\circ$ along the Y-axis.	97
5.4	Comparison of u-depth formation using LiDAR point cloud (a–c) and RGB-D camera (d–e): (a) A sample snapshot of a LiDAR point cloud, (b) Corresponding thresholded u-depth, (c) Corresponding u-depth after closing operation, (d) Depth image of same view from RGB-D camera, (e) Corresponding thresholded u-depth map.	100
5.5	U-depth representation from a point cloud: (a) An RGB snapshot from the KITTI data sequence [133], (b) A snapshot of the corresponding LiDAR point cloud, (c) A virtual depth representation created using the point cloud, (d) Corresponding u-depth representation, (e) Corresponding u-depth representation after thresholding and closing operations.	101

5.6	Comparison between v-depth and restricted v-depth representations on the KITTI data sequence [133]: (a) Formulated virtual depth image from the LiDAR point cloud. The red arrows indicate the height of an obstacle, (b) Corresponding v-depth representation, (c) Corresponding restricted v-depth representation. The red arrows from (b) to (c) indicate the specific obstacle segmented out from the v-depth representation.	103
5.7	Estimation of dimensions of an obstacle using u-depth and restricted v-depth representation on the KITTI data sequence [133]: (a) Formulated virtual depth image from the LiDAR point cloud, (b) Corresponding u-depth representation. The green arrows indicate estimating the width of the obstacle from the u-depth representation, (c) Corresponding restricted v-depth representation. The red arrows indicate estimating the height of the obstacle from the restricted v-depth map. . . . .	105
5.8	(a) An axis-aligned bounding box, initially estimated, is displayed in blue, (b) The corresponding oriented bounding box is shown in blue. . .	106
5.9	(a) 3D occupancy map representation using Octomap [134], (b) A magnified view shows an obstacle represented with multiple consecutive occupied 3D voxels. . . . .	107
5.10	Comparison of occupancy representation: (a) The car is the dynamic obstacle from the KITTI data sequence [133], (b) Corresponding point cloud, (c) Occupancy representation using Octomap, (d) Occupancy representation using the proposed method. . . . .	107
5.11	Time comparison for creating occupancy information from point clouds between Octomap (left vertical axis) and the proposed method (right vertical axis). . . . .	108
5.12	Obstacle detection using the proposed method on the point cloud from Fig. 5.5. An Rviz [99] snapshot displays 3D bounding boxes of detected obstacles in the coordinate frame $W$ , with the pose of the coordinate frame $L$ . . . . .	110
5.13	(a) The simulated environment in Gazebo [95] with two Husky robots [102], circled in red, (b) A Husky robot equipped with a Velodyne LiDAR and a RealSense D400 [43] RGB-D camera. . . . .	112
5.14	Comparison plots of the relative positions of the second Husky from the first Husky among the present system, 3DPCO, EODT, and the ground truth. The curve closer to the ground truth indicates better estimation. (a) X-axis, (b) Y-axis, (c) Z-axis. . . . .	113

5.15	Thin obstacle detection using the proposed method: (a) An RGB snapshot from the KITTI dataset [133], (b) The corresponding Rviz [99] snapshot displays the 3D bounding boxes of the detected obstacles, (c) A magnified view containing multiple poles on the road as detected obstacles, (d) Another magnified view showing a pole as an obstacle. The arrows indicate the corresponding pole in other images. . . . .	115
5.16	A visual comparison between the estimated centroid (in red) and the ground truth centroid (in green): (a) Top view, (b) Side view. . . . .	116
5.17	A X-axis comparison plot with the ground truth. The curve closer to the ground truth indicates better estimation. . . . .	116
5.18	A Y-axis comparison plot with the ground truth. The curve closer to the ground truth indicates better estimation. . . . .	117
5.19	Error plot of Euclidean distance estimation. . . . .	117
5.20	Experimental Setup – A TurtleBot3 Waffle Pi [135] equipped with a CE30-D Solid State 3D LiDAR. . . . .	118
5.21	Time-sampled snapshots of the experimental environments, where two robots are in motion. The first and the second robots are encircled in red and yellow, respectively. . . . .	119
5.22	(a) Captured point cloud is shown in Rviz [99], (b) Detected obstacles on the point cloud by the proposed method. . . . .	120
5.23	Comparison plots of the estimated relative positions of the dynamic obstacle (the second robot) from the first robot by the present system. The curve closer to the ground truth indicates better estimation. (a) X-axis, (b) Y-axis, (c) Z-axis. . . . .	121
5.24	Relative distance comparison plots for the dynamic obstacle (second robot) from the first robot. Closeness to the ground truth curve indicates better estimation. . . . .	122
5.25	An Rviz [99] view of detected obstacles at a specific time instance with the dataset in [136]: (a)–(b) Side views, (c) Top view. Multiple views establish accurate obstacle estimation with UAV motion. . . . .	123
6.1	The proposed system block diagram for autonomous visual inspection of aircraft using multiple UAVs. . . . .	130
6.2	A Hector quadrotor carries a Kinect RGB-D camera and a Velodyne 3D LiDAR. The three coordinate frames $B$ , $L$ , and $C$ are shown, where X, Y, and Z axes are in red, green, and blue, respectively. The transformations ${}^B\mathbf{T}_L$ and ${}^B\mathbf{T}_C$ are annotated with yellow arrows. . . . .	130
6.3	Gazebo model of an airport environment. . . . .	132

6.4	Multiple geometrical shaped colored patches on the aircraft's outer surface.	133
6.5	The map output from LIO-SAM visualized from different view points. .	134
6.6	Aircraft model extracted from the LIO-SAM map and visualized from multiple view points. . . . .	135
6.7	Templates of different landmarks are created from the aircraft model: (a) Template of the left engine, (b) Template of the tail. . . . .	135
6.8	Octomap representation of the simulated airport created using the LIO-SAM point cloud. . . . .	136
6.9	Bounded regions (red and yellow) for safe UAV operation. . . . .	137
6.10	Process flow diagram for registering the world coordinate frame $W$ of a UAV to the fixed coordinate frame $F$ . . . . .	139
6.11	An Rviz snapshot is explaining the current pose ('uav1/base_link') of the UAV and the goal pose ('uav1/goal_pose') in the coordinate system $W$ for reaching the corresponding landmark (Nose). . . . .	141
6.12	The output of the registration with the coordinate frame $F$ with two UAVs on Rviz visualizer: (a) The pose of $UAV_2$ relative to $UAV_1$ is indicated with an arrow from the base_link of $UAV_2$ ('uav1/uav2') to the base_link of $UAV_1$ ('uav1/base_link'), (b)–(c) Closer views of $UAV_2$ , where two poses indicate the estimated relative pose ('uav1/uav2') and the ground truth pose ('uav1/uav2_gt') from $UAV_1$ . . . . .	143
6.13	A sample output of the object detection module, where the detected three classes are annotated with bounding boxes and object IDs. . . . .	145
6.14	(a) A Gazebo snapshot illustrates a UAV positioned at the optimal viewing pose for Obj_1, (b) The corresponding Rviz visualization displays the base_link frame facing Obj_1 (three consecutive green triangles) and the captured RGB-D point cloud, where Obj_1 is visible. . . . .	146
6.15	(a) The segmented point cloud for Obj_1, representing the area of Obj_1, (b) The calculated maximum distance using the two furthest ending corner points. . . . .	147
6.16	Obtain planned path (green) and actual path traced by the UAV (yellow) for inspecting various points in different runs. . . . .	149
6.17	Error in the relative pose of $UAV_2$ from $UAV_1$ . Any value close to zero represents a more accurate estimation. . . . .	150
6.18	Error plot for area measurements of all three objects. . . . .	151
6.19	Error plot for the maximum distances of all three objects. . . . .	151
A.1	The perspective projection of a 3D point $M_C$ onto an image plane $M_I$ using the pinhole camera model. . . . .	158

A.2	Perspective projection incorporating the world, camera and image coordinates using pinhole camera model. . . . .	160
A.3	An example of matched ORB feature points between two images captured from two different view points. . . . .	163
A.4	Perspective projection of a 3D point $M_W$ in two different views as $M_l$ and $M_r$ . . . . .	164
A.5	Epipolar geometry. . . . .	164
A.6	An example of sparse 3D reconstruction using SfM pipeline: <b>(a)</b> Sample images that are used for the reconstruction, <b>(b)</b> Sparse reconstructed structure and estimated camera poses. . . . .	169
A.7	A dense reconstruction using PMVS2/CMVS pipeline [172], [173]: <b>(a)</b> – <b>(b)</b> Multiple views. . . . .	170

# List of Tables

3.1	Configurations of all experimental datasets . . . . .	45
3.2	Categories of all experiments . . . . .	46
3.3	Parameter Values of the Proposed Technique . . . . .	46
3.4	Experimental evaluation on PTB[7] and DepthTrack[52] datasets . . . . .	59
4.1	Size of Communication Messages . . . . .	83
4.2	Experimental Descriptions . . . . .	83
4.3	The RMS-ATE (meter) of single agent for experiments on EuRoC sequences [120]. . . . .	84
4.4	The RMS-ATE (meter) of multiple agents for experiments on EuRoC sequences [120]. . . . .	85
4.5	Quantitative details for experiment on Freiburg2 sequences. . . . .	87
5.1	Configurations of all experimental datasets . . . . .	111
5.2	Parameter Values in the Proposed System . . . . .	111
5.3	Comparison on the Dynamic Obstacle Detection on KITTI Object Tracking Dataset [133] . . . . .	114
5.4	Comparison of different methods in terms of execution Time (milliseconds). . . . .	123
6.1	Parameter Values in The Proposed System . . . . .	148
6.2	Effective and Actual Travelled Distances . . . . .	149
6.3	Execution Timing Details . . . . .	152





# List of Algorithms

1	: U-depth-map( ) . . . . .	32
2	: Restricted V-depth-map( ) . . . . .	37
3	: U-Depth Representation( ) . . . . .	99
4	: Restricted V-Depth Representation( ) . . . . .	104
5	: Local Path Planning( ) . . . . .	138



# List of Abbreviations

<b>2D</b>	Two( <b>2</b> ) <b>D</b> imensional
<b>3D</b>	Three( <b>3</b> ) <b>D</b> imensional
<b>3DPCO</b>	“Research on <b>3D P</b> oint <b>C</b> loud Data Preprocessing and Clustering Algorithm of <b>O</b> bstacles for Intelligent Vehicle” [1]
<b>AGVs</b>	<b>A</b> utonomous <b>G</b> round <b>V</b> ehicles
<b>AMRs</b>	<b>A</b> utonomous <b>M</b> obile <b>R</b> obots
<b>BA</b>	<b>B</b> undle <b>A</b> djustment
<b>BoW</b>	<b>B</b> ag of <b>W</b> ords
<b>CPP</b>	<b>C</b> overage <b>P</b> ath <b>P</b> lanning
<b>CNN</b>	<b>C</b> onvolutional <b>N</b> eural <b>N</b> etwork
<b>DCF</b>	<b>D</b> iscriminative <b>C</b> orrelation <b>F</b> ilter
<b>DoF</b>	<b>D</b> egrees of <b>F</b> reedom
<b>EODT</b>	“ <b>E</b> fficient <b>O</b> bstacle <b>D</b> etection and <b>T</b> racking Using RGB-D Sensor Data in Dynamic Environments for Robotic Applications” [2]
<b>FAST</b>	<b>F</b> eature from <b>A</b> ccelerated <b>S</b> egment <b>T</b> est
<b>FCL</b>	<b>F</b> lexible <b>C</b> ollision <b>C</b> hecking <b>L</b> ibrary
<b>FN</b>	<b>F</b> alse <b>N</b> egative
<b>FO3D2D</b>	“ <b>F</b> ast <b>O</b> bstacle <b>D</b> etection Using <b>3D-to-2D</b> LiDAR Point Cloud Segmentation for Collision-free Path Planning” [3]
<b>FOV</b>	<b>F</b> ield Of <b>V</b> iew
<b>FP</b>	<b>F</b> alse <b>P</b> ositive
<b>FPFH</b>	<b>F</b> ast <b>P</b> oint <b>F</b> eature <b>H</b> istograms
<b>GPS</b>	<b>G</b> lobal <b>P</b> ositioning <b>S</b> ystem
<b>GPU</b>	<b>G</b> raphics <b>P</b> rocessing <b>U</b> nit
<b>GT</b>	<b>G</b> round <b>T</b> ruth
<b>IMU</b>	<b>I</b> nertial <b>M</b> easurement <b>U</b> nit
<b>KFs</b>	<b>K</b> eyframes
<b>LiDAR</b>	<b>L</b> ight <b>D</b> etection <b>A</b> nd <b>R</b> anging
<b>MPs</b>	<b>M</b> ap <b>P</b> oints
<b>MAVs</b>	<b>M</b> icro <b>A</b> erial <b>V</b> ehicles
<b>NDT</b>	<b>N</b> on- <b>D</b> estructive <b>T</b> esting

<b>ORB</b>	Oriented FAST and Rotated BRIEF
<b>PFH</b>	Point Feature Histograms
<b>PnP</b>	Persepective- <b>n</b> -Point
<b>PRM</b>	Probabilistic Road Map
<b>RANSAC</b>	RANdom SAmple Consensus
<b>RCNN</b>	Region-based Convolutional Neural Network
<b>RMS-ATE</b>	Root Mean Square of Absolute Translation Error
<b>ROI</b>	Region Of Interest
<b>ROS</b>	Robot Operating System
<b>SfM</b>	Structure from Motion
<b>SIFT</b>	Scale Invariant Feature Transform
<b>SoA</b>	State of the Art
<b>SURF</b>	Speeded Up Robust Features
<b>SVD</b>	Singular Value Decomposition
<b>ToF</b>	Time of Flight
<b>TP</b>	True Positive
<b>UAVs</b>	Unmanned Aerial Vehicles
<b>VbOAD</b>	“Robust Vision-based Obstacle Avoidance for Micro Aerial Vehicles in Dynamic Environments” <a href="#">[4]</a>
<b>VIO</b>	Visual Inertial Odometry
<b>VI-SLAM</b>	Visual Inertial Simultaneous Localization And Mapping
<b>VO</b>	Visual Odometry
<b>VOT</b>	Visual Object Tracking
<b>VSLAM</b>	Visual Simultaneous Localization And Mapping

# List of Symbols

$B$	IMU/Body/base_link coordinate frame
$C$	Camera coordinate frame
$C_v$	Virtual Camera coordinate frame
$F$	Template coordinate frame
$I$	2D image coordinate frame
$L$	Velodyne LiDAR coordinate frame
$W$	World coordinate frame
${}^B\mathbf{T}_C$	Fixed transformation from the Camera to the base_link coordinate frame
${}^B\mathbf{T}_L$	Fixed transformation from the Velodyne to the base_link coordinate frame
${}^{C_v}\mathbf{T}_L$	Transformation from the Velodyne coordinate frame to the virtual camera coordinate frame
${}^W\mathbf{T}_B$	Robot Pose in the World coordinate frame
${}^W\mathbf{T}_C$	Camera Pose in the World coordinate frame
$\varphi(\cdot)$	Projection function that projects 3D world points to 2D image plane
$\Lambda$	Constant for u-depth thresholding
$\wp$	Constant for dynamic u-depth thresholding
$\odot$	Angle between two vectors
$\Phi$	Minimum number of ordered contour pixel points for an obstacle in u-depth image
$\mathbb{R}^2$	Two dimensional space
$\mathbb{R}^3$	Three dimensional space
$SE(3)$	Special Euclidean group [5]
$\Psi_{\varkappa_i, C_k}$	Uncertainty of observing a 3D point $\varkappa_i$ from a camera $C_k$
$\rho$	Unbiased Gaussian vector to represent uncertainty of an estimated camera pose
$\oplus$	Lie algebra approximation around an estimated camera pose
$\mathbf{Cov}_{C_k}$	Covariance matrix to represent observability accuracy of camera $C_k$
$\mathbf{J}_{\varkappa_i, C_k}$	Jacobian of observability measurement of camera $C_k$ for point $\varkappa_i$

$SO(3)$  Special Orthogonal Group in Three Dimensions

$Sim(3)$  Similarity Group in Three Dimensions

*I dedicate this thesis to the divine presence of Lord Balaji, whose boundless grace has guided me through every step of my academic journey. In moments of doubt and challenge, it is your unwavering support that has infused me with strength and resilience.*

*Lord Balaji, your divine blessings have illuminated my path, inspiring me to pursue knowledge with unwavering dedication and sincerity. You are the beacon of wisdom and the embodiment of divine benevolence, and it is with profound gratitude that I offer this humble work at your lotus feet.*

*May your divine blessings continue to shower upon me, guiding my endeavors and illuminating my quest for truth and understanding. With deep reverence and devotion, I dedicate this thesis to the divine presence of Lord Balaji.*





# Chapter 1

## Introduction

The past two decades have witnessed an explosion of technological advancements in robotics and its related domains, driven by the keen interest of researchers in these active research areas. This advancement has exerted a direct influence on nearly every industry, fostering a widespread adoption of robotic solutions. Consequently, there has been a substantial growth in the integration of robotic solutions across various industries. Industrial robots can be broadly classified into two groups:

- The first group comprises stationary robots that execute tasks without changing their positions, exemplified by robotic arms, welding robots, and similar applications.
- The second group consists of mobile robots that traverse their environments to perform assigned tasks. Examples include Autonomous Mobile Robots (AMRs), Autonomous Ground Vehicles (AGVs), Unmanned Aerial Vehicles (UAVs), and Humanoids.

Industrial mobile robots find applications in diverse fields such as farming, agriculture, manufacturing, healthcare, logistics, retail, hospitality, and services. This diversity has led to a notable proliferation of various industrial robotic applications. The maturity achieved in the autonomous navigation of robots has opened up possibilities for deploying multiple robots simultaneously in scenarios where a single robot may be inefficient due to the scale of the task or where quick completion is essential within a given timeframe. Utilizing multiple heterogeneous mobile robots in large-scale missions is advantageous because complex tasks can be decomposed and assigned to different robots based on their capabilities, thereby reducing the overall completion time. This thesis delves into different aspects and challenges associated with autonomous visual navigation in unknown environments by robots.

Throughout the remainder of the thesis, the terms ‘robots’, ‘clients’, or ‘agents’ are used interchangeably to denote the participating robots, and bold lowercase letters represent vectors, while bold uppercase letters or combinations thereof signify matrices.

The subsequent sections of this chapter are structured as follows: Section 1.1 focuses on the preliminaries, providing essential background information. Section 1.2 outlines the fundamental principles of visual navigation by autonomous robots. Section 1.3 articulates the objectives of the thesis, while Section 1.4 elucidates the specific contributions of the work. Finally, Section 1.5 delineates the organizational structure of the remaining chapters in this thesis.

## 1.1 Preliminaries

In contemporary times, robots are universally outfitted with either a single or multiple cameras, which capture two-dimensional (2D) images of their surrounding environments as they attempt to comprehend their surroundings. In this regard, understanding the relationship between the three-dimensional (3D) environment and the corresponding 2D images becomes crucial. Appendix A.1 provides a concise explanation of this fundamental relationship. Additionally, Appendix A.2 delves into the fundamentals of reconstructing a 3D structure from a compilation of 2D images using Structure from Motion (SfM) [6].

## 1.2 Autonomous Navigation of Robots

The autonomous navigation of mobile robots in unfamiliar environments necessitates an understanding of the surroundings, represented as a 3D structure, to determine a navigation path by avoiding obstacles. Simultaneously, it requires continuous awareness of the robot's location within the environment. This 3D representation of the environment is referred to as a map, while the estimation of the robot's poses (position and orientation) within the environment is termed self-localization. As detailed in Appendix A.2, SfM [6] emerges as a significant technology for reconstructing a 3D structure from an assortment of unordered 2D images and for estimating the poses associated with those images. This technology finds equal applicability in the realms of self-localization and map creation while the robot is in motion. However, a notable drawback of SfM lies in its exponentially growing execution time, rendering it unsuitable for real-time robotic motion estimation, which is an essential requirement for effective navigation. Consequently, researchers have devised variations of the core technology that preserve its essence while being capable of real-time execution. Within the context of this thesis, we delve into the evolution of different techniques associated with autonomous navigation, including obstacle estimation and tracking for safe path planning, as well as self-localization and map generation using visual sensors.

### 1.2.1 Obstacle Estimation and Path Planning

A mobile robot operating autonomously within an environment necessitates a comprehensive understanding of both occupied and vacant regions derived from the reconstructed map. Occupied regions represent obstacles that must be avoided, directing the robot's path exclusively through empty spaces. These obstacles typically fall into two categories: static and dynamic. Dynamic obstacles include movable entities such as humans, cars, and other robots. Consequently, obstacle detection stands as a crucial task in enabling autonomous navigation for mobile robots. Fig. 1.1 depicts a sample captured image in a static environment along with the corresponding detected obstacles. The complexity of this task amplifies in dynamic and cluttered environments,

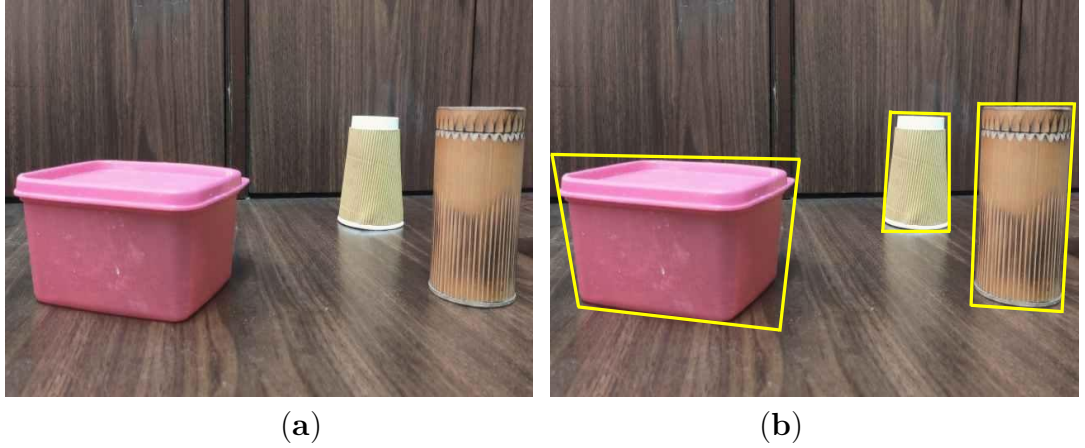


FIGURE 1.1: An example of obstacle detection: (a) An RGB image captured in a static environment, (b) Detected obstacles are highlighted with yellow bounding boxes.

where the objectives are threefold:

- Detect obstacles within the field of view (FOV) of sensors mounted on robots.
- Measure the states of obstacles, encompassing whether they are static or dynamic, their dimensions, and the velocities of dynamic obstacles.
- Predict the future locations of obstacles to plan for collision-free navigation.

An obstacle, in this context, refers to any object capable of impeding the motion of a mobile robot, spanning AMRs, AGVs, or UAVs. These obstacles can be of varying shapes and sizes, with dynamic obstacles dynamically altering their characteristics. Detecting and tracking multiple obstacles in a cluttered environment becomes even more intricate due to these diverse attributes. A mobile robot frequently requires the ability to comprehend and track multiple dynamic obstacles swiftly, responding promptly to avoid potential collisions. Therefore, the entire process must be executed

onboard to circumvent communication delays. Furthermore, the obstacle detection and processing must be not only accurate but also swift enough for real-time execution onboard. Despite the progress made, existing approaches still fall short in addressing all the diverse characteristics that obstacles may exhibit.

Similar to obstacle detection, Visual Object Tracking (VOT) is a pertinent research topic in computer vision. The primary objective of VOT is to track one or more specified objects within a given video sequence. However, the concept of obstacle tracking differs from conventional VOT. To illustrate these distinctions, consider Fig. 1.2, which features three non-consecutive snapshots from the PTB dataset [7]. The results

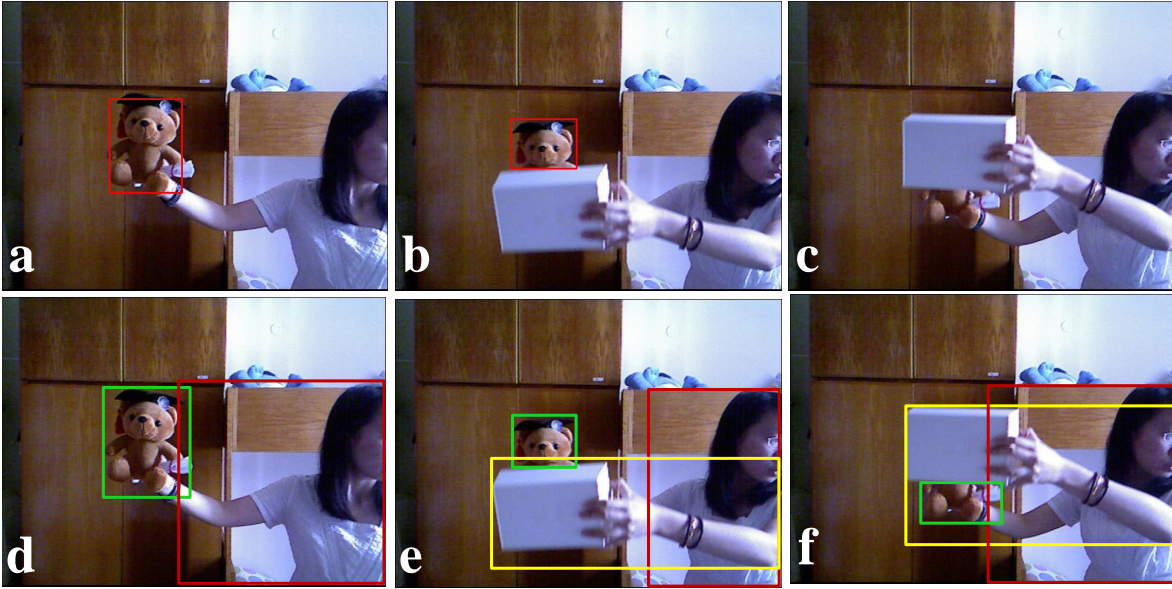


FIGURE 1.2: Three non-consecutive snapshots from the PTB [7] dataset, where (a)–(c) display the ground truth of object tracking, specifically the toy bear, marked with red bounding boxes, in contrast, (d)–(f) illustrate the likely outcomes of obstacle detection, marked by boxes in red, yellow, and green.

of object tracking are depicted in Fig. 1.2(a)–(c), focusing on a toy bear as the object of interest. The red bounding boxes indicate the detected portions of the bear. In Fig. 1.2(a), the bear is successfully detected. In Fig. 1.2(b), the lower portion of the bear is obscured by a box, resulting in only the upper portion being detected. Meanwhile, Fig. 1.2(c) reveals that VOT failed to detect the bear’s major portion (face) due to occlusion. Unlike VOT, where specific objects are tracked, obstacle detection aims to identify any object in front of the capturing device that could obstruct the robot’s motion. Therefore, the main goal in obstacle detection is to detect all objects, with no specific object of interest. Fig. 1.2(d)–(f) illustrates potential outputs of an obstacle detection method. Fig. 1.2(d) shows detected obstacles such as a bear and a lady marked in green and red boxes, respectively. Fig. 1.2(e), (f) present scenarios with

three obstacles marked in green, red, and yellow boxes, highlighting contextual differences that restrict the direct application of VOT to obstacle detection and tracking in a robotic environment for navigation purposes.

The task of path estimation or planning for a mobile robot involves determining a sequence of maneuvers necessary for the robot to reach its destination from its starting or current location while avoiding collisions with obstacles. This assumes a mobile robot with a certain degree of freedom within a 3D environment and a set of obstacles within that space. A continuous path is deemed legal only if it neither intersects any obstacles nor self-intersects. Fig. 1.3 depicts a single sample path among many possible paths for the example in Fig. 1.1. Various algorithms address path planning and broadly fall

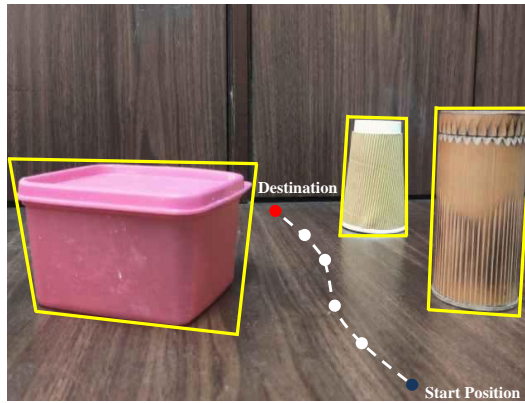


FIGURE 1.3: A sample path to reach the destination for the previous example in Fig. 1.1.

into two categories: (i) occupancy grid-based and (ii) graph-based methods.

- **Occupancy grid-based methods** divide the entire space into 3D voxels (small cuboid grids) and assign them as occupied or free. The planning algorithm considers trajectories as the shortest lines that do not cross any occupied voxels, reaching the destination voxel from the current location voxel.
- **Graph-based methods** create a structure where nodes represent places (e.g., bedrooms, kitchens, balcony), and edges define paths between places (e.g., doors connecting rooms, doors connecting rooms and kitchens). These edges also carry weights representing the difficulty of traversing the path. Finding the trajectory involves determining the shortest path between the current location node and the destination node.

Recent works [8], [9] highlight existing challenges in the path planning domain, such as estimating free spaces of convex shapes, determining navigable space, and identifying the best possible path in real-time.

### 1.2.2 Visual Odometry

Visual Odometry (VO) [10]–[12] is a process for estimating robot poses using inputs solely from either single or stereo cameras attached to robots. It constitutes a specific case of SfM where the cameras are pre-calibrated, and the images are ordered sequentially. The term ‘VO’ derives from the concept of wheel odometry, which estimates vehicle motion by integrating the number of wheel turns over time. VO mirrors wheel odometry in estimating vehicle poses by analyzing changes in sequences of images captured by an onboard moving camera. Specifically, VO focuses on determining the 3D motion of the camera as soon as a new frame arrives, with the entire estimation process expected to complete before the next frame arrives. Consequently, VO works with a sequence of images and necessitates sufficient illumination for efficient functioning. Additionally, consecutive images must have ample scene overlap for accurate estimation. A notable advantage of VO over wheel odometry is its applicability to any robot with at least one camera. Other major advantages include its suitability for Global Positioning System (GPS)-denied areas, immunity to wheel slippage in uneven terrain, and the capability to produce more accurate trajectory estimations, making VO an appealing candidate for continuous robot pose estimation.

The typical steps of VO are outlined below:

- Estimates the pose for the first two images, denoted with  $IMG_0$  and  $IMG_1$ , respectively, through the estimation of Fundamental and Essential matrices. Decompose the Essential matrix to obtain the rotation matrix and translation vector as described in Appendix A.2.
- Use homography [13], [14] based initialization in the case of a planar scene.
- For new  $(k + 1)^{th}$  image, denoted with  $IMG_{k+1}$ , find matched features between the images  $IMG_k$  and  $IMG_{k+1}$ .
- Estimate the Essential matrix ( $\mathbf{E}_{k \dots k+1}$ ) from  $IMG_k$  to  $IMG_{k+1}$  using the matched features.
- Extract the rotation matrix,  $\mathbf{R}_{k \dots k+1}$ , and the translation vector,  $\mathbf{t}_{k \dots k+1}$ , through the decomposition of the Essential matrix and compute the relative pose

$$\mathbf{T}_{k \dots k+1} = [\mathbf{R}_{k \dots k+1} | \mathbf{t}_{k \dots k+1}]$$

.

- Correct the scale of  $\mathbf{t}_{k-1 \dots k}$  and  $\mathbf{t}_{k \dots k+1}$  using three-view geometry [13].



- Calculate the pose of image  $IMG_{k+1}$  relative to the first image as

$$\mathbf{T}_{0\dots k+1} = \mathbf{T}_{0\dots k} \times \mathbf{T}_{k\dots k+1}$$

This incremental estimation is described in Fig. 1.4.

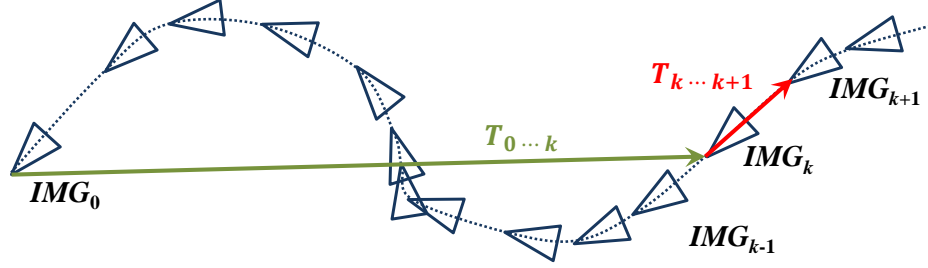


FIGURE 1.4: A pictorial view of incremental camera pose estimation in VO.

- For enhanced camera pose estimation in VO, Bundle Adjustment (BA) [10] is employed on the pose graph linked to the current image  $IMG_{k+1}$ . Appendix A provides the details of BA. In VO estimation, BA is typically applied to a limited subset of images, known as local optimization or sliding window-based BA.

Some of the major demerits of VO are listed below:

- Incremental drift is a significant issue, as errors introduced during image-to-image motion estimation accumulate over time, leading to noticeable drift. While sliding window-based BA, as mentioned above, mitigates this to some extent, incremental drift becomes more pronounced in long-term robotic autonomous navigation, presenting potential challenges.
- VO estimation is in an unknown scale, making it unable to measure the exact distance between two locations in a metric unit.

### 1.2.3 Visual SLAM

Visual Simultaneous Localization and Mapping (VSLAM) [15]–[20] represents another avenue of research in the fields of computer vision and autonomous visual navigation. VSLAM closely aligns with VO but operates with greater precision, imposing additional constraints on camera path estimation and refinement. VSLAM constructs a graph where robot poses serve as nodes, and the special constraints between these poses act as edges, forming what is referred to as the pose graph.

A fundamental distinction between VO and VSLAM lies in their objectives. While VO aims to incrementally estimate the current robot pose and reconstruct the current

scene view, VSLAM seeks to establish a globally consistent robot path and globally correct 3D structure reconstruction. To achieve this, VSLAM introduces additional verification steps and restrictions, with loop closure detection being a pivotal verification step. Loop closure detection occurs when the robot revisits a previous location, allowing the system to add special constraints among recent and old poses, enhancing pose graph connectivity. Unlike VO, VSLAM retains the entire robot path and map structure from initialization, leading to potential operational inefficiencies due to the rapid accumulation of data. To address this, VSLAM introduces the concept of keyframes (KFs) and eliminates redundant images, ensuring a lightweight and operable system for extended durations. VSLAM places significant emphasis on optimization for creating a global and consistent map, employing sliding window-based Bundle Adjustment similar to VO after each frame insertion. Additionally, VSLAM performs BA on the entire loop whenever a loop is detected and on the entire robot path to maintain a consistent global map. Consequently, VSLAM emerges as a stable framework for autonomous visual navigation, finding widespread applications in AMRs, UAVs, and UGVs.

VSLAM can be broadly categorized into two types: (i) feature-based methods and (ii) direct methods.

- **Feature-based methods:** These methods consider specific points, lines, corners, etc., as features in the images, assuming their uniqueness for easy identification across consecutive images. Feature-based methods extract and match these features to estimate camera poses, as visual odometry (Section 1.2.2).
- **Direct methods:** In contrast, direct methods utilize a large number of pixels or all pixels without feature extraction. These methods track pixel movements based on photogrammetric properties such as color, brightness, and intensity gradient, shifting the problem from geometric error minimization to photometric error minimization. Direct methods yield denser maps, offering advantages in completeness, object inference, and robust tracking, especially in featureless environments and changing lighting conditions.

This thesis primarily focuses on the feature-based method due to its suitability for collaborative VSLAM, as discussed in [21]. Consequently, the basic architecture of a feature-based VSLAM is presented. Fig. 1.5 illustrates the architecture of a well-known feature-based VSLAM, ORB-SLAM2 [15], supporting monocular, stereo, and RGB-D cameras.

The architecture is distributed across three major threads: tracking, mapping, and loop closure and optimization.



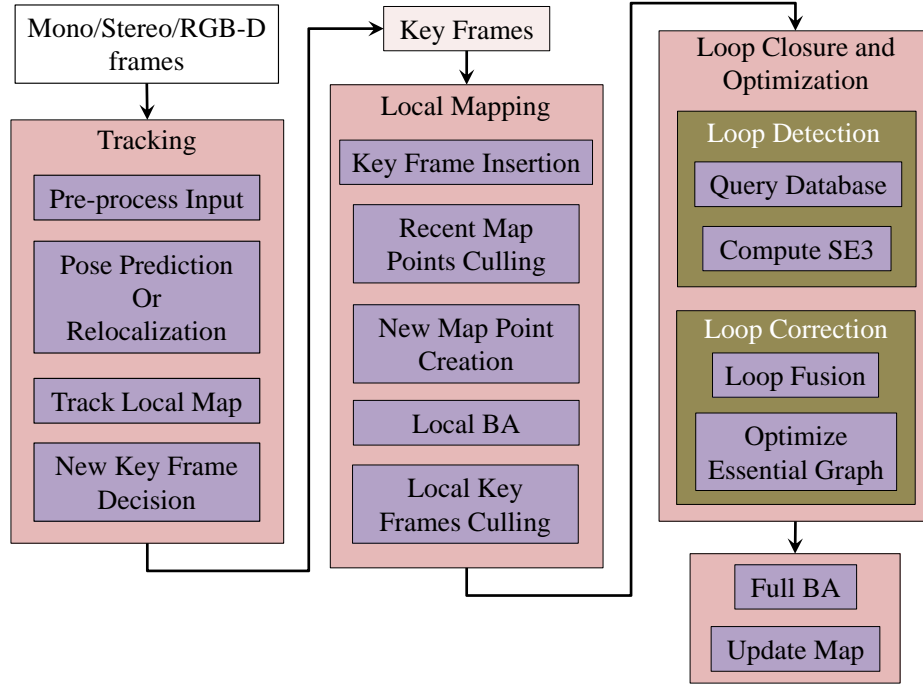


FIGURE 1.5: ORB-SLAM2: A VSLAM Framework.

- **Tracking Thread:** This thread receives input images from the camera, pre-processes them to find matched features, predicts the initial pose using a linear motion model, refines the pose through matches from local map points (MPs), and decides whether to consider the image as a new KF.
- **Mapping Thread:** This thread is responsible for creating the map, inserting new KFs, and performing maintenance operations on the map, such as inserting new KFs and MPs, removing redundant KFs and MPs, and running local BA.
- **Loop Closure Thread:** This thread continually searches for matches with previously visited locations using a database and identifies a loop in the robot path when a stable match is found. It corrects the entire path by introducing new edges between old and new KFs and refines the map with global BA on the entire loop.

Despite its advantages, VSLAM faces inherent challenges such as losing camera tracking and all estimations residing in unknown scale. The estimation of robot poses through feature matching relies on substantial scene overlap between consecutive frames, posing difficulties when a robot moves freely. Consequently, VSLAM systems frequently struggle to find enough matched features, resulting in an unstable pose estimation, a situation known as losing camera track [15]. In such scenarios, VSLAM systems resort to re-localization, searching for sufficient feature matches with

all previous keyframes before resuming standard camera tracking procedures. Moreover, VSLAM, like SfM and VO, is unable to determine the exact metric scale, leaving the estimation in an unknown scale. This limitation means that obtaining absolute scale in VSLAM, essential for navigating specific distances, remains unattainable.

#### 1.2.4 Visual-Inertial SLAM

Visual-Inertial SLAM (VI-SLAM) stands as an alternative SLAM framework in which camera tracking leverages inputs from two distinct sensors: the Inertial Measurement Unit (IMU) and the camera. The IMU sensor comprises three sensors, namely accelerometer, gyroscope, and magnetometer. The accelerometer measures linear acceleration across three axes (roll, pitch, yaw), the gyroscope measures rotational rates along these axes, and the magnetometer aids in gravitational force measurement. Consequently, the IMU provides a preliminary measurement of the robot's motion, serving as an initial estimate for camera motion. This initial estimate facilitates the discovery of matched features in a guided manner, particularly beneficial for scenarios involving significant or jerky motion. This approach minimizes the risk of losing camera track, a persistent issue in VSLAM. The process of estimating the camera track using both IMU and camera sensors is known as Visual-Inertial Odometry (VIO) [22]–[25], and VI-SLAM employs the VIO technique for camera tracking. Fig. 1.6 illustrates the IMU-integrated version of ORB-SLAM2 [15], where the tracking thread derives the motion model from IMU data, deviating from the linear motion model in ORB-SLAM2. Feature matching utilizes this motion model to enhance feature matching. Notably, motion estimation from the IMU is consistently in metric units, enabling VI-SLAM to provide metric unit estimations, thus supporting navigation over specified distances.

In essence, VIO is anticipated to yield more accurate pose estimations than VO, contingent upon the quality of the IMU sensor [18]. However, the potential introduction of noise in IMU data may result in a correspondingly noisy estimation of camera motion, subsequently impacting the precision of camera pose estimation. Consequently, in practical applications, the presence of noisy IMU measurements poses a challenge to the quality of the combined measurements. As a result, the system necessitates intelligence to validate IMU measurements before integrating them with other sensors, introducing a challenging aspect to system design.

#### 1.2.5 Collaborative VSLAM

In scenarios involving extensive and distributed missions, relying on a single robot may prove inadequate due to mission deadlines and individual robot limitations, such as the

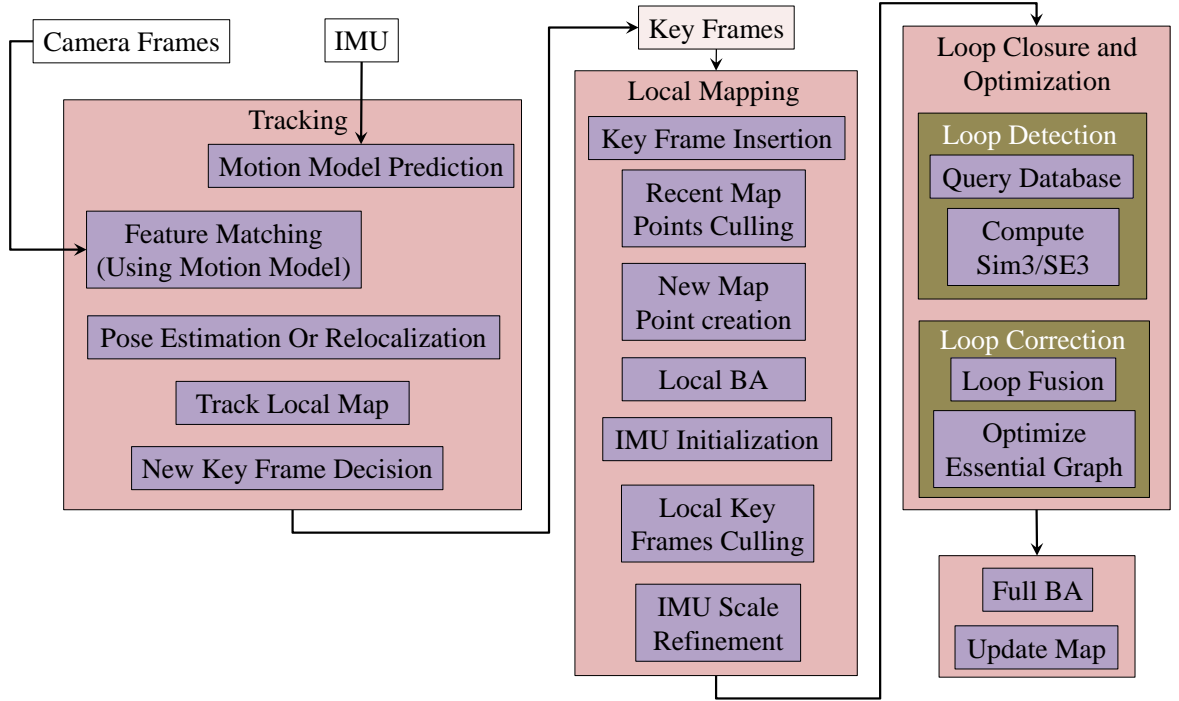


FIGURE 1.6: An IMU integrated version of ORB-SLAM2.

limited endurance of commercial drones. Drones, for instance, often have a maximum endurance of 30 minutes, making them insufficient for extensive missions. Employing multiple heterogeneous robots, combining UGVs and UAVs equipped with various sensors, becomes advantageous in such missions. Collaborative VSLAM frameworks, as discussed in [21], [26], [27], enable multiple robots to share information, collaboratively create a global environmental map, and enhance system reliability. Collaborative approaches facilitate rapid exploration of regions, ensuring system functionality even if one or more robots become inoperable. Two main types of collaborative SLAM frameworks exist: centralized [21], [26], [28] and distributed [27], [29]–[31]. Centralized systems involve a centralized server controlling communication, whereas in distributed systems, participating robots, usually homogeneous, share information directly.

The primary challenges in collaborative frameworks encompass effective communication among robots, ensuring information availability, minimizing information sharing to maintain a small network footprint, fusing partial maps from individual robots, preventing loss of tracking, reusing map information, managing network delays, and more.

To recapitulate the aforementioned discussion, obstacle detection proves integral in autonomous navigation. We explored what constitutes obstacles and why VOT is not directly applicable to detect dynamic obstacles. Challenges in dynamic obstacle detection were highlighted. Appendix A presented the fundamentals of SfM for understanding 3D structure reconstruction from a set of 2D images. We then delved into

how VO evolves from SfM and utilizes similar techniques for autonomous navigation. Challenges in VO were discussed, and VSLAMs were introduced as a solution, addressing some issues. However, VSLAMs may still be inefficient for many autonomous navigation scenarios. In this context, VI-SLAMs exhibit the potential for increased accuracy. For multiple robots, collaborative VSLAMs adopt either distributed or centralized models, each with its own advantages and limitations. Challenges associated with collaborative VSLAMs were also outlined.

### 1.3 Objectives of the Thesis

We have examined various aspects of visual autonomous navigation by mobile robots, witnessing significant progress over the last two decades. Throughout this exploration, we have underscored the prevalent problems and challenges related to obstacle estimation, path planning, self-localization, map generation, among others. This thesis aims to present a comprehensive navigation framework wherein multiple heterogeneous robots autonomously identify surrounding obstacles and collaboratively explore the environment using collaborative VI-SLAM. The proposed navigation framework holds potential applications in distributed autonomous robotic inspection tasks across diverse domains. The primary objectives are delineated below:

- Develop capabilities to estimate dynamic obstacles in highly dynamic environments.
- Ensure quick real-time dynamic obstacle estimation by leveraging existing sensor inputs.
- Establish a collaborative framework wherein multiple robots, equipped with heterogeneous visual sensors, can seamlessly collaborate.
- Enable participating robots to share information within the collaborative framework.
- Facilitate rapid environmental map generation through collaborative efforts.
- Ensure global consistency in the collaborative map for subsequent robotic navigation tasks, such as obstacle detection and manipulation.
- Support long-term navigation within the collaborative framework.
- Ensure stability and robustness of the SLAM algorithm in various scenarios, such as providing accurate estimations in linear motion, minimizing camera track loss, and sustaining performance during jerky motions.

- Foster self-awareness in individual robots to mitigate erroneous estimations.
- Establish an autonomous inspection system wherein robots can execute autonomous navigation for comprehensive inspection tasks.

## 1.4 Contributions of the thesis

This thesis focuses on addressing multiple challenges in robotic navigation, specifically targeting dynamic obstacle avoidance for collision-free safe navigation and promoting collaborative visual navigation with multiple cooperative robots. The comprehensive analysis encompasses two key aspects of autonomous visual navigation by mobile robots, culminating in the presentation of an autonomous visual inspection system utilizing collaborative UAVs. The primary contributions are outlined below:

- **Efficient Dynamic Obstacle Estimation:**
  - Utilizes only depth maps from an RGB-D sensor for precise collision-free navigation in cluttered environments.
  - Implements dynamic binary thresholding on u-depth maps to enhance obstacle detection and accurately estimate their dimensions.
  - Introduces a representation method using restricted v-depth maps for improved estimation of obstacle dimensions.
  - Presents an algorithm for obstacle tracking, employing efficient processing of u-depth maps.
- **Heterogeneous Centralized Collaborative VI-SLAM Framework:**
  - Proposes a collaborative VI-SLAM framework allowing participating robots or clients to carry heterogeneous visual sensors, such as monocular, stereo, or RGB-D cameras.
  - Distributes low computation tasks to clients and offloads high computation tasks to a central server.
  - Implements tightly-coupled VIO or only VO for pose estimation on each robot.
  - Adapts the camera tracking module to auto-adjust for erroneous IMU pose integration.
  - Supports multi-map scenarios on each client, eliminating the need for re-localization.

- Introduces an efficient criterion for estimating the reliability of camera pose, triggering loss of tracking if reliability falls below a threshold.
- Presents a novel algorithm for efficient map fusion on the server, applicable for a single client or collaboration among multiple clients.

- **Improved Dynamic Obstacle Estimation Using 3D LiDAR:**

- Proposes enhanced dynamic obstacle estimation using 3D Light Detection and Ranging (LiDAR) sensor data.
- Defines u-depth and restricted v-depth representations from 3D LiDAR point clouds to facilitate obstacle detection, estimation, and tracking. While depth maps are typically associated with RGB-D cameras, LiDAR provides colorless point clouds. This work’s primary contribution lies in establishing u-depth and restricted v-depth representations from 3D LiDAR point clouds and utilizing them for the detection and estimation of obstacles.
- Eliminates computationally intensive modules, such as ground plane segmentation and clustering, by estimating obstacles from u-depth and restricted v-depth representations.

- **Autonomous Aircraft Inspection System Using Collaborative UAVs:**

- Introduces a novel registration process among participating UAVs, requiring no common landmark for matching and initiating the registration process.
- Enables each UAV to autonomously generate poses to view specific portions of the aircraft surface optimally.

## 1.5 Organization of the thesis

We have delved into fundamental concepts of autonomous visual navigation, encompassing perspective projection and the structure-from-motion pipeline (detailed in Appendix A). Our exploration spanned various self-localization techniques, ranging from visual odometry and visual SLAM to visual-inertial SLAM, along with collaborative visual SLAM for multiple robots. Additionally, we addressed dynamic obstacle detection and path planning. To encapsulate, the thesis is structured as follows:

**Chapter 2:** Provides an in-depth review of existing literature in the domains of single and collaborative VSLAMs and VI-SLAMs. It also examines dynamic obstacle estimation utilizing visual sensors.

**Chapter 3:** Introduces a novel approach for obstacle estimation in dynamic cluttered environments, leveraging depth images from RGB-D cameras. The chapter presents an innovative algorithm for continuous dynamic obstacle tracking, contributing to collision-free path planning. It also highlights the limitations of RGB-D cameras in navigation and obstacle detection.

**Chapter 4:** Proposes a new centralized framework for collaborative visual-inertial navigation among heterogeneous robots, equipped with different visual cameras. The framework evaluates the quality of IMU data before utilization, supports multi-map operations within a single robot or among multiple robots, and incorporates an efficient map fusion algorithm.

**Chapter 5:** Introduces another novel approach for obstacle estimation utilizing 3D LiDAR sensors. The method processes LiDAR data as depth images, enabling quicker and more accurate estimations compared to RGB-D cameras.

**Chapter 6:** Presents a distributed visual inspection task employing multiple collaborative drones. The self-localization and multi-robot cooperation are facilitated by 3D LiDAR sensors, while the inspection task is performed with RGB-D cameras. The chapter introduces a novel registration technique.

**Chapter 7:** Concludes the thesis by summarizing the contributions achieved and outlining potential directions for future work.





## Chapter 2

# Literature Survey

This chapter presents detailed study on the existing works on dynamic obstacles detection and tracking using visual sensors, and further extend the study to VSLAMs and collaborative VSLAMs as these are the backbone of autonomous navigation for mobile robots.

## 2.1 Obstacle Detection

Existing approaches use both active and passive sensors for detecting obstacles. Dynamic obstacle detection commonly relies on active sensors like ultrasonic and radar. Despite their widespread use, these sensors present limitations such as a low field of view and sparse sensing, as detailed in [32], [33]. In this literature survey, our primary emphasis is on exploring conventional geometric-based obstacle detection systems that utilize sensors like stereo cameras and RGB-D cameras.

### 2.1.1 Stereo Images

In obstacle detection, researchers widely use stereo cameras to calculate the distance of obstacles through disparity measurements from image pairs. Researchers place intense focus on disparity processing for detecting obstacles by AGV. Labayrade *et al.* first introduced u-disparity and v-disparity image formation from stereo disparity in [34], [35], explaining their properties for object identification after ground plane segmentation. The work concentrates on detecting obstacles standing on the ground plane, such as other vehicles, trees, and pedestrians. Helen *et al.* introduced a low-latency obstacle avoidance system [36] utilizing u-disparity for rapid obstacle detection in cluttered environments. However, dynamic obstacles were deliberately excluded from the scope of their study. Kormann *et al.* demonstrate improved road segmentation with a spline road model and obstacle detection from  $uv$ -disparity in [37]. Adrian *et al.* showcase

multiple representations of the disparity image and  $uv\theta$ -disparity to achieve obstacle detection in [38]. Song *et al.* [39] present obstacle detection using a considerate  $uv$ -disparity that employs a refined  $v$ -disparity for accurate road segmentation. These approaches are equally applicable in a system aiding visually impaired persons [40] due to the similarities in motion characteristics between humans and AGVs. Therefore, these above-discussed approaches are limited to AGV-type motion.

### 2.1.2 RGB-D Images

Huang *et al.* present an obstacle detection system [41] for indoors using Microsoft Kinect [42], which first denoises the depth image using morphological operations and afterward segments the ground using a  $v$ -depth map. The system considers all remaining regions as obstacles after ground segmentation. The region-growing algorithm tracks dynamic obstacles in subsequent frames. The system limits indoor UGV-type motion and requires high computations for denoising and region-growing operations. Lately, low-cost RGB-D cameras (e.g., RealSense D400 [43]) have gained popularity due to technological advancements in terms of indoor and outdoor capabilities, weight reduction, and form factor. These small, lightweight cameras are perfect for fitting on Micro Aerial Vehicles (MAVs). Therefore, researchers focus on detecting objects and obstacles with RGB-D cameras.

VOT has become highly popular and made substantial progress in object tracking scenarios over the last decade [44]–[46]. While conventional methods predominantly concentrate on tracking objects in RGB video sequences, challenges persist in RGB-based tracking due to factors like cluttered backgrounds, occlusion, and deformation. Consequently, researchers have explored VOT by incorporating RGB-D data to address these challenges. Hannuna *et al.* propose a real-time RGB-D tracker, DS-KCF [47], which is built upon the KCF tracker [48] and uses depth cues to handle occlusion, scale variation, and shape changes. Kart *et al.* propose CSR-rgbd++ [49], a general framework that uses depth segmentation-based occlusion detection in a discriminative correlation filter (DCF) framework. Liu *et al.* propose a three-dimensional extension of the classical mean-shift tracker [50] that deals with occlusions more effectively. Recently, Kart *et al.* also propose OTR, a long term RGB-D tracker that proposes modeling appearance changes via 3D target reconstruction. More recently, Qian *et al.* propose DAL [51], an RGB-D tracker that embeds depth information into deep features through the reformulation of a deep DCF. Very recently, Yan *et al.* propose the first offline trained RGB-D tracker, DeT [52], which is based on two RGB trackers: ATOM [53] and DiMP [54]. It uses an additional depth branch to extract depth features and a module for feature fusion. The system is trained using generated RGB-D

videos from existing monocular RGB tracking training data. The system also uses the DepthTrack [52] dataset for training and showed remarkable performance on the testing set of DepthTrack. These RGB-D trackers are mainly trained using deep networks and require a good amount of contextual data for training. We excluded these RGB-D trackers from the state-of-the-art (SoA) comparison due to contextual differences, as shown in Fig. 1.2 and explained in Section 1.2.1.

Yang *et al.* [55] present a system for dynamic obstacle segmentation that converts a depth image to a point cloud, segments out the planar road, and considers all remaining points as obstacles. The system differentiates static and dynamic obstacles using the DECOLOR algorithm. The system is not suitable for MAVs due to its demand for high computational resources in processing the point cloud. Odelga *et al.* [56] present an obstacle detection and tracking system for teleoperated UAVs that uses a bin occupancy filter, breaking the entire visible region into smaller bins and searching for the presence of an obstacle in a bin in a probabilistic way. Luiten *et al.* [57] present an approach, MOTSFusion, that uses 3D tracklets to estimate dynamic obstacles and their trajectories. The algorithm requires high computation for dense optical flow calculations. Lin *et al.* [4] present a vision-based dynamic obstacle avoidance system for MAV (VbOAD). The system uses u-depth maps for detecting obstacles, a multivariate Gaussian probability density function to track obstacles in subsequent frames, and a Kalman filter-based approach to predict their probable future positions and velocities. Estimation of an obstacle’s dimensions using only u-depth maps can become incorrect, which is further explained in details in Section 3.2.2.2. The system makes a restrictive assumption by assuming fixed obstacle sizes, a constraint that does not align with real-world scenarios, as evidenced in the OpenLORIS-Scene market data sequence [58]. The system uses a predefined obstacle height while detecting obstacles from a u-depth map, limiting the system to a predefined obstacle size. The experimental results are limited to only detecting multiple walking humans in an empty room and corridor.

## 2.2 VO and VSLAM

VSLAM serves as the fundamental backbone for self-localization and environmental map generation in autonomous visual navigation by robots. VSLAMs can be broadly categorized into two groups: feature-based methods and direct methods. The basic concepts behind both types are discussed in Section 1.2.3. The successive sections provide a literature survey of both types of VSLAMs.

### 2.2.1 Feature-based Methods

PTAM [59] is one of the milestone VSLAM architecture proposed by Klein and Murray, presenting a design to run VSLAM real-time on CPU by partitioning the tracking and mapping modules onto multiple threads. The system is designed as a KF-based SLAM using images from camera phones to operate in small indoor environments. PTAM uses Feature from Accelerated Segment Test (FAST) corner points [60] as features and employs similar methods as described in Section 1.2.3 for self-localization and mapping. PTAM has a bootstrapping problem related to manual initialization from a monocular camera. It fails to produce reliable camera pose estimations in multiple scenarios, such as abrupt camera motion, sudden rotation, low-textured environments, etc. The authors of PTAM propose an extended version of PTAM in [61] to work on camera phones, but the proposed system has similar drawbacks as PTAM. Taihú Pire *et al.* present a stereo version of PTAM, S-PTAM [62], [63], where it follows a similar threaded structure, separating the time-constrained camera tracking task from high computational map building and refinement tasks. The stereo initialization process allows reconstructing a metric 3D map and bypasses the bootstrapping problem in initialization.

Raúl *et al.* present another VSLAM framework using Oriented Rotated Brief (ORB) point features [15], ORB-SLAM [64], highly influenced by PTAM and follows an automatic initialization based on a statistical approach to model selection between planar and non-planar scenes using homography or the fundamental matrix [13], respectively. Subsequently, Raúl *et al.* come up with a groundbreaking VSLAM architecture known as ORB-SLAM2 [15], designed for three types of cameras: monocular, stereo, and RGB-D. The accuracy and robustness in multiple scenarios, e.g., indoor-outdoor, small-large scale, multiple types of motion, etc., make this the most acceptable VSLAM architecture. ORB-SLAM2 works quite accurately in feature-enriched environments but struggles in texture-less environments, as shown in [16], and frequently loses the camera track in abrupt motion.

Pumarola *et al.* present a VSLAM architecture that uses points and lines together as features, PL-SLAM [65], which significantly increases the number of features and produces more accurate results in feature-enriched environments, but handling dual features increases the amount of data and the computation cost significantly and makes the system unreliable in the long run. Subsequently, Pumarola *et al.* also show the usage of PL-SLAM for self-localization in an aerial manipulation task in [66]. Maity *et al.* present a VSLAM architecture specially designed for low-textured environments, where the system uses points lying on edges. The system proposes a novel initialization technique using geometrical validation from a reconstructed map and a novel camera pose

recovery mechanism when the system experiences an unstable situation due to fewer matched features. The system also proposes a unique method for loop detection using structural properties of edges in the images. This system exhibits better accuracy in low-textured environments but is unable to sustain jerky and abrupt motion.

Pyojin *et al.* present a low-drift visual odometry algorithm for RGB-D images [67], where rotational motions and translational motions are separately estimated from points, lines, and planes features. The proposed system first concentrates on retrieving the drift-free rotational motion using both lines and planes by exploiting environmental regularities with an efficient  $SO(3)$ -manifold constrained mean shift algorithm. Afterwards, the translational motion is estimated through de-rotated reprojection error minimization. The system shows an exceptional improvement in pure rotational motion estimation.

The performance of feature-based VSLAM algorithms highly depends on the type of features used, and therefore, researchers use points, lines, and planes features in combinations. Humans usually track any landmark not as any of such features but rather watch or track objects as a whole; therefore, researchers put effort into adding semantic information in camera tracking. In this context, Yang and Scherer present CubeSLAM [68], where every visible object is represented with 3D bounding cuboids and tracked using this 3D cuboid object representation. This system incorporates dynamic objects into the estimation, therefore, able to segregate the static and dynamic part of the scene and shows a better SLAM estimation.

We have noticed that losing camera track is a pertinent problem for VSLAMs and it is handled with re-localization procedures (see Section 1.2.3). However, re-localization is a probabilistic procedure and therefore not guaranteed to be completed within a finite time period. Thus, it is not feasible to wait for re-localization after losing camera track in a real mission. In this context, recently Richard *et al.* proposed ORB-Atlas [69], a multi-map system for a single agent, in which it creates a new sub-map after tracking fails and merges multiple sub-maps afterward once map overlapping is found. The system is designed only for a single client, but it provides a possibility to further extend for multiple clients in a collaborative framework.

### 2.2.2 Direct Methods

We have discussed multiple feature-based VSLAMs and their properties, where we observed that feature-based VSLAMs utilize a set of KFs and feature points to construct a sparse map of the environment. This approach has enabled VSLAM to run in real-time on consumer-grade computers and mobile devices. We have also noticed a significant improvement in computer processing power and camera performance in the

last decade. Therefore, the desire to create a denser map of the environment seems tangible through Direct Photogrammetric SLAM or Direct SLAM. We focus on the evolution of direct VSLAM methods over the last decade and interesting trends that have emerged.

Silveira and Malis present one of the early works on direct methods in [70], where pixel intensities are used as the observation criterion, and the initialization is made without any assumptions about either the scene structure or the camera motion. The system also associates some geometric constraints, namely the cheirality and the rigidity, with the intensity variation and estimates the parameters using an efficient second-order approximation method.

DTAM [71] is another initial work on direct VSLAM presented by Richard *et al.*, where it processes all pixels to generate a dense map. DTAM heavily relies on a Graphics Processing Unit (GPU) to process such a vast amount of data in real-time. DTAM initializes with a stereo baseline, creates the initial map with stereo measurements, and establishes a camera motion model from camera tracking. The depth of each pixel on successive images is computed and optimized by minimizing the total depth energy. DTAM shows better performance in low-textured environments and is better suited for variable/auto-focus cameras and motion-blurred images.

Jakob *et al.* proposed the idea of Large Scale Direct SLAM, LSD-SLAM [72], where it looks for high-gradient regions of the scene, especially edges, and analyzes the pixel's intensity only on those regions. The main idea is that there is very little information to track between frames in low-gradient or uniform pixel areas for depth estimation. The depth initialization in LSD-SLAM with random values of high uncertainty by using inverse depth parametrization [73], and it further optimizes the depth based on the disparity computed on image pixels. The optimization does not converge with true depth in multiple scenarios for the noisy initialization and noisy photometric error computations.

Christian *et al.* propose an extension of VO with the introduction of Semi-direct VO, SVO [74], where it proposes to use a sparser map with the direct method and blurs the differences between feature-based and direct methods. SVO extracts feature points like a feature-based method but uses the direct approach to perform camera motion estimation on the tracked features. In addition, SVO uses BA as the optimization tool to refine the camera poses and map structure. The main advantage of SVO is that it operates near-constant time and runs at relatively high frame rates. The positional accuracy of SVO under fast and variable motion is better, but SVO only provides tracking and avoids loop closure or global map optimization features, making the system unreliable for long runs. Ruben *et al.* propose an upgraded version of SVO

using a point-line combination as the feature, PL-SVO [75], and have shown improved accuracy in low-textured environments.

Jakob *et al.* further extend their work towards direct SLAM to propose Direct Sparse Odometry, DSO [76], a direct method with a sparse map. DSO splits the entire image into multiple regions and chooses sampled pixels from all the regions with some intensity gradients for tracking. This pixel selection criterion ensures that tracked points lie across the images and produce better estimations. The initial solution is not a complete VSLAM method as the loop closure and global optimization were excluded, but a later version, LDSO [77], proposed by Gao *et al.*, includes loop closure and can create maps with relatively small drifts.

Yang and Scherer propose a direct monocular odometry [17] using point and line features to benefit from the advantages of both direct and feature-based methods. The proposed system recovers the camera pose by minimizing both the photometric and geometric errors to the matched edge in a probabilistic framework. The system shows better accuracy in low-textured environments, illumination changes, and fast motions.

## 2.3 VIO and VI-SLAM

VIO has entered the field of robotics as the fusion of visual and inertial cues becomes popular due to the complementary nature of these two sensing modalities. Existing visual-inertial fusion approaches can be categorized into two types: loosely-coupled systems and tightly-coupled systems. In loosely-coupled systems, IMU measurements and visual measurements are incorporated independently, and the final camera pose is estimated by combining both estimations. We refer to [78] as an example of a loosely-coupled system. In contrast, a tightly-coupled system jointly estimates all sensor states and optimizes them together. The IMU frequency is much higher than the frequency of the camera, and Forster *et al.* show a pre-integration method on IMU data in [79], which can further be integrated with KF-based VIO estimation, producing better estimation than loosely-coupled systems. Therefore, we focus only on tightly-coupled systems.

Leutenegger *et al.* propose an early tightly-coupled VIO system, OKVIS [22], where estimations from an IMU are tightly integrated with visual measurements, and an IMU error term is introduced along with the reprojection error in a fully probabilistic manner, optimized with a joint non-linear cost on a sliding window.

Tong *et al.* present a robust corner-feature-based visual-inertial SLAM, VINS-Mono [80]. VINS-MONO initializes with a vision-only SfM structure to estimate a graph of up-to-scale camera poses and feature positions. An IMU pre-integration



estimates the camera motion, and a loose alignment of camera motion with the reconstructed visual structure is performed afterward for initialization. The main idea behind this alignment is to match the up-to-scale visual structure with metric scale pre-integrated IMU measurements. VINS-MONO uses a tightly coupled VIO, re-localization, efficient global optimization, and exhibits superior performance against other state-of-the-art implementations. The authors of VINS-MONO extend their work to VINS-Fusion to support stereo cameras. The availability of the source as open-source and very precise accuracy on multiple open datasets makes VINS-MONO very famous among researchers.

He *et al.* propose an extended version of VINS-MONO with points and lines, PL-VIO [81], which is a tightly-coupled monocular visual-inertial odometry that exploits both point and line features. The line features provide more connected constraints and yield better estimation.

The requirement for semantic information for robotic navigation and manipulation is essential, and keeping this objective in mind, Rosinol *et al.* provide an open-sourced library for real-time metric-semantic VI-SLAM, Kimera [82], where the library performs mesh reconstruction and semantic labeling in 3D in the form of a 3D dynamic scene graph [83].

Very recently, Campos *et al.* present ORB-SLAM3 [84], a visual-inertial version of ORB-SLAM2 [15] with multi-map support. ORB-SLAM3 is also a tightly-coupled system that provides multi-map support, meaning it reinitializes from the beginning whenever it finds the estimated camera poses are not reliable and stores the old map for later use. It tries to fuse older maps with the current map whenever the robot visits any old locations. ORB-SLAM3 shows state-of-the-art accuracy on various open datasets.

A detailed survey on all variants of VSLAMs is presented in [19], [20], and we can draw the following conclusions from the above study and the survey. Direct VSLAMs with semi-dense or dense map generation are still computationally heavy for standard computers; therefore, researchers focus on sparse map reconstruction using direct methods. Feature-based VSLAMs provide great accuracy but are limited to only feature-rich environments and perform poorly in low-textured environments. Therefore, researchers either use a combination of multiple features or some special optimization technique to overcome the problem and provide a better estimation in low-textured environments. Both feature-based and direct VSLAMs are not capable enough to sustain jerky or abrupt motions. On the other side, VI-SLAMs are more capable of handling jerky and abrupt motions. The accuracy of VI-SLAMs is established to be better than VSLAMs on multiple open datasets. Existing VI-SLAMs assume that the IMU measurements



always provide the correct motion model, but in a real scenario, this is not always true and depends on the quality of the IMU.

## 2.4 Collaborative VSLAMs and VI-SLAMs

We have discussed the applicability and basic idea of collaborative VSLAM in Section 1.2.5, where we also have explored the two categories of collaborative VSLAMs: distributed and centralized. Researchers predominantly opt for feature-based VSLAMs in collaborative frameworks due to accurate information sharing, as explained in [21]. Therefore, we restrict our discussion to feature-based collaborative VSLAMs and VI-SLAMs.

### 2.4.1 Distributed Collaborative Methods

Various collaborative frameworks are presented in [85]–[87], which utilize the global fused map to guide the trajectory estimation of UAVs but refrain from sharing the global map with the UAVs. Consequently, these systems are limited to 3D scene reconstruction. Choudhary *et al.* [88] propose an object-based distributed SLAM system where all agents exchange information directly among each other and perform all information fusion on-board, without having a central instance. Relative localization is based on commonly observed pre-trained objects. Recent distributed SLAM systems are proposed in [27], [29]–[31], focusing on different aspects (e.g., decentralized place recognition, map overlap identification, efficient distributed loop closure, data exchange, robustness, etc.) of decentralized collaborative SLAM. However, challenges with these distributed systems include the necessity for participating robots to be equipped with high computational processing, assurance of data consistency, and the avoidance of double-counting of information.

### 2.4.2 Centralized Collaborative Methods

One of the early works on centralized collaborative VSLAM is CoSLAM [89], where multiple cameras can participate in building a map collaboratively using vision-only SLAM algorithms. CoSLAM initializes all cameras by viewing a common scene and segregates the dynamic foreground from the static background. CoSLAM clusters cameras into multiple groups based on their view overlap. One of the main disadvantages of CoSLAM is that all cameras are synchronized by observing the same scene at initialization, which is not realistic in a big mission. The proposed architecture of CoSLAM has become noncompetitive after the introduction of VI-SLAMs.

Another early work on centralized collaborative VSLAM is proposed by Forster *et al.* [90], where the system is designed only for collaborative MAVs based on the SfM pipeline. The MAVs stream their poses and feature points from the selected KFs to a central ground control station. The ground control station performs map merging once an overlapped region is found. The ground control station never shares optimized information with the MAVs, and therefore, the MAVs are unable to benefit from the updated information.

Riazuelo *et al.* propose C<sup>2</sup>TAM [91], based on a client-server model, where each client uses a distributed framework by applying the expensive map optimization as a service in the cloud. Each client is designed to run PTAM [59] for their self-localization. The server performs the map merging task and periodically sends back the optimized complete map to every agent. The system is communication-heavy for a large map in widely distributed areas because of repeated transmission of the complete map to every client.

Deutsch *et al.* present a collaborative framework [92] that allows participating agents to use different monocular SLAMs as long as the agents provide a pose graph, where every node is associated with an image, and the graph is in absolute scale. The server performs the merging of pose graphs, which is purely visual-based, and each agent is informed only of updates to its local pose graph and is unaware of sub-maps from other agents.

Schmuck and Chli propose CCM-SLAM [21], a client-server-based collaborative SLAM framework where each client runs monocular VO on a local map with a fixed number of KFs and offloads the older map to a centralized server. The server maintains maps of all clients and performs all high computational tasks, e.g., map merging, loop closure, global optimization, etc. The proposed design of CCM-SLAM distributes the low and high computational tasks to support less-powered agents to participate and benefit. However, the main disadvantage is that the clients offload the older maps to the server and are unable to re-localize with older maps after losing the camera track due to any abrupt or jerky motion, which is evident in a VSLAM scenario.

Recently, Ouyang *et al.* present a collaborative framework [93], which uses a similar design to CCM-SLAM for UGVs to carry only monocular or RGB-D cameras. The presented system supports only UGV motions, limiting the system for a distributed mission with heterogeneous robots. The proposed system claims to fuse maps between a monocular camera (without metric scale) and an RGB-D camera (with scale), but the literature does not provide the mechanism of such fusion. The proposed system considers each camera as an independent client in a scenario where a single agent carries multiple cameras, making the system more computationally intensive.

Marco *et al.* propose the VIO version of CCM-SLAM, CVI-SLAM [26], for only monocular cameras and use pre-integrated IMU measurements for the optimization of KF poses. The system shows a SoA accuracy for monocular cameras on multiple open datasets. Jialing *et al.* present a collaborative visual-inertial SLAM [94] using monocular cameras for an augmented-reality application in which multiple users interact with their smartphones. The system is designed as a client-server architecture that models the maps as deformable maps, and all the sub-maps in the fused maps are optimized independently to solve the problem of common map distortion. Patrik *et al.* recently present a visual inertial SLAM for centralized collaboration, COVINS [28], which is also designed as a client-server architecture and given special focus to support as many numbers of agents that can run in real-time and shows the system is capable of incorporating twelve agents jointly. All these proposed visual inertial SLAMs show SoA accuracy on open datasets equipped with good IMU sensors. However, no system is adaptable enough to deal with noisy sensor measurements.



## Chapter 3

# Obstacle Detection and Tracking Using Depth Image

### 3.1 Introduction

We have examined the fundamental concept of obstacle detection and tracking, recognizing its significance in autonomous navigation, followed by an extensive literature survey. The literature survey encompasses an exploration of the constraints found in current approaches and the challenges associated with obstacle detection and tracking in dynamic and cluttered environments. Achieving accurate and less computation-intensive obstacle detection and processing is crucial for real-time execution. However, current approaches still fall short in addressing all the characteristics of dynamic obstacles. In this chapter, we introduce a dynamic obstacle detection system based on the research outlined in [2]. This system is designed to run on board any autonomous system, including UGVs and MAVs, aiming to mitigate issues such as tracking multiple dynamic obstacles with minimal computations and handling dynamic obstacles with varying shapes and sizes, as discussed in Section 1.2.1. Our system employs u-depth and v-depth maps for obstacle detection, and we present a novel algorithm for tracking obstacles in subsequent frames. The proposed system estimates relative velocities between the camera and obstacles, transforming them into the fixed world coordinate frame using self-localization. Utilizing an RGB-D camera and an IMU as sensors, each capturing data in their respective coordinate frames (denoted as  $C$  for the camera and  $B$  for the body/base\_link of the IMU in this chapter), our system ensures fixed transformations between these frames. The fixed transformation from the camera coordinate frame to the base\_link coordinate frame is denoted as  ${}^B\mathbf{T}_C$ , with  $W$  representing the world coordinate frame.

The rest of this chapter is organized as follows. The architecture of the proposed system and our contributions are described in Section 3.2. A series of evaluations

and real-world tests is presented in Section 3.3. Finally, the conclusions are drawn in Section 3.4.

## 3.2 System Architecture

The implementation of the proposed system has two main modules, which are shown in Fig. 3.1, (i) Localization module; and (ii) Obstacle detection and tracking module. The

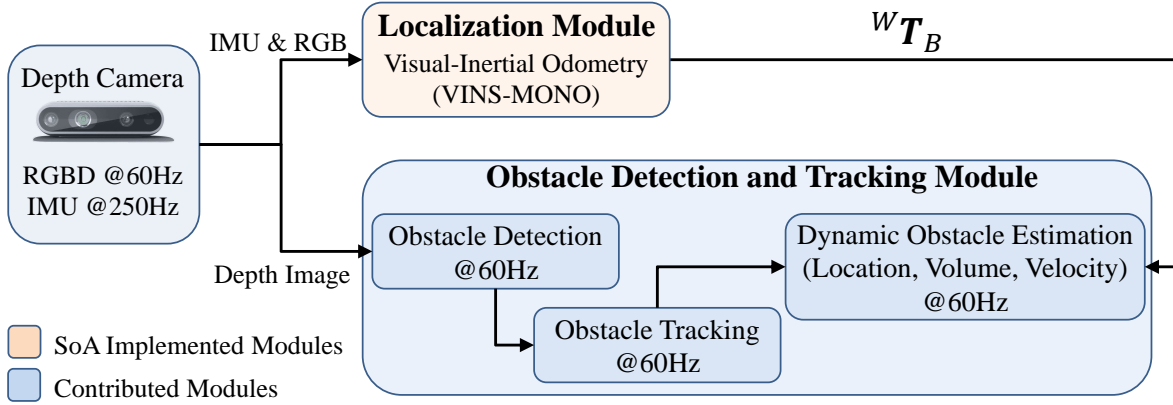


FIGURE 3.1: Block diagram of the proposed system for robust dynamic obstacle detection and tracking using RGB-D camera sensor data.

proposed system can be considered the perception module of any autonomous vehicle, where the primary responsibility of the perception module is to perceive the environment for autonomous navigation. We use the state-of-the-art visual-inertial system VINS-MONO [80] for self-localization. The obstacle detection module receives depth images from an RGB-D sensor, such as the Intel RealSense D435i [43] or Microsoft Kinect [42]. The proposed system assumes the depth images are rectified, so that an RGB and its corresponding depth images must have one to one pixel mapping. We annotated detected obstacles using our proposed method on multiple RGB images in this chapter, where those RGB images are used only for better visualization because the obstacle detection and tracking module processes only depth images. Finally, the system estimates the velocities of all dynamic obstacles in the world coordinate frame  $W$ .

### 3.2.1 Self Localization

The proposed obstacle detection module has the requirements of identifying an obstacle's state (i.e., static or dynamic) and estimating the velocities of all dynamic obstacles. The system is intended for a mobile robot, and therefore, a static obstacle shows a displacement in consecutive images when the robot is in motion. This means that estimating the motion of any obstacle from the camera coordinate frame is

not possible, as the camera coordinate frame moves along with the robot. Therefore, we require a fixed coordinate frame for estimating the motions of all obstacles. Any localization module produces robot poses at every instance from a fixed coordinate frame: the world coordinate frame  $W$ . We can transform the estimated location of any obstacle from the camera coordinate frame to the world coordinate frame with a coordinate transformation through the robot poses. VINS-MONO [80] stands out as a SoA localization system, utilizing monocular images and IMU data to estimate robot poses in the fixed world coordinate frame, denoted as  $W$ . VINS-MONO is open-source software and produces acceptable outcomes in many open sequences, which led us to select VINS-MONO as the localization module in our framework. We refer to [80] for a detailed description of VINS-MONO. If we have a robot pose as  ${}^W\mathbf{T}_B$ , then  ${}^W\mathbf{T}_B ({}^B\mathbf{T}_C)$  is the transformation from the camera coordinate frame to the world coordinate frame.

### 3.2.2 Obstacle Detection

In this section, we describe our obstacle detection component of the proposed system. The obstacle detection module receives depth images as input from the input sensor. VbOAD [4] uses a column-wise histogram representation of the depth image called the u-depth map [34] for obstacle detection and its dimension estimation. The approach in VbOAD has a serious limitation in obstacle height estimation, where empty space is represented as being occupied in a specific situation. There are certain limitations of VbOAD, which are discussed below. In the proposed method to overcome these limitations, we use two-step depth map representations, where we use the u-depth map representation first. Afterward, we use a restricted row-wise histogram representation of the depth image called a restricted v-depth map.

#### 3.2.2.1 Depth Map Processing

The objective of depth map processing is to identify obstacles and estimate the positions and dimensions of those obstacles. U-depth map is a representation of depth values, where only the locations of obstacles becomes bright horizontal lines. Identification of these bright horizontal lines from a dark background is quite easy, therefore, detection and estimation of obstacle from u-depth map is also easy and quick. Therefore, we first discuss the computation of the u-depth map.

The u-depth map is a column-wise histogram representing the depth values of the depth image. We use row-column order notation to represent any matrix, 2D coordinates, image resolution, or rectangle size in the rest of the thesis. Let us consider a depth image  $\mathbf{IMGD}_{h_I \times w_I}$  with a 2D coordinate frame  $I$ , where  $h_I$  is the height and

$w_I$  is the width. The total number of histogram bins is  $n$ , and the sensor depth range is  $[min_d, max_d]$ . Then, the range of each bin is  $\frac{(max_d - min_d)}{n}$ , and the corresponding u-depth map is  $\mathbf{uIMGD}_{n \times w_I}$ . The position of any obstacle is estimated from this column-wise histogram, and for better understanding, the histogram is considered a gray image in which white patches represent obstacles. The details of the u-depth map calculation are summarized in Algorithm 1: U-depth-map( ).

---

**Algorithm 1** : U-depth-map( )

---

**Input:** depth image  $\mathbf{IMGD}_{h_I \times w_I}$ , number of bins  $n$ , sensor depth range  $[min_d, max_d]$

**Output:** u-depth map  $\mathbf{uIMGD}_{n \times w_I}$

---

```

1: Initialization :  $\mathbf{uIMGD} \leftarrow 0$ ,  $Scale = \frac{255}{h_I}$ 
2: for  $i = 1$  to  $w_I$  do
3:   for  $j = 1$  to  $h_I$  do
4:     if  $(min_d \leq \mathbf{IMGD}(j, i) \leq max_d)$  then
5:        $Index = \lfloor \{\frac{(n-1)}{(max_d - min_d)} \times (\mathbf{IMGD}(j, i) - min_d)\} + 1 \rfloor$ 
6:        $\mathbf{uIMGD}(Index, i) = \mathbf{uIMGD}(Index, i) + 1$ 
7:     end if
8:   end for
9: end for
10:  $\mathbf{uIMGD} = \mathbf{uIMGD} \times Scale$ 
11: return  $\mathbf{uIMGD}$ 

```

---

The calculation of the u-depth map outlined above deviates from the approach presented in [38], where the calculation stems from a stereo disparity map. In the current method, specifically in line 10 of Algorithm 1: U-depth-map( ), the values of the u-depth map  $\mathbf{uIMGD}$  are normalized within the range  $[0, 255]$  for enhanced processing. In this scheme, the histogram bins are arranged in ascending order from top to bottom, signifying that a row  $i$  corresponds to a smaller depth than a row  $j$ , with  $i < j$ . Consequently, any closer obstacle contributes to the upper rows in the u-depth map, resulting in the depth bins associated with the obstacles appearing as white horizontal patches. Fig. 3.2 illustrates the relationship between obstacle positions and these white patch positions in a u-depth map, depicting two obstacles situated in different depth ranges. The dimensions of a white patch offer information about the corresponding obstacles position and size in the depth image, necessitating the segmentation of these white patches. Basic image processing operations are employed to achieve this segmentation. Initially, the u-depth map is converted into a binary image, making the white patches more noticeable and easier to identify. Binary thresholding is performed, where the threshold value of each row ( $T_{row}$ ) is determined using the Equ. 3.1.



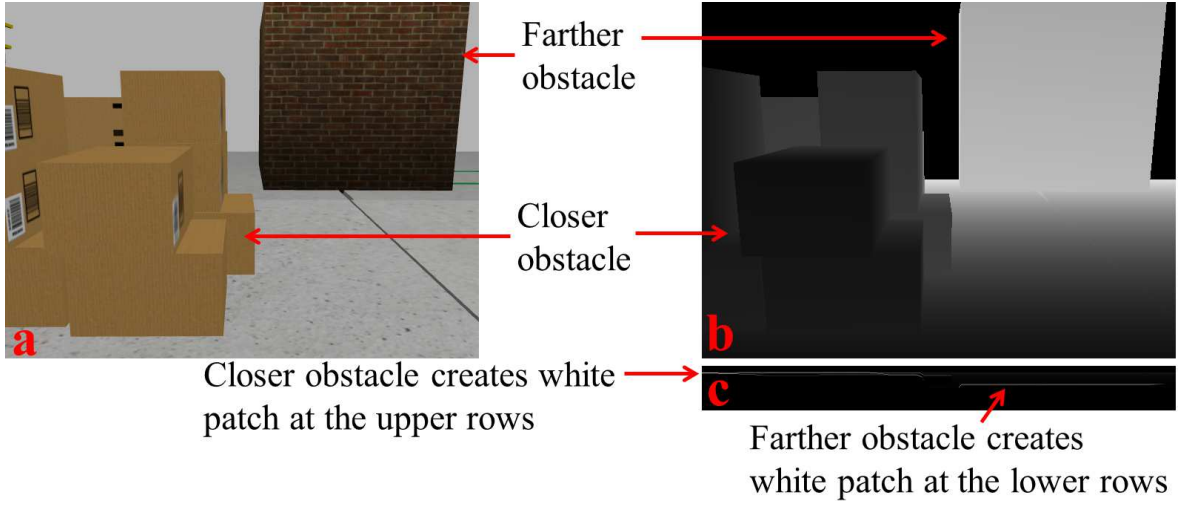


FIGURE 3.2: Illustration of obstacles in the u-depth map: (a) A sample RGB snapshot in Gazebo [95], (b) Corresponding depth image with lower intensity representing smaller depths, and (c) Corresponding u-depth map, where the white patch at a smaller row index indicates the closer obstacle.

$$T_{row} = \Lambda + \wp \times \mathbf{uIMGD}_{row} \mid \mathbf{uIMGD}_{row} \in [1, 2, 3, \dots, n] \quad (3.1)$$

where  $\Lambda$  and  $\wp$  are constants, experimentally determined as  $\Lambda = 18.96$ ,  $\wp = 2.04$ . Equ. 3.1 shows that each row of u-depth map ( $\mathbf{uIMGD}$ ) is assigned a different threshold value, which is proportional to the row number of the u-depth map. This approach gives greater emphasis to closer obstacles, even if they are smaller in size. This dynamic binary thresholding enhances obstacle detection and facilitates accurate dimension estimation, as detailed in Section 3.3.4 and Section 3.3.6, respectively. Due to the inherent noise in the depth estimation of RGB-D sensors like the Intel RealSense D435i, the u-depth map may contain some noise, causing the white patches to be discontinuous. In such cases, a closing operation [96] with a  $(3 \times 5)$  structuring element is applied to these white patches, producing continuous patches efficiently. Figure 3.3 illustrates a u-depth map and its corresponding thresholded u-depth map after the closing operation. Subsequently, individual components are obtained using the component analysis technique [97], with each component segmented within a bounding box. Henceforth in this chapter, the term ‘u-depth map’ is used to represent the ‘thresholded and closed binary u-depth map’. At this point, we calculate the dimensions of the detected components.



FIGURE 3.3: Estimating obstacle width from u-depth map: (a) Snapshot from the OpenLORIS-Scene market data sequence [58], (b) Corresponding u-depth map, (c) Corresponding thresholded u-depth map after a closing operation.

### 3.2.2.2 Dimension Estimation

In the previous section, we saw the component analysis technique segment each obstacle in a u-depth map within a bounding box. Now, we present the method to calculate the dimensions of any obstacle in the 3D world coordinates. First, we obtain the position and dimensions on the depth image (i.e., 2D positions and dimensions), and then we extend the method to find the positions and dimensions in 3D. Let us consider a sample white patch in a u-depth map with a bounding box as shown in Fig. 3.4, where the top-left corner of the bounding box is  $(u_t, u_l)$  and the right-bottom corner is  $(u_t + u_h, u_l + u_w)$ . We can find the depth range  $[d_{min}, d_{max}]$  for the corresponding



FIGURE 3.4: A sample thresholded u-depth map with a segmented obstacle and the corresponding bounding box.

obstacle in the camera coordinate frame ( $C$ ) from the row indexes  $u_t$  and  $(u_t + u_h)$  using Equ. (3.2):

$$\begin{aligned} d_{min} &= (u_t - 1) \times \frac{(max_d - min_d)}{(n - 1)} + min_d \\ d_{max} &= (u_t + u_h - 1) \times \frac{(max_d - min_d)}{(n - 1)} + min_d \end{aligned} \quad (3.2)$$

where,  $n$  is the number of histogram bins and  $[min_d, max_d]$  is the sensor depth range. The width of the corresponding obstacle on the depth image will be the same as the width of the white patch on the u-depth map because the u-depth map contains a column-wise histogram. Therefore, the column location does not change from the depth image to the u-depth map calculation. The relation between the width of a white patch and its corresponding obstacle's width is pictorially shown with red arrows in Fig. 3.3. In order to estimate the height of the obstacle, VbOAD [4] collects the pixels from the depth image that contain depth values within the range  $[d_{min}, d_{max}]$  and within the column range  $[u_l, u_l + u_w]$ . It then calculates the minimum and maximum rows based on the positions of those selected pixels. Now, let us consider a scenario where two or more obstacles are present within the column range  $[u_l, u_l + u_w]$  and within the depth range  $[d_{min}, d_{max}]$ . Then, all those obstacles will contribute to the same depth bins, and a single white patch in the u-depth map will be obtained. Therefore, VbOAD combines the height of all obstacles present within the same column range and the same depth range. Fig. 3.5 shows one such example where the height of a single obstacle becomes the combined height of multiple obstacles. Fig. 3.5(a) shows a person standing at a

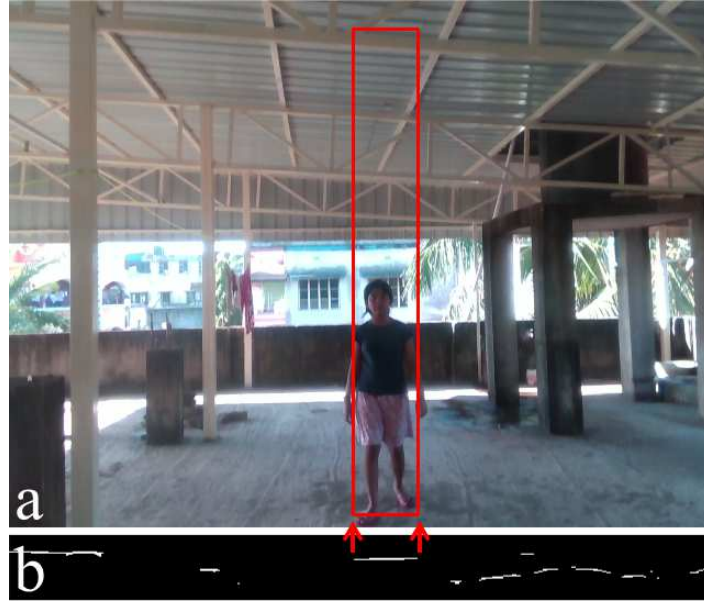


FIGURE 3.5: Height estimation of an obstacle using VbOAD [4]: (a) A snapshot captured with D435i along with the estimated height of the obstacle (the person), (b) Corresponding thresholded u-depth map.

certain distance from the camera, and a portion of the roof is also visible at the same distance from the camera. Fig. 3.5(b) is the corresponding u-depth map, and the red arrows show the width of the person estimated from the white patch of the u-depth map. The estimated height of the person, as in VbOAD, is shown with a red rectangle

in Fig. 3.5(a). The empty space between the person's head and the roof becomes a part of the person's height.

Like, u-depth map, we can take a row-wise histogram of the depth image, and then we should get a vertical white patch for an obstacle, and the height of the obstacle will equal the height of the corresponding vertical patch. The depth of the ground plane grows from bottom to top. Therefore, when we take a row-wise histogram, the ground plane becomes more prominent, and the identification of an obstacle's height is not straightforward. Labayrade *et al.* [34], [35] show a row-wise histogram representation of the depth image, a v-depth map, that helps in ground plane segmentation for any UGV. Fig. 3.6 shows the v-depth map pictorially, where Fig. 3.6(b) shows the v-depth map of Fig. 3.6(a). Fig. 3.6(c) is the corresponding thresholded v-depth map, where



FIGURE 3.6: Obstacle height calculation using the proposed restricted v-depth map: (a) The same snapshot from Fig. 3.3(a), (b) Corresponding v-depth map, (c) Corresponding thresholded v-depth map, (d) Corresponding proposed thresholded restricted v-depth map.

the curved white patch is the representation of the ground plane. The obstacle's height is not understandable from a v-depth map. This is the major drawback of estimating the height of an obstacle from a v-depth map.

In the present method, instead of considering the entire image, we consider only the column ranges of that particular obstacle. As a result, if we only consider the column range  $[u_l, u_l + u_w]$  and the depth range  $[d_{min}, d_{max}]$  in v-depth map formation, then obstacle height estimation becomes simple and accurate because we only use the pixels that lie on the obstacle, and the corresponding white patch only becomes visible on the thresholded image, as shown in Fig. 3.6(d). We call this map as a restricted v-depth map and use it only for accurate height estimation. We compute binary thresholding on the restricted v-depth map for a similar reason, as with the u-depth map. The details of the restricted v-depth map with binary thresholding are summarized in Algorithm 2: Restricted V-depth-map( ).

---

**Algorithm 2** : Restricted V-depth-map( )

---

**Input:** depth image  $\mathbf{IMGD}_{h_I \times w_I}$ , number of bins  $n$ , column range  $[u_l, u_l + u_w]$ , sensor depth range  $[min_d, max_d]$ , obstacle's depth range  $[d_{min}, d_{max}]$ **Output:** Thresholded v-depth map  $\mathbf{vIMGD}_{h_I \times n}$ 

```

1: Initialization :  $\mathbf{vIMGD} \leftarrow 0$ 
2: for  $i = 1$  to  $h_I$  do
3:   for  $j = u_l$  to  $u_l + u_w$  do
4:     if  $(d_{min} \leq \mathbf{IMGD}(i, j) \leq d_{max})$  then
5:        $Index = \lfloor \{ \frac{(n-1)}{(max_d - min_d)} \times (\mathbf{IMGD}(i, j) - min_d) \} + 1 \rfloor$ 
6:        $\mathbf{vIMGD}(i, Index) = 1$ 
7:     end if
8:   end for
9: end for
10:  $\mathbf{vIMGD} = \mathbf{vIMGD} \times 255$ 
11: return  $\mathbf{vIMGD}$ 

```

---

One important aspect of the above calculation is selecting the threshold value to be one, because the selected column range and depth range allow us to select only those pixels that lie on an obstacle. We do not want to lose a single pixel that lies on the obstacle. We also perform a closing operation with a  $(5 \times 3)$  structuring element to generate continuous vertical white patches. Then, we apply component analysis, such as a u-depth map, to find the bounding boxes. The relationship between the height of the white patch and its corresponding obstacle's height is pictorially shown with red arrows in Fig. 3.6. Hereafter, in this chapter, we use the term 'restricted v-depth map' to represent the 'thresholded and closed binary restricted v-depth map'.

The restricted v-depth map produces multiple patches if multiple obstacles exist in the same depth range. Fig. 3.7 shows the height correction of the previous example, where the height of the girl come as the combined height of the girl and the roof, as pictorially shown in Fig. 3.5. Fig. 3.7(a) shows the height estimation using the proposed restricted v-depth map, and Fig. 3.7(b) shows the thresholded restricted v-depth map where two different obstacles, the person and a small portion of the roof, are visible. Fig. 3.7(c) is the magnified view of the portion, where the roof is present in the restricted v-depth map. The estimated height of the person using the proposed restricted v-depth map is shown with a green rectangle in Fig. 3.7(a), and the arrows show the height of the green rectangle derived from the white patches of the restricted v-depth map.

Now, let us consider a sample white patch on a restricted v-depth map with a bounding box as shown in Fig. 3.8, where the top-left corner of the bounding box





FIGURE 3.7: Estimation of the height of an obstacle using the proposed restricted v-depth map: (a) The same image used in Fig. 3.5 and the result of the proposed technique, (b) Result of the thresholded restricted v-depth map, (c) The magnified view of the thresholded restricted v-depth map shows the portion of the roof as a separate obstacle.

is  $(v_t, v_l)$  and the right-bottom corner is  $(v_t + v_h, v_l + v_w)$ . The corresponding obstacle

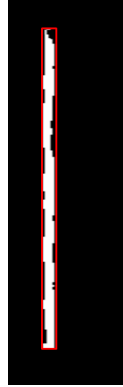


FIGURE 3.8: A sample magnified, thresholded restricted v-depth map with a segmented obstacle within a bounding box.

on the depth image must be of the same height as the white patch on the restricted v-depth map. Now, we have the bounding box using the u-depth map represented by coordinates  $(u_t, u_l)$  and  $(u_t + u_h, u_l + u_w)$ , and we have the bounding box using the restricted v-depth map with coordinates  $(v_t, v_l)$  and  $(v_t + v_h, v_l + v_w)$ . The dimensions of these bounding boxes allow us to calculate the dimensions of the corresponding bounding box on the depth image. Therefore, the bounding box for the current obstacle on the depth image is represented by the coordinates  $(v_t, u_l)$  and  $(v_t + v_h, u_l + u_w)$ .

We transform this two-dimensional rectangle of the depth image into the three-dimensional camera coordinate frame using the relationship between the image coordinate  $I$  and the camera coordinate frame  $C$ . We have provided a quick description about this relation with perspective projection in Appendix A.1 and refer to [13] for

a detailed description of this relationship. Furthermore, we assume all obstacles are of a rectangular parallelepiped in shape with dimensions ( ${}^Cdim_h, {}^Cdim_w, {}^Cdim_d$ ) and a position ( ${}^CP_x, {}^CP_y, {}^CP_z$ ). Now, let us consider the most simplistic and popular camera model (i.e., the pinhole camera model as described in Appendix A.1), which we use to transform the measurements from 2D to 3D. Here, we take the simplistic form of the intrinsic camera matrix and assume  $f_x$  and  $f_y$  are the focal lengths in the image's horizontal and vertical directions, respectively, and  $(c_y, c_x)$  is the principal point. We use the concept as shown in Equ. (A.1) and formulate the size calculation of the rectangular parallelepiped-shaped obstacle as presented in Equ. (3.3)

$$\begin{aligned}
{}^Cdim_w &= \frac{((u_l + u_w) - c_x)d_{max}}{f_x} - \frac{(u_l - c_x)d_{max}}{f_x} \\
&= \frac{u_w \times d_{max}}{f_x} \\
{}^Cdim_h &= \frac{((v_t + v_h) - c_y)d_{max}}{f_y} - \frac{(v_t - c_y)d_{max}}{f_y} \\
&= \frac{v_h \times d_{max}}{f_y} \\
{}^Cdim_d &= (d_{max} - d_{min})
\end{aligned} \tag{3.3}$$

Similarly, the expression to calculate the centroid of the rectangular parallelepiped is given in Equ. (3.4):

$$\begin{aligned}
{}^CP_x &= \frac{((u_l + \frac{u_w}{2}) - c_x) \times (d_{min} + \frac{{}^Cdim_d}{2})}{f_x} \\
&= \frac{(2u_l + u_w - 2c_x) \times (2d_{min} + {}^Cdim_d)}{4f_x} \\
{}^CP_y &= \frac{((v_t + \frac{v_h}{2}) - c_y) \times (d_{min} + \frac{{}^Cdim_d}{2})}{f_y} \\
&= \frac{(2v_t + v_h - 2c_y) \times (2d_{min} + {}^Cdim_d)}{4f_y} \\
{}^CP_z &= d_{min} + \frac{{}^Cdim_d}{2}
\end{aligned} \tag{3.4}$$

Let us assume  ${}^B\mathbf{T}_C$  is the transformation from the camera coordinate frame to the body coordinate frame.  ${}^B\mathbf{T}_C$  is a fixed transformation for any robot, and the estimation of  ${}^B\mathbf{T}_C$  is performed offline. We refer to [98] for details on the estimation of  ${}^B\mathbf{T}_C$ . Let us also assume  ${}^W\mathbf{T}_B$  (see Fig. 3.1) is the transformation from the body coordinate frame to the fixed world coordinate frame at any instance, that is the robot pose.

The estimation of  ${}^W\mathbf{T}_B$  comes from the self-localization module, VINS-MONO [80], as shown in Fig. 3.1. We use the transformation  ${}^W\mathbf{T}_B ({}^B\mathbf{T}_C)$  to transform the rectangular parallelepiped's location and size from the camera coordinate frame  $C$  to the fixed world coordinate frame  $W$ . This world coordinate frame  $W$  helps to track the dynamic obstacles. Fig. 3.9 presents an example of a detected obstacle on the OpenLORIS-Scene market data sequence [58], where the dimensions of the obstacle are annotated in the world coordinate frame  $W$ .



FIGURE 3.9: (a) An RGB snapshot with detected obstacles in the OpenLORIS-Scene market data [58], (b) Corresponding Rviz [99] visualization with annotated dimensions.

### 3.2.3 Obstacle Tracking

Tracking associates the detected obstacles in subsequent images and helps predict their future positions within a predicted zone. The usual methods of tracking is either using some visual features [55] or using some probability function [4]. Here, we ignore visual features as these are computationally heavy, and the tracking time grows with the size of an obstacle in the image frame. We process a minimal number of pixels from the u-depth map to track obstacles in subsequent frames. One popular way to match two image segments is through Hu Moments [100] calculation, where two images are compared with their structural properties. However, the white patches do not have such good structural properties, and we discover in our multiple experiments that Hu-moment matching produces many false-positive results. Thus, we exclude Hu-moment matching and propose a suitable matching algorithm.

In this work, we propose a simple matching algorithm. First, we generate a signature for an obstacle using the u-depth map and then search for a similar signature in the neighboring region of the next frame. As discussed earlier, the u-depth map effectively contains an obstacles location, width, and height. On the other hand, a restricted v-depth map contains only height information. To incorporate the restricted



v-depth map into the obstacle signature, we would need to combine it with the u-depth map to retain all relevant information. However, this would increase the complexity of the obstacle signature. Since our goal is to create a fast and efficient tracking algorithm, we chose not to include patches from the restricted v-depth map, focusing instead on minimizing complexity. We observe that signatures with the u-depth map are capable of being tracked using our proposed tracking algorithm as described below. Another advantage of using a u-depth map is that if any obstacle moves parallel to the optical direction of the camera (i.e., the relative depth change is at its maximum), the corresponding position change in the u-depth map is minimal, and tracking works well. However, if any obstacle moves from left to right or vice versa, the position change of the obstacle in the u-depth map would be at the same rate as in the depth image, and the tracking algorithm is required to adapt to such movement.

Let's consider a hypothetical obstacle  $A$ , represented by a set of white pixels denoted as  $P_A$  in a u-depth map. For a visual explanation of our pixel selection, refer to Fig. 3.10, where the white pixels in Fig. 3.10(a) illustrate the selected pixels of  $P_A$ . Additionally, there exists another set of pixels,  $C_A \subset P_A$ , representing the contour

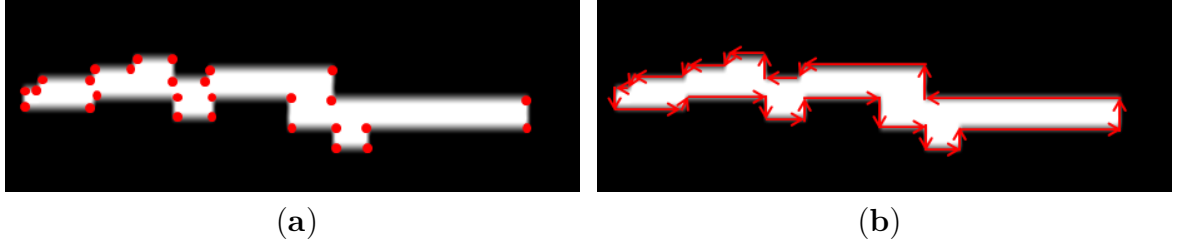


FIGURE 3.10: A sample pictorial view of the selected points in  $\Phi_A$  for obstacle  $A$ : (a) Red dots represent the points present in  $\Phi_A$ , (b) Vector formation using the points in  $\Phi_A$ .

of  $A$ . The contour is defined as an ordered set of pixels where two consecutive pixels are neighboring pixels. Now, we establish another set comprising the minimum number of ordered pixel points  $\Phi_A = \{\phi_1, \phi_2, \dots, \phi_k\} \subset C_A$ . This set is designed such that  $\phi_i \rightarrow \phi_{i+1} \in \Phi_A$  for  $i \in \{1, 2, \dots, k-1, k\}$  ( $\phi_{k+1} \equiv \phi_1$ ), connected with a vector  $\mathbf{l}_{A_i}$ , which must be entirely contained within  $C_A$ . The length of each vector should be maximized, as  $\Phi_A$  is formed with the minimum number of pixel points. The red dots in Fig. 3.10(a) represent the points of  $\Phi_A$ , where  $|\Phi_A|$  is significantly shortened from  $|P_A|$ . Fig. 3.10(b) shows the vectors that are formed with the points of  $\Phi_A$ . The directions of these vectors depend on the start and end point coordinates. Therefore, they can be in any direction, and Fig. 3.10(b) also shows that some small vectors are slanted. We compute the extreme left point,  $\Phi_{A_L}$ , and the extreme right point,  $\Phi_{A_R}$ , of  $\Phi_A$ . We also deduce the visibility of the obstacle  $Vis_A$  in terms of being fully visible or partially visible using the points  $\Phi_{A_L}$  and  $\Phi_{A_R}$ . We consider obstacle  $A$  to be partially visible

if the point  $\Phi_{A_L}$  touches the left edge or the point  $\Phi_{A_R}$  touches the right edge of the u-depth map. We consider  $\{\Phi_A, \Phi_{A_L}, \Phi_{A_R}, Vis_A\}$  to be the signature of obstacle  $A$ , where  $\Phi_A$  contains very few points but retains complete structural information and aids in fast execution. We also create a probable zone around the obstacle  $A$ . The probable zone selection is based on the maximum allowable relative speed of any obstacle, the FOV and frequency of the input RGB-D camera, and the sensor's operating depth range.

In the context of obstacle signature, the described method appears to be very similar to convex hull. While it is possible to represent an obstacle more efficiently using a convex hull with fewer points, this method often includes extra regions that do not belong to the actual obstacle. Consequently, convex hull representations for different obstacles with similar appearances may become indistinguishable. In the case of dynamic obstacles, whose shapes change over time, convex hull representations can vary drastically, which may lead to difficulties in tracking. Therefore, for simplicity, we chose to use underline representation as shown in Fig. 3.10 to ensure stable tracking.

To match with another obstacle  $B$  located within the probable zone of obstacle  $A$  in the subsequent depth image, we initiate the alignment of obstacle  $B$  with obstacle  $A$ . This alignment is determined by the visibilities of obstacles  $A$  and  $B$ . When both obstacles  $A$  and  $B$  are entirely visible, the alignment occurs at the nearest extreme points i.e., either  $\{\Phi_{A_L}, \Phi_{B_L}\}$  or  $\{\Phi_{A_R}, \Phi_{B_R}\}$ . In cases where either of the obstacles is partially visible to the left edge of the u-depth map, the alignment takes place at the right extreme points, specifically  $\{\Phi_{A_R}, \Phi_{B_R}\}$ , or vice versa. Equ. (3.5) illustrates the calculations for left and right alignment:

$$\begin{aligned} Align_{L_{\{A,B\}}} &= \Phi_{B_L} - \Phi_{A_L} \\ Align_{R_{\{A,B\}}} &= \Phi_{B_R} - \Phi_{A_R} \end{aligned} \quad (3.5)$$

We use either  $Align_{L_{\{A,B\}}}$  or  $Align_{R_{\{A,B\}}}$  to transform the signature of  $B$  based on the visibility of obstacle  $B$ , and we name the selected alignment  $Align_{\{A,B\}}$ . We measure the dissimilarity between the signature of obstacle  $A$  and the transformed  $B$  using Equ. (3.6):

$$Difference_{\{A,B\}} = Dim_{\{A,B\}} + Pos_{\{A,B\}} + Length_{\{A,B\}} + Angle_{\{A,B\}} \quad (3.6)$$

where,  $Dim_{\{A,B\}}$ ,  $Pos_{\{A,B\}}$ ,  $Length_{\{A,B\}}$ , and  $Angle_{\{A,B\}}$  are the dissimilarity costs of the dimension, position, vector length, and vector direction, respectively, between obstacles  $A$  and  $B$ . The expression to calculate  $Dim_{\{A,B\}}$  is given in Equ. (3.7), and

the  $Pos_{\{A,B\}}$  is calculated using Equ. (3.8):

$$Dim_{\{A,B\}} = \frac{||(\Phi_{A_R} - \Phi_{A_L})|| - ||(\Phi_{B_R} - \Phi_{B_L})||}{||(\Phi_{A_R} - \Phi_{A_L})||} \quad (3.7)$$

$$Pos_{\{A,B\}} = \frac{||Align_{\{A,B\}}||}{Max_h} \quad (3.8)$$

where,  $Max_h$  is the maximum horizontal pixel displacement permitted for any obstacle. Equ. (3.9) shows the relation for calculating the  $Length_{\{A,B\}}$ :

$$Length_{\{A,B\}} = \frac{\frac{1}{|\Phi_A|} \sum_{i \in \Phi_A, j \in \Phi_B} |(\|\mathbf{l}_{A_i}\| - \|\mathbf{l}_{B_j}\|)|}{MaxLengthDiff_{\{A,B\}}} \quad (3.9)$$

where,  $\mathbf{l}_{A_i}$  is the vector formed with the  $i^{th}$  and  $(i+1)^{th}$  point in  $\Phi_A$  and  $\mathbf{l}_{B_j}$  is the vector formed with the  $j^{th}$  and  $(j+1)^{th}$  point in  $\Phi_B$ .  $MaxLengthDiff_{\{A,B\}}$  is the maximum allowable difference in the vector length between obstacles  $A$  and  $B$ . The  $Angle_{\{A,B\}}$  between two obstacles ( $A$  and  $B$ ) is computed using Equ. (3.10):

$$Angle_{\{A,B\}} = \frac{\frac{1}{|\Phi_A|} \sum_{i \in \Phi_A, j \in \Phi_B} (\mathbf{l}_{A_i} \odot \mathbf{l}_{B_j})}{MaxAngleDiff_{\{A,B\}}} \quad (3.10)$$

where,  $\odot$  calculates the angle between two vectors and  $MaxAngleDiff_{\{A,B\}}$  is the maximum allowable angle difference between the pair  $\{\mathbf{l}_{A_i}, \mathbf{l}_{B_j}\}$ .

We introduce additional considerations in estimating the overall dissimilarity cost between obstacles  $A$  and  $B$ , as expressed in Equ. (3.6). These considerations are rooted in the assumptions that obstacle  $B$  is located nearly in the same position as obstacle  $A$ , and the dimensions of  $A$  and  $B$  are almost identical. Consequently, certain closely matching terms on the right-hand side of Equ. (3.6) are set to zero. The conditions are listed below:

1. If at least one of the matching obstacle from obstacles  $A$  and  $B$  is fully visible, and only  $Dim_{\{A,B\}} < Th_{Dim_1}$ , then we consider  $Length_{\{A,B\}} = 0$  and  $Angle_{\{A,B\}} = 0$ . This is the case where obstacle  $A$  moves from its previous position, but the width of the obstacle matches closely even after complete visibility in one frame.
2. If at least one of the matching obstacle from obstacles  $A$  and  $B$  is fully visible, and only the horizontal component of  $Align_{\{A,B\}} < Th_{Align_1}$ , then we consider

$Pos_{\{A,B\}} = 0$ . This is the case where the obstacle  $A$  does not move much, but there is a width change due to partial visibility in one frame.

3. If at least one of the matching obstacle from obstacles  $A$  and  $B$  is fully visible,  $Dim_{\{A,B\}} < Th_{Dim_1}$ , and the horizontal component of  $Align_{\{A,B\}} < Th_{Align_1}$ , then  $Pos_{\{A,B\}} = 0$ ,  $Length_{\{A,B\}} = 0$ , and  $Angle_{\{A,B\}} = 0$ . This is the case where the obstacle  $A$  almost stays at its previous position, and the width of the obstacle closely matches even after full visibility in one frame.
4. If both obstacles  $A$  and  $B$  are partially visible,  $Dim_{\{A,B\}} < Th_{Dim_2}$ , and the horizontal component of  $Align_{\{A,B\}} < Th_{Align_2}$ , then we consider  $Length_{\{A,B\}} = 0$  and  $Angle_{\{A,B\}} = 0$ . This is the case where obstacle  $A$  almost stays at its previous position, and the widths of the obstacles closely match in partial visibilities.

We consider the signatures of obstacles  $A$  and  $B$  a match if the difference, denoted as  $Difference_{A,B}$ , falls below a specified threshold  $Th_{A,B}$ . In the event of multiple obstacles having a score below  $Th_{A,B}$ , we designate the obstacle with the lowest score as the matched one. Subsequent to establishing a signature match, we update the signature with the most recent information. This straightforward approach contributes to stable tracking with minimal processing time.

### 3.3 Experimental Results

For experimental purposes, we use an NVidia Jetson TX2 embedded computing board to implement and test the proposed method in C++ with the Robot Operating System (ROS) [101] environment. In experiment, we evaluate and analyze the performance of our proposed method using various datasets, such as indoor and outdoor scenes, multiple static and dynamic obstacles, and fast-moving obstacles.

#### 3.3.1 Datasets and Experimental Description

Multiple datasets that we use are broadly of five types, and the detailed configurations of datasets are presented in Table 3.1. The  $Set_1$  dataset of Table 3.1 is a self-captured simulated dataset recorded in the rosbag format that contains the continuous RGB-D images, IMU measurements, and ground-truth (GT) poses of all the robots. The  $Set_2$  and  $Set_3$  datasets are RGB-D open video sequences, which have the GT of object tracking. The  $Set_4$  dataset is a self-captured real dataset with continuous RGB-D images and IMU measurements recorded in the rosbag format. We captured multiple sequences of data under this dataset, where all of them are outdoors with direct and

TABLE 3.1: Configurations of all experimental datasets

Datasets		Type	Mounted on	Depth Sensor	Image Size and rate (Hz)	Description of Dynamic Obstacle
$Set_1$	Gazebo Simulation [95]	Indoor	Husky Robot [102]	Microsoft Kinect [42]	$480 \times 640$ 30	Single
$Set_2$	PTB [7]	Indoor	Fixed/Hand-held	Microsoft Kinect [42]	$480 \times 640$	Single, Multiple
$Set_3$	DepthTrack [52]	Indoor/outdoor	Fixed/Hand-held	Realsense 415 [43]	$360 \times 640$	Single, Multiple Small obstacle, Fast moving
$Set_4$	Self-Captured	Outdoor, Indirect Sunlight, Direct Sunlight	Hand-held	RealSense D435i [43]	$480 \times 848$ 60	Single, Multiple, Dynamic size and shape, Small obstacle, Fast moving
$Set_5$	OpenLORIS-Scene [58]	Indoor	Wheeled Robot	RealSense D435i [43]	$480 \times 848$ 30	Single, Multiple, Multiple heights

shaded sunlight on the obstacles, avoiding direct scorching sunlight to avoid depth corruption. The  $Set_5$  is another open dataset, which is available in rosbag format and contains continuous RGB-D images and IMU measurements, as well as the GT self-localization pose of the robot.

We categorize all of our experiments broadly in five groups, and Table 3.2 presents these categories to understand the utility of the experiments. We implement the complete system as outlined in the system description and assess its performance incrementally. The initial category aims to verify the system’s accuracy against the GT; hence, we select datasets that include GTs for this evaluation. The second and third categories are intended to validate the system’s performance with single and multiple non-rigid dynamic obstacles, respectively. The fourth category emphasizes the effectiveness of dynamic thresholding. While dynamic thresholding was applied in all previous examples, this specific experiment elucidates its true significance. In the fifth and final category, we evaluate the system’s performance with a very fast-moving obstacle.

In all of our experiments, the obstacle detection and tracking modules jointly take 0.4–0.9 ms for a single obstacle, irrespective of its size in the image frame. The average tracking time is about 4–5 ms, considering a maximum of five obstacles. Therefore, the system can perform in real time with five obstacles or more with a 60-Hz camera. The tracking algorithm confirms successful tracking of an obstacle with a maximum velocity of 2.5 m/s with a 60-Hz camera.

TABLE 3.2: Categories of all experiments

Experimental Objectives	Datasets	Availability of Ground Truth	Features of the selected data
Accuracy measurement with rigid obstacles	Gazebo Simulation, PTB, DepthTrack	Yes	Single and multiple obstacles
Tracking a non-rigid dynamic obstacle	Self-Captured	No	a human is walking, sitting, standing in outdoor
Tracking multiple non-rigid dynamic obstacles	OpenLORIS-Scene	Yes	Multiple humans are walking in groups as well as separately in a shopping center
	Self-Captured	No	Multiple humans are walking and crossing each other
Effect of our proposed dynamic thresholding with a specific example	OpenLORIS-Scene	Yes	Multiple humans are walking or standing in groups as well as separately in a shopping center
Tracking a very fast moving obstacle	Self-Captured	No	Human throwing basket ball

### 3.3.2 Parameter Tuning

We experimentally determined the values of the parameters introduced in Section 3.2.3, and each parameter maintains a constant value across all of our experiments. The parameter values are listed in Table 3.3; however, the descriptions are discussed below.

TABLE 3.3: Parameter Values of the Proposed Technique

Parameters	Resolution (480 × 640)	Resolution (480 × 848)
$Max_h$	40	49
$MaxLengthDiff_{\{A,B\}}$	30	
$MaxAngleDiff_{\{A,B\}}$	100°	
$Th_{Dim_1}$	0.1	
$Th_{Dim_2}$	0.25	
$Th_{Align_1}$	10	
$Th_{Align_2}$	20	
$Th_{A,B}$	1.5	

- $Max_h$  is the maximum allowable horizontal pixel displacement for any obstacle, related to the obstacle speed and quantified in the pixel domain. Therefore,  $Max_h$  must have a higher value in cases where the environment has fast-moving obstacles or the image width is large. Setting  $Max_h$  to a large number enables the system to deal with fast-moving obstacles, but it increases the processing time. We found that setting the value of  $Max_h$  to 6% – 7% of the image width

produces stable tracking. We set  $Max_h$  to 40 for a resolution of  $480 \times 640$  and 49 for a resolution of  $480 \times 848$ .

- $MaxLengthDiff_{\{A,B\}}$  and  $MaxAngleDiff_{\{A,B\}}$  represent the maximum allowable length and angle differences between two vectors. These values encode the changes in obstacle signatures in u-depth maps due to motion. Therefore, keeping lower values for these parameters enforces a hard constraint in signature matching between two obstacles and rejects those with a small signature mismatch. On the other hand, a large value for these parameters can increase the false-positive matching result. In our experimental evaluation, we set  $MaxLengthDiff_{\{A,B\}} = 30$  pixels and  $MaxAngleDiff_{\{A,B\}} = 100^\circ$ .
- We consider the pairs  $(Th_{Align_1}, Th_{Dim_1})$  and  $(Th_{Align_2}, Th_{Dim_2})$  as two-level threshold measurements. Each level considers the alignment distance and dimension dissimilarities between the two signatures, respectively. The first level indicates an exact match, and the second level indicates a good match between the two signatures. The two levels of thresholding are created to use some clues to match quickly between two closely related signatures, bypassing some processing. Therefore, this two-level thresholding concept decreases the total processing time and does not affect the accuracy of the system. The values are experimentally set to  $Th_{Dim_1} = 0.1$ ,  $Th_{Dim_2} = 0.25$ ,  $Th_{Align_1} = 10$ , and  $Th_{Align_2} = 20$ .
- Finally, we accept a signature as matched if the dissimilarity score between the signatures of obstacles is below the threshold  $Th_{A,B}$ , which is set experimentally to 1.5.

### 3.3.3 Experiment 1: Tracking Single Dynamic Obstacle and Comparison with Ground Truth

We evaluate the accuracy of the proposed algorithm using datasets that provide the GT location of all obstacles throughout the entire duration. This category of experiments utilizes both simulated and open datasets. Accordingly, we begin by discussing the environmental setup of the simulated dataset and then proceed to elaborate on the experiments conducted within that context. Subsequently, we provide a detailed description of the experiments conducted with the open datasets.

#### 3.3.3.1 Gazebo Environment and Experimental Set-up

We create a shop floor environment with two Husky robots [102], defining paths with a series of way points. The defined path ensures that Husky1 (in Fig. 3.11) observes



Husky2 in motion, allowing Husky1 to view multiple static obstacles and a dynamic obstacle (Husky2) during its motion. Fig. 3.11 presents some snapshots of the environment for a better understanding of their motion pictorially, where two Husky robots, Husky1 and Husky2, are encircled with white and yellow circles, respectively. Their camera viewing directions are shown with front arrow lines, the FOV is shown with colored triangles, and the travelled path is shown with curved white and yellow lines, respectively. The sizes and directions of the arrows and FOV triangles are merely indicative and do not correspond to the actual scales.

We compare the proposed obstacle detection and tracking algorithm with Boosting [103], KCF [48], MedianFlow [104], MIL [105], MOSSE [106], TLD [107], and VbOAD [4]. The first six algorithms use RGB image-based tracking, whereas VbOAD and the proposed method use depth image-based tracking. RGB image-based tracking is highly dependent on initialization because these algorithms segment the initialized region of interest as foreground and background and try to track the foreground in subsequent image frames. In real-life scenarios, obstacles typically enter the FOV from the exterior, initially appearing partially visible and gradually becoming fully visible as they move within the FOV. Consequently, we conduct two types of testing scenarios. The first one involves initializing the RGB-based tracking algorithms with full visibility of Husky2, while the second one initializes these algorithms with partial visibility of Husky2. Both VbOAD and the proposed algorithm successfully detect Husky2 as soon as it becomes partially visible in both test scenarios. For all RGB-based tracking, we utilize the implementation provided by OpenCV [108].

### 3.3.3.2 Initialization at Full Visibility of Husky2

In this experiment, all RGB image-based tracking algorithms, as mentioned before, initialize or detect Husky2 as a dynamic obstacle when it becomes completely visible, but VbOAD and the proposed algorithms detect Husky2 as soon as it is visible partially in the depth image. Fig. 3.12 shows the tracking and dimension estimation results of Husky2, where the detected position and dimensions are indicated with bounding boxes. The first column of all RGB image-based tracking algorithms in Fig. 3.12 represents the initialization frames, where Husky2 is completely visible. In this experiment, we find that the performances of KCF and MOSSE are better than other RGB-based tracking algorithms because these two algorithms could detect when Husky2 is exiting the FOV and stop tracking as soon as Husky2 becomes partially invisible. The Boosting algorithm produces erroneous results when Husky2 goes out of the FOV and becomes partially invisible, and it continues to produce erroneous tracking results even after Husky2 is totally out of the FOV. This is visible in the snapshots from the fourth



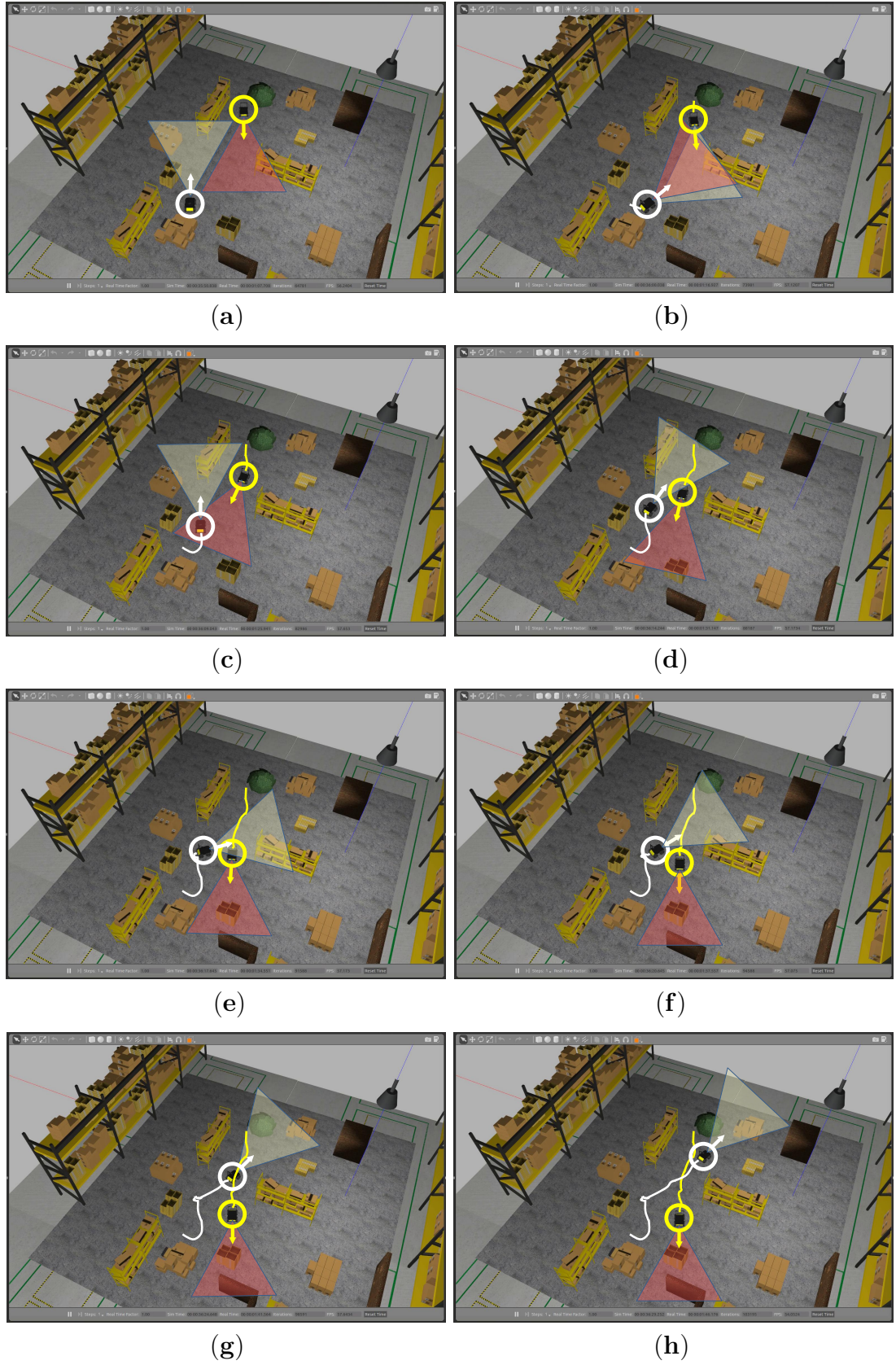


FIGURE 3.11: Snapshots of Gazebo [95] environments: (a–h) are time-sampled snapshots depicting the movements of Husky1 (white) and Husky2 (yellow).

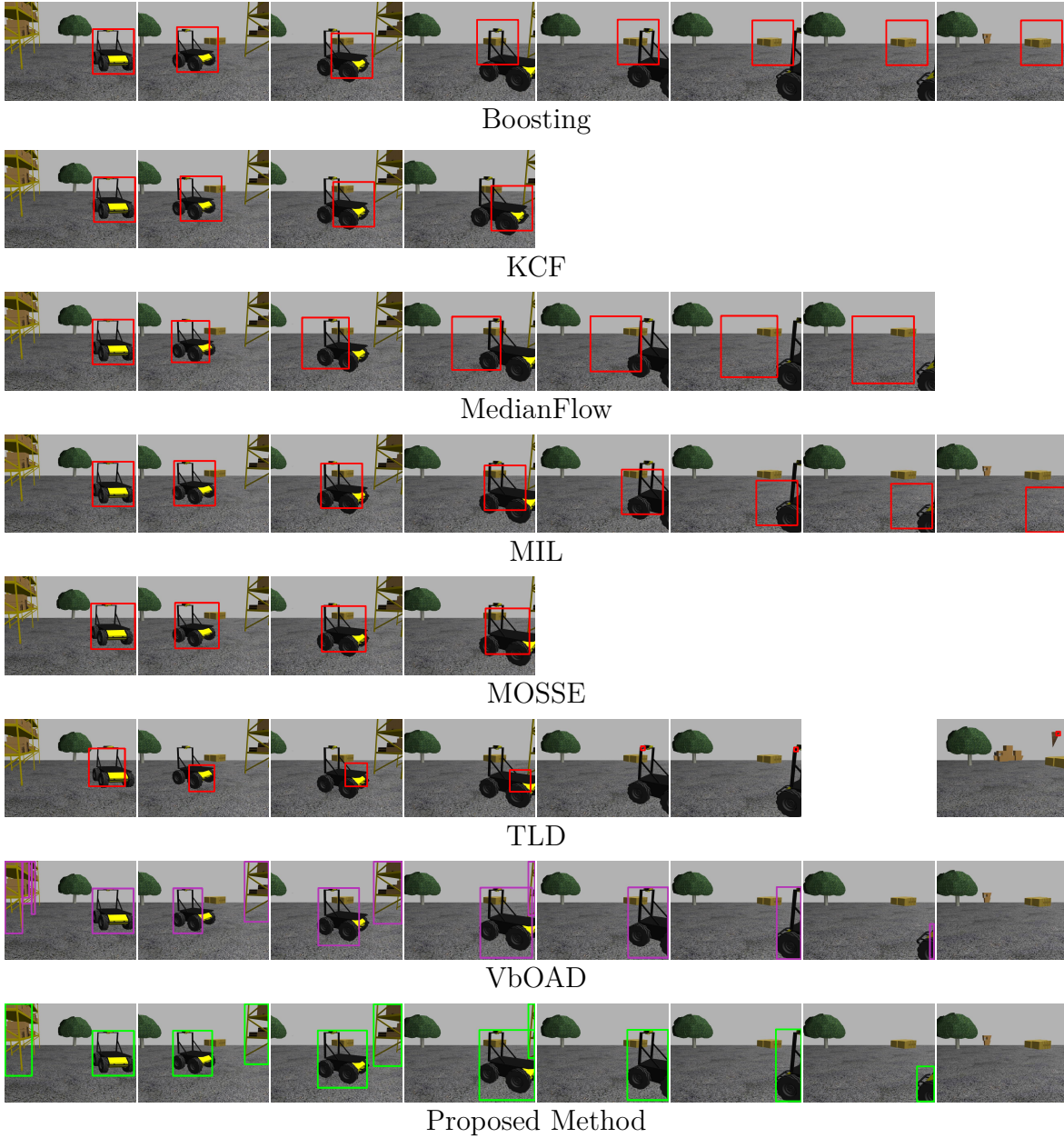


FIGURE 3.12: Comparison of tracking a dynamic obstacle (Husky2), where all RGB-based algorithms (top six rows) are initialized when Husky2 becomes completely visible (first column). VbOAD and the proposed method are initialized when Husky2 is partially visible.

to the eighth columns of the Boosting algorithm in Fig. 3.12. The MedianFlow algorithm starts producing erroneous results when Husky2 changes its appearance from the initialization appearance, as shown from the third column onward in Fig. 3.12, where the estimation error increases even further as Husky2 partially goes out of the FOV. The MedianFlow algorithm also produces erroneous tracking results similar to the Boosting algorithm when Husky2 is totally out of the FOV.

We find MIL to be the best algorithm among all RGB-based tracking algorithms used in this experiment. The performance of the algorithm is similar to the Boosting algorithm. Still, the dimension estimation is less erroneous than the Boosting algorithm, as shown in the snapshots of the MIL algorithm in Fig. 3.12.

The TLD algorithm has a more significant influence on the appearance of the obstacles. Therefore, it is more prone to producing erroneous results when an obstacle changes its appearance. We find erroneous tracking results in the dimension estimation of Husky2, as shown in the snapshots from the second column to the sixth column of the TLD algorithm in Fig. 3.12.

The VbOAD algorithm is executed with a threshold height of 1 m, and it produces more accurate results compared to all RGB-based algorithms. Nevertheless, the proposed method achieves a more precise estimation of Husky2 dimensions compared to VbOAD. This enhancement is evident in the snapshots at the second, third, and seventh columns of the proposed method in Fig. 3.12.

### 3.3.3.3 Initialization at Partial Visibility of Husky2

In this experiment, all algorithms, as mentioned before, initialize or detect Husky2 as a dynamic obstacle as soon as it becomes partially visible to Husky1. Fig. 3.13 shows the tracking and dimension estimation results of Husky2, where the detected position and dimensions are drawn with bounding boxes. The first column of Fig. 3.13 represents the initialization frames, where Husky2 is partially visible. The RGB-based tracking algorithms (first six rows in Fig. 3.13) produce erroneous tracking results and inaccurate obstacle dimensions when Husky2 becomes fully visible. The VbOAD algorithm executes with a threshold height of 1 m. It identifies Husky2 as soon as it is partially visible in the depth image and generates more accurate results than all RGB-based tracking. The proposed method also detects Husky2 as soon as it appears partially in the depth image. The proposed system estimates the dimensions of Husky2 more accurately than VbOAD in some situations, and the accuracy improvement is evident in the comparison snapshots between VbOAD and the proposed method in the 3rd and 4th columns of Fig. 3.13.



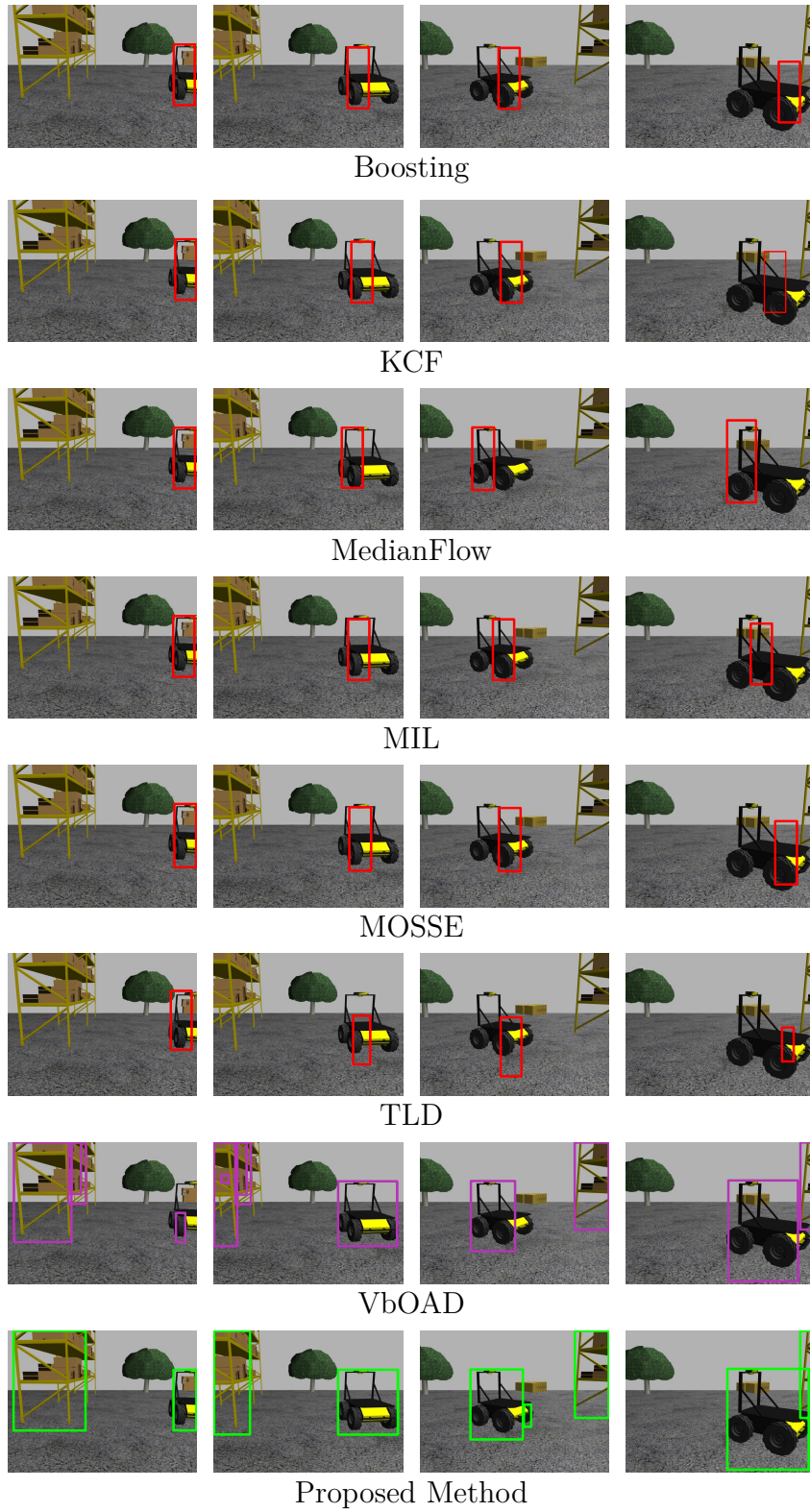


FIGURE 3.13: Comparison among the mentioned tracking algorithms with a dynamic obstacle (Husky2). All algorithms are initialized when Husky2 becomes partially visible (first column).

### 3.3.3.4 Accuracy Comparison on simulated data

We evaluate the accuracy of the proposed method against the GT of the experiment where both VbOAD and the proposed method are initialized when Husky2 is partially visible, while all RGB-based algorithms are initialized when Husky2 is completely visible. Fig. 3.14 presents a comparative analysis in terms of the deviation of the estimated relative distances from the actual relative distances of Husky2 from Husky1. We cal-

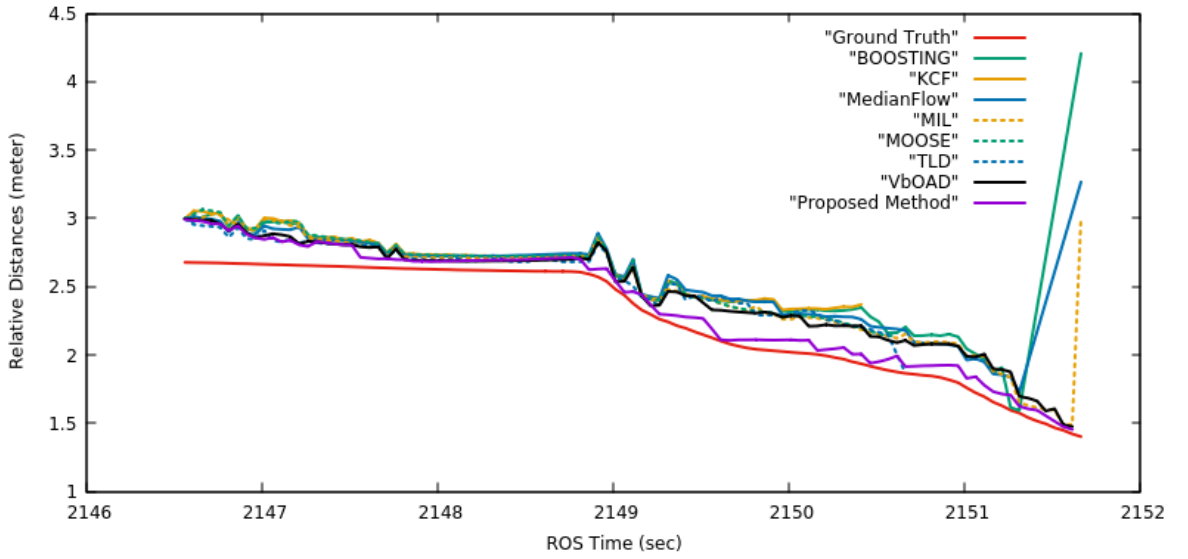


FIGURE 3.14: Comparative analysis of the deviation between the estimated and actual relative distance of Husky2 from Husky1 in the camera coordinate frame  $C$ .

culate the GT of the relative distances by considering the absolute positions of Husky1 and Husky2 from Gazebo. These absolute positions are measured in the body coordinate frame  $B$ . Therefore, this GT represents the relative distances between the two IMUs of the two Huskies. Fig. 3.14 shows the estimated curves of all SoA algorithms and the proposed algorithm, where any estimation curve closer to the GT curve represents more accurate estimation. All tracking algorithms in Fig. 3.14 do not have any knowledge about the position of the IMU in Husky2 and calculate the relative position in the camera coordinate frame  $C$  as explained in Equ. (3.4). The Boosting, MedianFlow, and MIL algorithms show a sudden jump between 2151 sec and 2152 sec to high distances because these algorithms detect an erroneous obstacle at very far distances after Husky2 goes out of the FOV. This incident is captured in the snapshots from the sixth to the eighth columns of Boosting, seventh and eighth columns of MedianFlow, and eighth column of MIL in Fig. 3.12. The curve of the proposed method (purple) is the closest to the GT curve among all the SoA methods presented

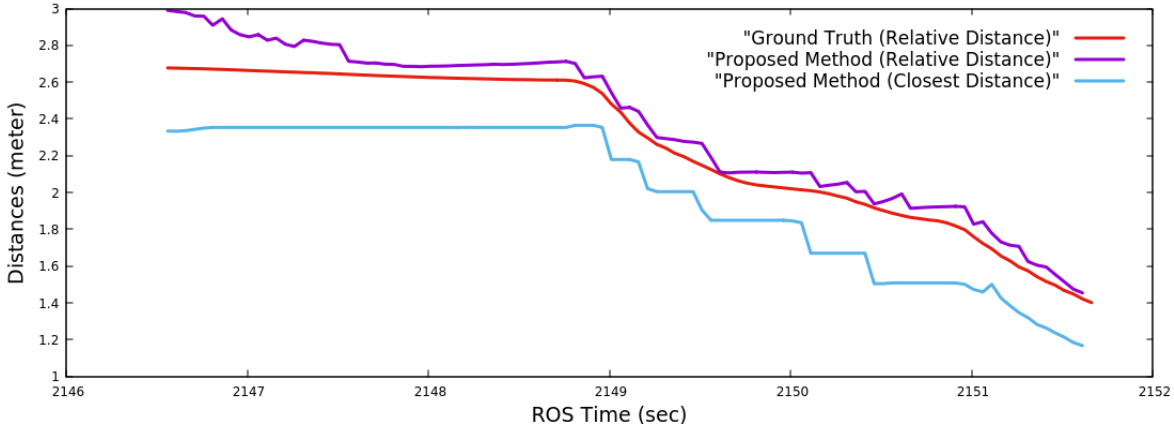


FIGURE 3.15: The proposed method consistently estimates Husky2's closest distance, which is always smaller than the relative distances at any given time.

in Fig. 3.14. Therefore, we can conclude that the proposed dynamic obstacle tracking and dimension estimation show the best accuracy among all the SoA algorithms presented in this experiment.

The position of a Husky is determined by the IMU's location within the body of the Husky. However, in autonomous navigation, the closest distance to any obstacle (i.e., the nearest point on the obstacle body) holds significant importance, as an obstacle avoidance algorithm may utilize this distance for ensuring safe and collision-free navigation. Therefore, we pinpoint the closest portion of the obstacle and estimate the closest distance. Fig. 3.15 compares the estimated closest distances of the proposed method with the GT of relative distances. The estimated closest distance is calculated as the distance between the optical center of the Kinect [42] of Husky1 and the closest surface of Husky2, as detected from the depth image. Consequently, at any given time instance, the closest distance should always be smaller than the corresponding relative distance between the IMUs of the two Huskies [102], and this effect is illustrated in the comparison graph.

Researchers often seek to identify error sources in obstacle estimation by assessing accuracy separately in each axis. Consequently, we evaluate the absolute position of the obstacle in all three axes, aiming to discern error sources by comparing with the GT. Fig. 3.16 illustrates the estimation of absolute positions in all three axes in the world coordinate frame against the GT. A closer alignment between the estimation curve and the corresponding GT curve indicates more accurate estimation. The absolute position of Husky2 is determined as the centroid of the detected dynamic obstacle using the proposed method. In this experimental setup, the XY plane represents the ground plane. Therefore, estimations on the X and Y axes improve as Husky2 approaches Husky1 (around 2146 sec in Fig. 3.16), and conversely, the error increases as Husky2

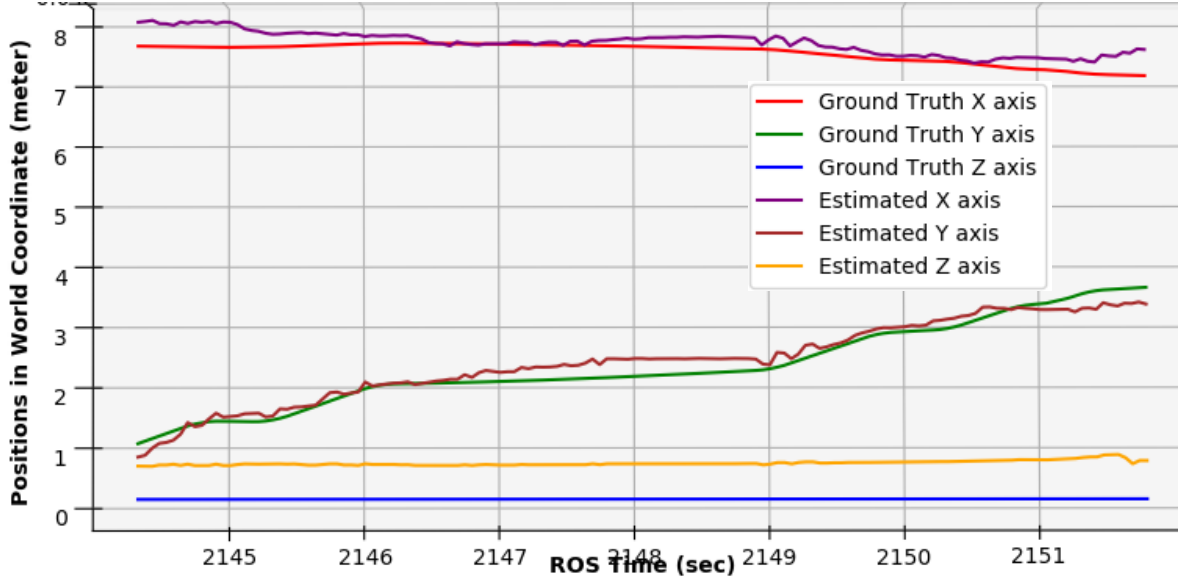


FIGURE 3.16: Absolute positional estimation of Husky2 in the world coordinate frame by the proposed method compared to the GT of the experiment, where all algorithms are initialized when Husky2 is partially visible.

becomes partially invisible to Husky1 (after 2150.5 sec in Fig. 3.16). The Z-axis exhibits nearly constant error (Z-axis in Fig. 3.16), attributed to the actual IMU position of Husky2 being lower than the estimated centroid. We conducted 12 motions within the Gazebo environment, allowing one Husky to observe the other in motion, and repeated each set five times. The maximum error identified in all our evaluation tests is 0.9 m.

### 3.3.3.5 Accuracy of the Proposed Method on Open datasets

We evaluate the proposed method using two open datasets, PTB [7] and Depth-Track [52], each comprising multiple RGB-D videos with GT provided in the form of 2D bounding boxes for object tracking in each frame. Our obstacle detection focuses solely on depth images and does not engage in object-level segmentation. Consequently, conceptual distinctions between object detection and obstacle detection, as explained in Section 1.2.1, limit our ability to directly compare our estimation with the provided GT and execute SoA methods. In this scenario, we omit the comparison with SoA and establish a distinct formulation for comparison with the GT. Let  $A_g$  represent the GT rectangular area and  $A_e$  denote our estimated rectangular area. We define the accuracy measure as  $ACC = (A_e \cap A_g)/A_g$ .  $ACC = 0$  indicates the worst case, while  $ACC = 1$  signifies the best case. Fig. 3.17 displays snapshots from all five training sequences of the PTB dataset, featuring our estimations alongside the GT. The depth images (the second column of Fig. 3.17) with our estimations offer a clearer

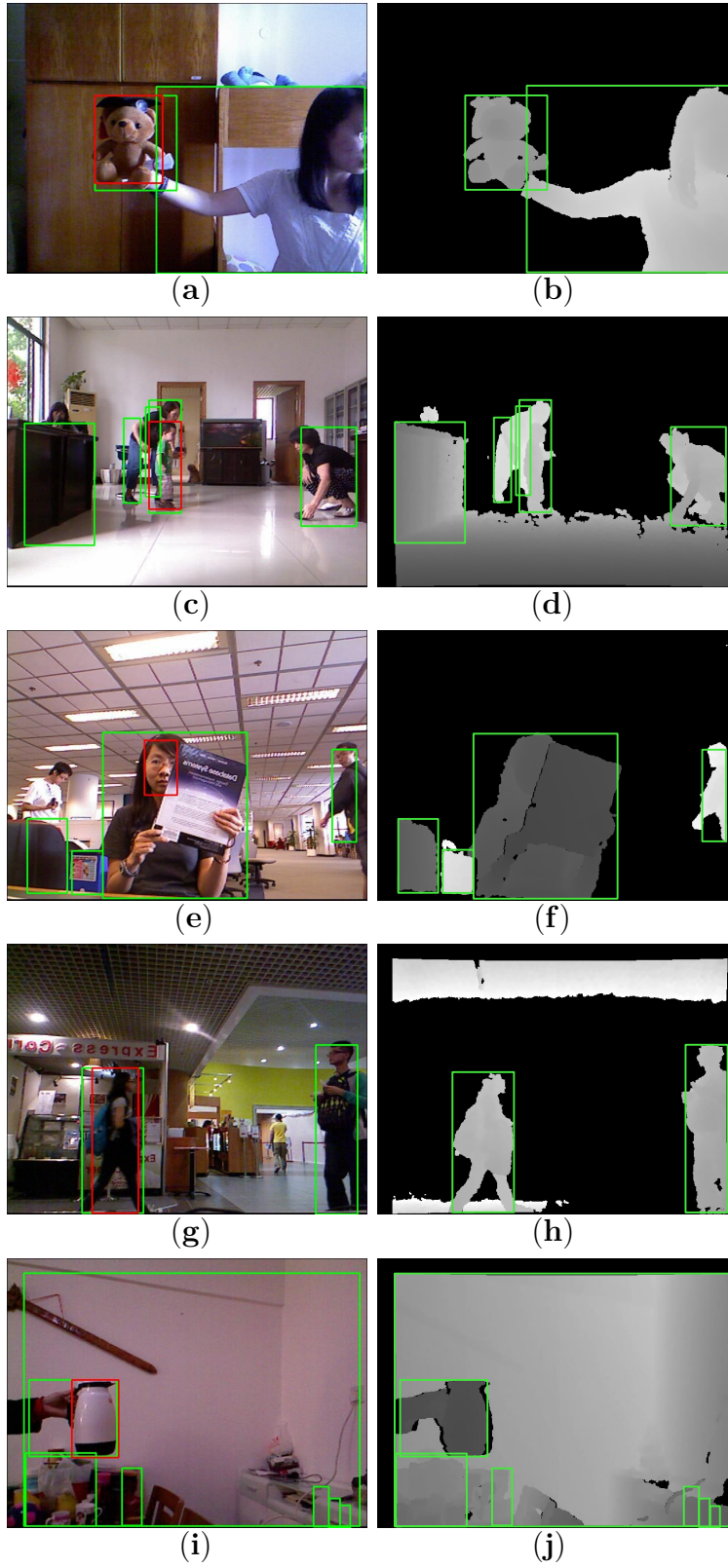


FIGURE 3.17: RGB and corresponding depth snapshots from the PTB data set [7]: (a),(b): bear\_front, (c),(d): child\_no1, (e),(f): face\_occ5, (g),(h): new\_ex\_occ4, and (i),(j): zcup\_move\_1. The RGB images display our obstacle estimation in green rectangular boxes, with ground truth annotations represented by red rectangular boxes.



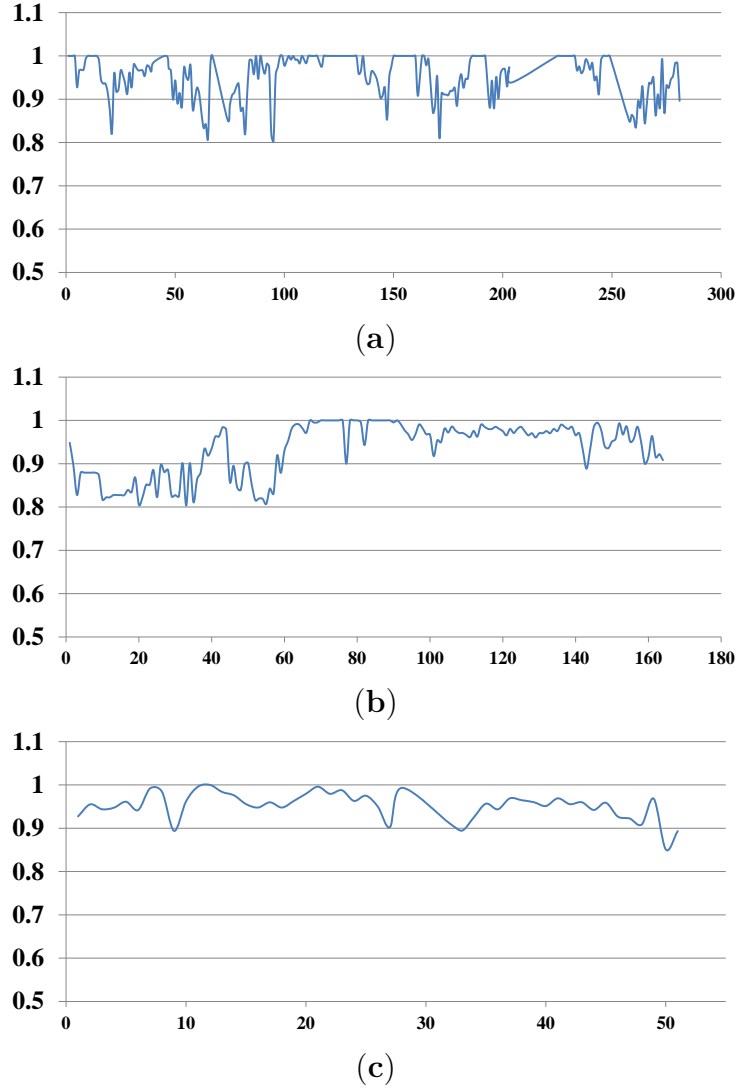


FIGURE 3.18:  $ACC$  plot of our estimation for (a) *bear\_front*, (b) *child\_no1*, and (c) *new\_ex\_occ4* video sequences from the PTB dataset [7]. An  $ACC = 1$  indicates best estimation.

perception of accurate obstacle detection and dimension estimation, successfully segmenting all obstacles with proper dimensions. To quantify the accuracy of the proposed method, we plot the error, and Fig. 3.18 illustrates the accuracy plots of our evaluations on three video sequences of the PTB dataset. The accuracy values consistently exceed 0.8. However, we observe a reduction in accuracy to approximately 0.8 when multiple obstacles are in close proximity, and depth discontinuities are less prominent. In summary, our method showcases strong performance in obstacle detection and dimension estimation, with accuracy metrics consistently surpassing 0.8, as evidenced by evaluations on the PTB dataset, despite challenges posed by close-proximity obstacles.

Fig. 3.19 illustrates the tracking outcomes of our proposed method across eight video sequences sourced from the DepthTrack [52] dataset. These sequences encom-

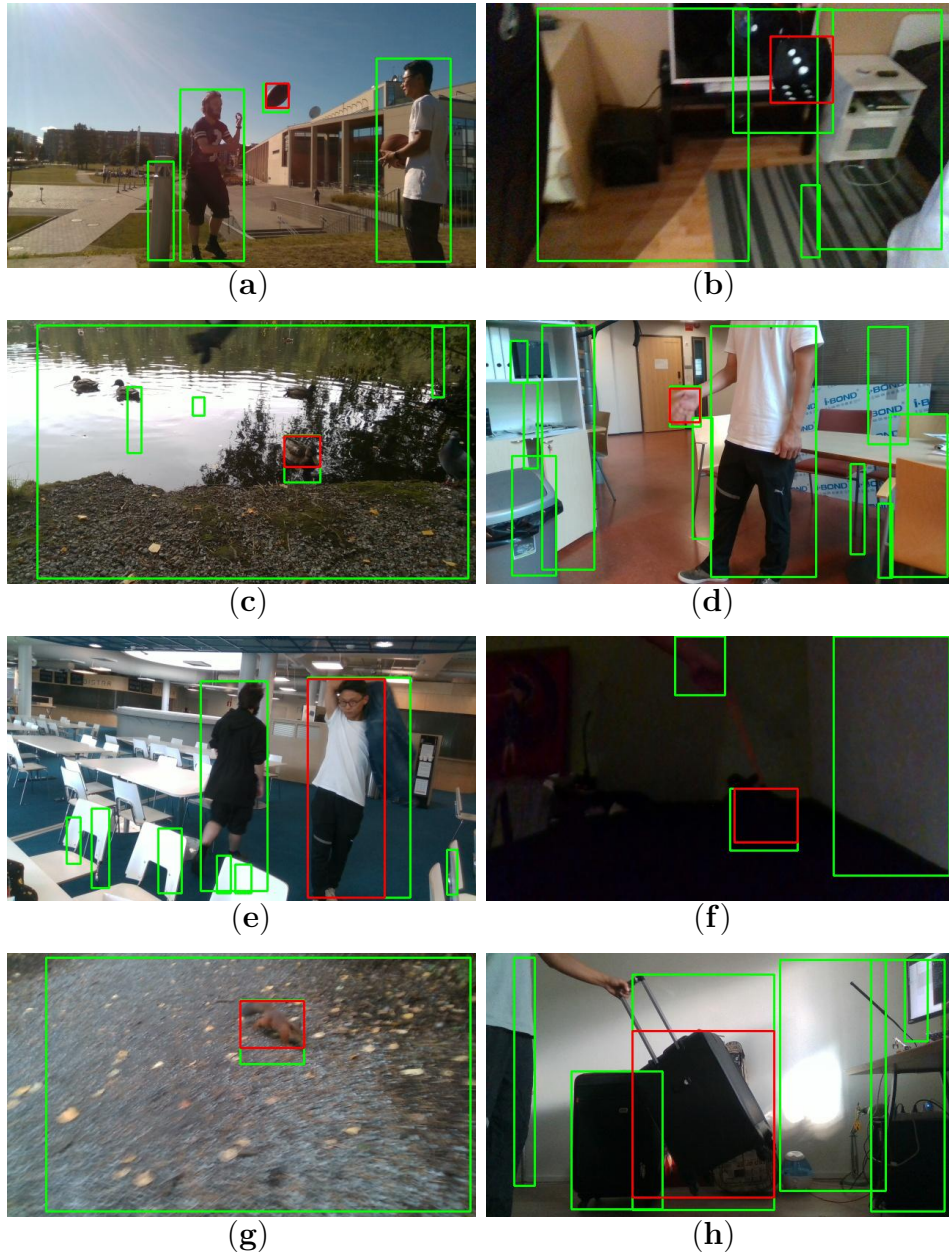


FIGURE 3.19: Results of our proposed obstacle tracking system for the (a) ball10\_wild, (b) cube03\_indoor, (c) duck03\_wild, (d) hand01\_indoor, (e) human02\_indoor, (f) pot\_indoor, (g) squirrel\_wild, and (h) suitcase\_indoor video sequences from the Depth-Track dataset [52].

pass diverse scenarios, including varying lighting conditions, differing motion speeds, and objects of varying sizes, both indoors and outdoors. Notably, challenges arise in tracking a small obstacle, such as a person’s hand, in the hand01\_indoor video sequence (Fig. 3.19(d)), where dynamic thresholding on the u-depth map prioritizes smaller obstacles closer to the camera, leading to occasional failures. A similar trend is observed in the ball10\_wild sequence (Fig. 3.19(a)). Despite these challenges, the proposed method exhibits superior performance in the remaining video sequences, as reflected in the average accuracy values presented in Table 3.4. The table outlines the average  $ACC$  values (i.e.,  $ACC_{avg}$ ) for all tested video sequences from both the PTB and DepthTrack datasets. Despite encountering challenges in tracking small obstacles, such as a person’s hand or small ball, in specific video sequences, our proposed method demonstrates superior performance across diverse scenarios, including variations in lighting, motion speeds, and object sizes, both indoors and outdoors, as evidenced by consistently high average accuracy values exceeding 0.82 (Table 3.4) in the tested video sequences from the PTB and DepthTrack datasets.

TABLE 3.4: Experimental evaluation on PTB[7] and DepthTrack[52] datasets

Dataset	Sequence	Type	$ACC_{avg}$
PTB	bear_front	Indoor	0.952
	child_no1	Indoor	0.934
	face_occ5	Indoor	0.981
	new_ex_occ4	Indoor	0.952
	zcup_move_1	Moving camera	0.913
DepthTrack	ball10_wild	Very small obstacle, direct sunlight	0.821
	cube03_indoor	Very small obstacle, random motion	0.852
	duck03_wild	Daylight condition, moving camera	0.921
	hand01_indoor	Very small obstacle	0.820
	human02_indoor	Human motion	0.948
	pot_indoor	Very high motion	0.914
	squirrel_wild	Jerky motion, moving camera	0.871
	suitcase_indoor	Indoor	0.933

### 3.3.4 Experiment 2: Tracking Single Non-Rigid Dynamic Obstacle

We evaluate the proposed method with a dynamic obstacle that abruptly changes its shape and size while in motion. We choose this scenario because it is a common behavior where robots and humans work in the same environment, and different human

motions result in dynamic obstacles that change their shape and size. We use self-captured outdoor data in indirect sunlight, where a girl of a height of 1.53 m walks toward the camera at an average speed of 1 m/s. She bends down suddenly for 4 seconds, stands up again, and walks again. Fig. 3.20 presents a comparison of the proposed method with the VbOAD. Fig. 3.20(a) presents the tracking results of the VbOAD,

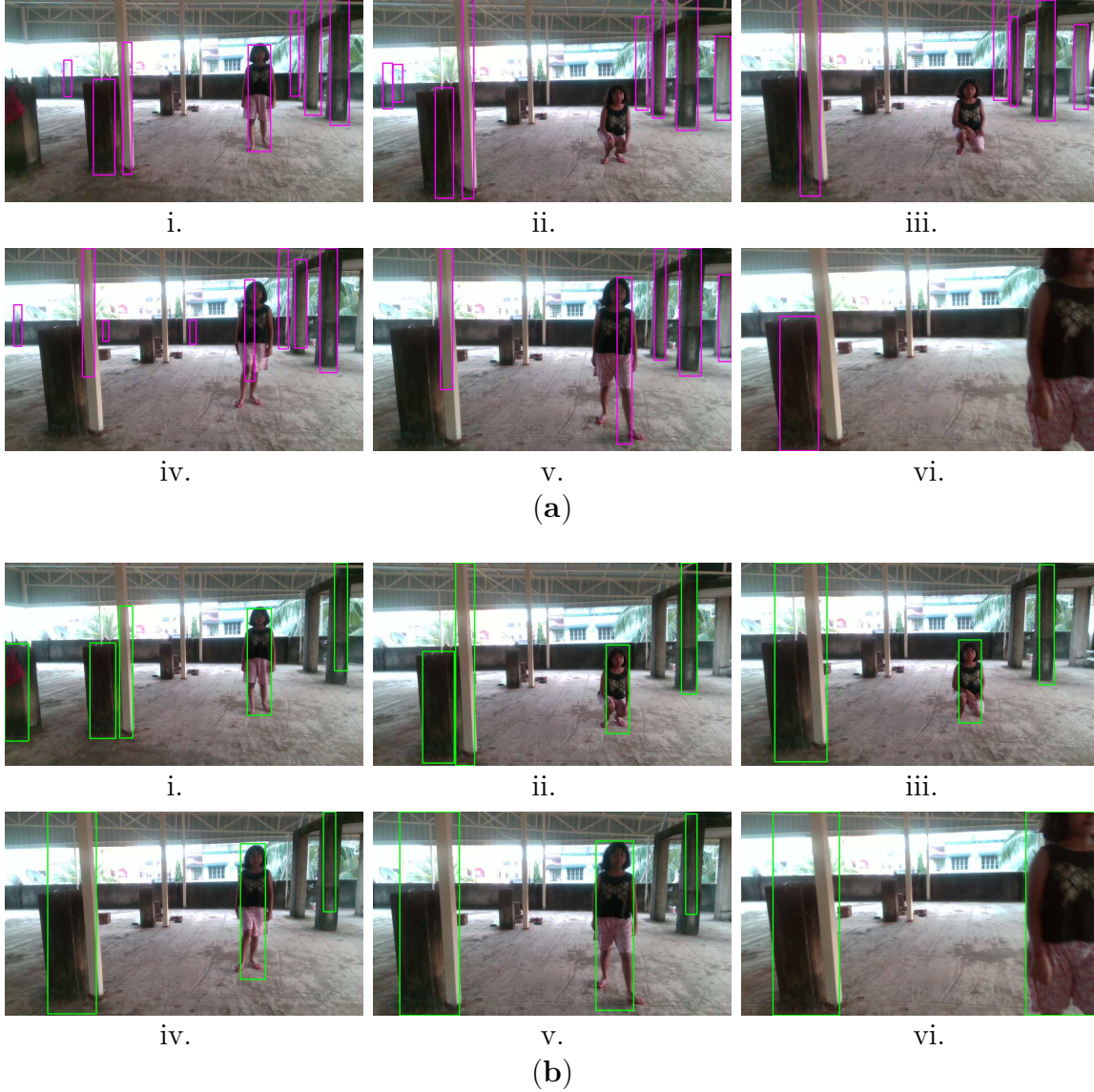


FIGURE 3.20: Comparison of the proposed method with the VbOAD [4] algorithm in the presence of a dynamic obstacle that changes its shape and size abruptly during motion: (a) Tracking output of VbOAD, (b) Tracking output of the proposed method.

where obstacles are denoted with purple-colored bounding boxes. The VbOAD uses a fixed threshold on the u-depth map, and the threshold value is set to 1.524 m. Therefore, the VbOAD detects all obstacles with heights equal to or greater than 1.524 m. The VbOAD algorithm fails to detect any obstacle with a height below the threshold



of 1.524 m. As a result, the VbOAD fails to detect the girl when she bends down and shortens her stature, as illustrated in Fig. 3.20(a)(ii),(iii). Fig. 3.20(b) shows the tracking results of the proposed method. The proposed method successfully detects and tracks the girl continuously while she is within the FOV.

### 3.3.5 Experiment 3: Tracking Multiple Non-Rigid Dynamic Obstacles

We evaluate the proposed method with multiple dynamic obstacles of different shapes and sizes, where the obstacles are either moving in different directions and crossing each other or moving in a group. We choose this scenario because it is a common behavior where multiple humans and robots interact in the same environment. We use outdoor sequences of self-captured data and indoor sequences of the open dataset OpenLORIS-Scene [58].

#### 3.3.5.1 Dynamic Obstacles Moving in Different Directions

We test the proposed method with two dynamic obstacles and multiple static obstacles using self-captured outdoor data in indirect sunlight. Two walking girls are considered dynamic obstacles, where they walk at an average velocity of 1 m/s but in different directions. The two dynamic obstacles cross each other in proximity. Fig. 3.21 shows the tracking results along with the timestamps and estimated distances of both detected dynamic obstacles. The proposed method detects the two dynamic obstacles and successfully tracks them, as shown in Fig. 3.21(a),(b). Then, the proposed method considers the two obstacles as a single one when they come into proximity, as shown in Fig. 3.21(c), and again considers them two as new obstacles as they become farther apart, as shown in Fig. 3.21(d). The accuracy of the proposed system is comparable to VbOAD; thus, it is excluded from the comparison.

#### 3.3.5.2 Multiple Dynamic Obstacles with Different Heights

We evaluate VbOAD and the proposed method with dynamic obstacles of different heights using the indoor open dataset OpenLORIS-Scene. In the data, two people are visible toward the left of the image, and they gradually walk together toward the right and finally leave the FOV. Afterward, a walking child suddenly comes within the FOV from the right edge of the image. Fig. 3.22 presents the comparison output between VbOAD and the proposed method. Fig. 3.22(a) shows the snapshots of the tracking results of VbOAD, where obstacles are denoted with a purple-colored bounding box, and Fig. 3.22(b) shows the snapshots of the tracking results of the proposed

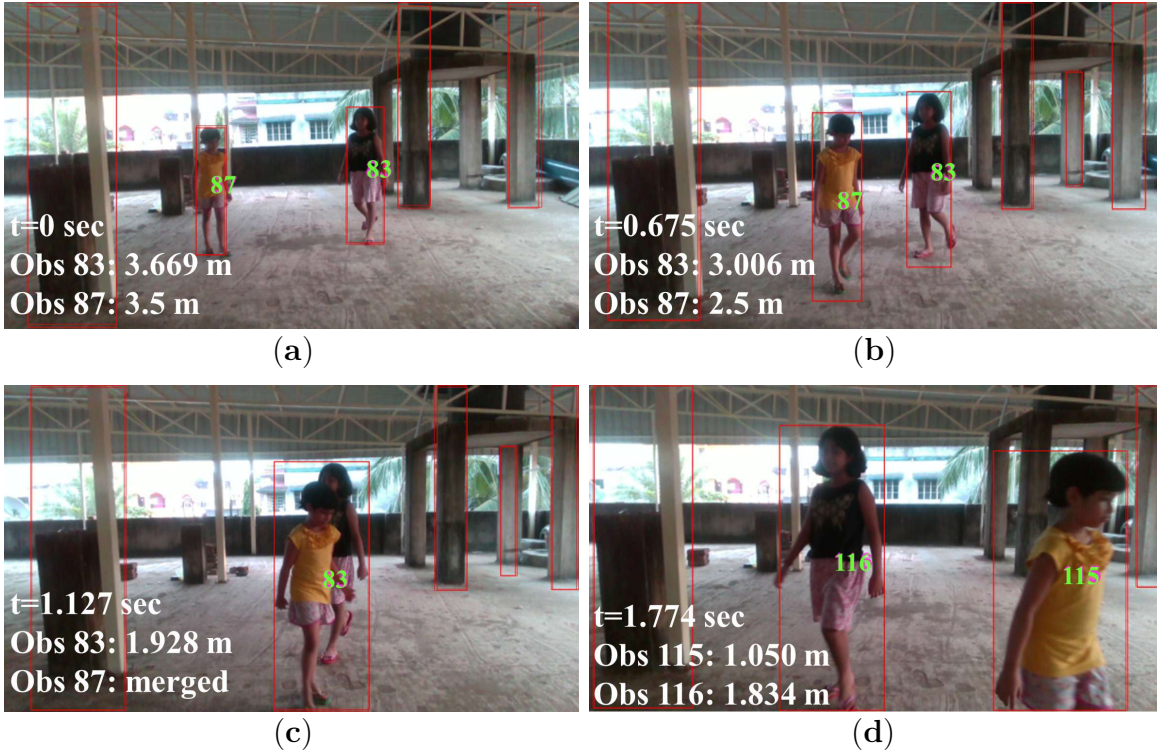


FIGURE 3.21: Performance of the proposed tracking algorithm with two moving obstacles alongside multiple static obstacles: (a–d): Time-sampled snapshots displaying the results using our proposed method.

method, where obstacles are denoted with a green-colored bounding box. The VbOAD algorithm is executed with a 1.524-meter obstacle height, successfully detecting the two walking persons. Still, it fails to identify the true dimensions of the obstacles because of wrong thresholding, as shown in Fig. 3.22(a)(i)–(ii). The VbOAD algorithm fails to detect the small child with a height of about 1 m, as shown in the snapshots in Fig. 3.22(a)(iv)–(vi). The proposed dynamic thresholding on the u-depth map successfully detects and estimates the dimensions of obstacles of various sizes, as shown in Fig. 3.22(b).

### 3.3.6 Experiment 4: Advantages of Dynamic U-Depth Thresholding

We compare the effect of the proposed dynamic thresholding with that of fixed thresholding as proposed in VbOAD. The edges of the white patches on the u-depth map do not become bright when an obstacle has a bent shape, and fixed thresholding may cut the edges of such white patches that are not as bright, irrespective of their position and size. We show this phenomenon in Fig. 3.23 with two examples. Fig. 3.23(a) presents the first example, where Fig. 3.23(a)(i) shows the comparison of detected obstacles



FIGURE 3.22: Comparison of the proposed method with the VbOAD [4] algorithm on the market sequence of the Open LORIS-Scene dataset [58] with multiple dynamic obstacles of different sizes: (a) (i–vi) Time-sampled snapshots with the output of VbOAD, (b) (i–vi) Time-sampled snapshots with the output of our proposed method.





FIGURE 3.23: The effectiveness of the proposed dynamic thresholding compared to fixed thresholding as presented in VbOAD [4] on the market sequence of the Open LORIS-Scene dataset [58]: (a) Example 1, (b) Example 2.



on the same snapshots, and Fig. 3.23(a)(ii) shows the corresponding unthresholded u-depth map. Fig. 3.23(a)(iii) shows the corresponding thresholded u-depth map. VbOAD generates incorrect dimensions of the obstacle in the thresholded u-depth map with a fixed threshold value, as shown in the first column of Fig. 3.23(a)(iii). The fixed value thresholding generates an erroneous result due to the bent shape of the leg of the standing lady. The proposed method with dynamic thresholding produces an accurate thresholded u-depth map, as shown in the second column of Fig. 3.23(a)(iii).

Fig. 3.23(b) shows the output of the second example with a similar pattern as the previous example. The thresholding of VbOAD generates an erroneous result (the first column of Fig. 3.23(b)(iii)) because the obstacles are very close to the camera in this frame, and both sides of the white patches fall below the given fixed threshold value. The proposed dynamic thresholding generates more accurate dimensions in this case, as shown in the second column of Fig. 3.23(b)(iii).

### 3.3.7 Experiment 5: Tracking a Fast-Moving Obstacle

We assessed the performance of the proposed method using self-captured outdoor data featuring a fast-moving basketball thrown toward the camera at an average speed of approximately 5 m/s. The experimental results, presented in Fig. 3.24, depict snapshots displaying only the estimated dynamic obstacles for improved visibility. Each snapshot

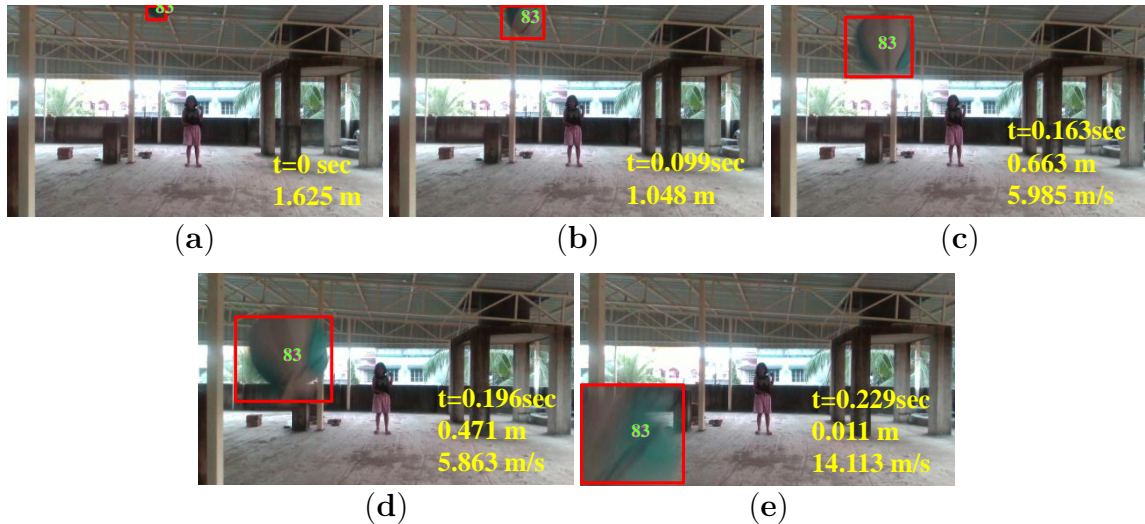


FIGURE 3.24: Performance of the proposed tracking algorithm with a very fast-moving obstacle: (a–e): Time-sampled snapshots displaying the results using our proposed method.

includes timestamps and estimated distances, while Fig. 3.24(c)–(e) additionally denote estimated velocities, calculated after successful tracking in consecutive 5 frames. Initially, the algorithm struggles to detect the ball when it appears small and distant, as the proposed dynamic thresholding of the u-depth map prioritizes nearby objects

with smaller sizes. Consequently, the ball is rejected due to its perceived small size and distance from the camera. However, successful detection occurs when the basketball is approximately 1.625 m away, as shown in Fig. 3.24(a), allowing continuous tracking until it exits the FOV. The average estimated velocity is approximately 7.114 m/s. The basketball's motion, parallel to the camera's viewing direction, results in relatively low positional changes in the u-depth map, facilitating successful tracking. Nevertheless, the system may encounter challenges in detecting obstacles moving with equal or greater speed from left to right or vice versa.

### 3.3.8 Execution Time

Fig. 3.25 provides a comparison of execution times based on an experiment conducted on simulated data, wherein RGB-based algorithms are initiated when the dynamic obstacle (Husky2) is entirely visible, as discussed in Section 3.3.3.2. The minimum,

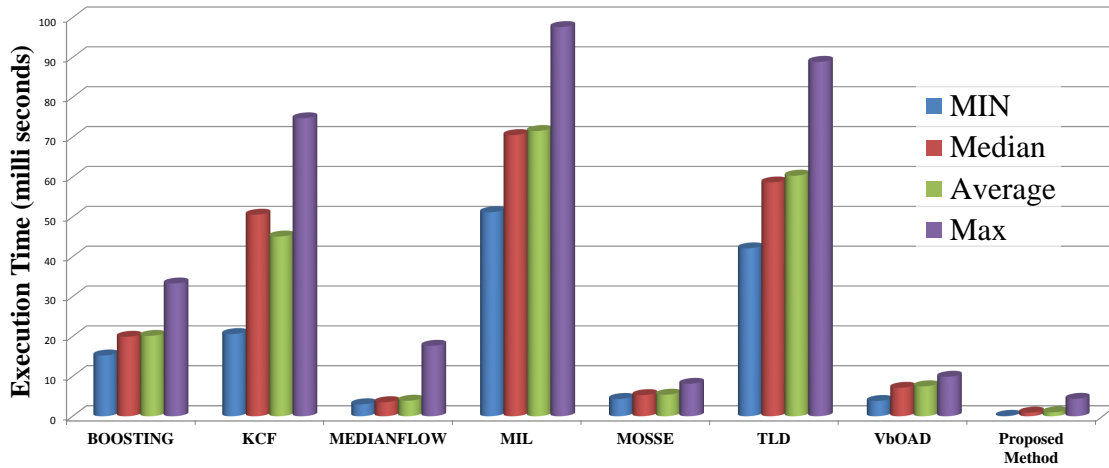


FIGURE 3.25: Comparison of minimum, median, average, and maximum execution times for single obstacle tracking.

median, average, and maximum execution times for the proposed method and other SoA algorithms are presented, with time details specified for single obstacle tracking. The initialization resolution for the testing obstacle on an RGB image is set at  $174 \times 403$ . To enhance clarity, we exclude the initialization time from the plot for all RGB image-based tracking algorithms, given its range between 18.5641 and 100.2011 ms, which impacts the maximum time value. Conversely, the initialization time for the proposed method is 4.3722 ms. Fig. 3.25 highlights significantly higher execution times for BOOSTING, KCF, MIL, and TLD in comparison to MedianFlow, MOSSE, VbOAD, and the proposed method. The time details for VbOAD [4] are sourced from the

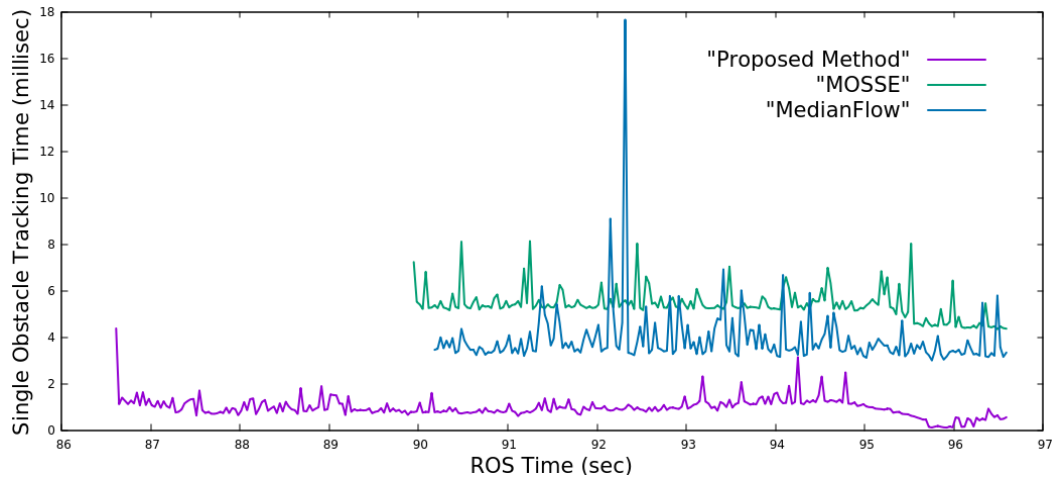


FIGURE 3.26: Comparison of continuous execution time for single obstacle tracking.

literature. For improved visibility and comprehension, we limit our further comparative representation to only MedianFlow, MOSSE, and the proposed method.

Fig. 3.26 illustrates the plots of continuous running time for the proposed method, MOSSE, and MedianFlow in the same experiment mentioned above. In every scenario, the proposed method demonstrates a minimum execution time, as depicted in Fig. 3.26. Correspondingly, Fig. 3.27 provides the box plot, where the 75<sup>th</sup> percentile of the tracking time for a single obstacle in our proposed method remains below 1.15 ms, and the maximum tracking time for a single obstacle is 4.37 ms. The percentile indicates

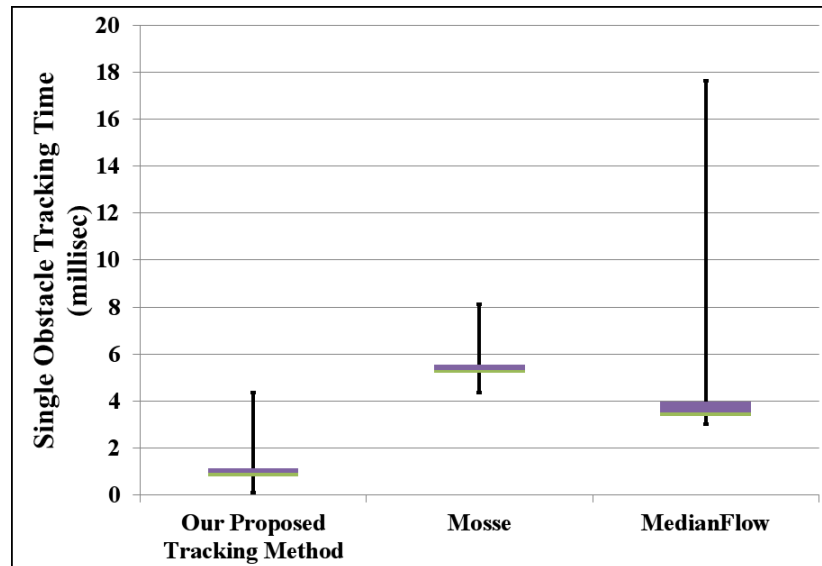


FIGURE 3.27: Box plot of the execution time for single obstacle tracking, as illustrated in Fig. 3.26.

that in 75% of the frames, the obstacle processing time remains under 1.15 ms. In contrast, the 75<sup>th</sup> percentiles of the tracking time for a single obstacle in MedianFlow

and MOSSE are 3.983 ms and 5.547 ms, respectively, which are significantly higher than those of the proposed method. The 75<sup>th</sup> percentile of the tracking time for a single obstacle in the VbOAD algorithm, as reported in the literature, is below 8 ms. Consequently, the proposed method proves to be more than two times faster, supporting the conclusion that it can effectively perform real-time obstacle detection with a 60 Hz camera in parallel with any real-time SLAM and path planner modules.

### 3.4 Conclusions

This chapter introduces a dynamic obstacle detection and tracking system for robotic applications in dynamic environments, leveraging depth images. The system utilizes a u-depth map for obstacle detection and a restricted v-depth map in conjunction with the u-depth map for precise estimation of obstacle dimensions. To enhance detection accuracy and dimension estimation, a dynamic binary thresholding approach is applied to the u-depth map. An efficient algorithm is proposed to track obstacles across diverse scenarios, including indoor and outdoor environments, varying lighting conditions, multiple dynamic obstacles moving in different directions, obstacles with fast motion, small-sized dynamic obstacles, and those dynamically changing shapes and sizes. The system's performance is evaluated using both self-captured and open data sequences. It demonstrates a real-time capability to run onboard with 60 Hz, achieving an average computation time of 0.6 ms per obstacle detection and tracking, successfully tracking obstacles at speeds of up to 5 m/s. The proposed system outperforms SoA methods in terms of obstacle state estimation accuracy and execution time, making it suitable for dynamic obstacle detection in mobile robot navigation. However, the system has limitations in detecting very fast-moving obstacles due to the maximum operating depth range of RGB-D sensors, which is typically around 3 meters. Future work will explore sensors with longer-range perception to detect and track fast-moving obstacles beyond the current system's capabilities.

With this proposed obstacle detection solution, an agent can successfully navigate in a cluttered dynamic environment with a successful collision-free path planning. The maturity in the VI-SLAMs domain for single agent motivate us to solve problems of collaborative visual navigation, where multiple robot can share information and help each other to complete a big and distributed mission quickly, we shall look to formulate a collaborative VI-SLAM framework to support multi-agent cooperative navigation.

## Chapter 4

# An Adaptive Collaborative Visual-Inertial SLAM for Multi-Agent Localization

### 4.1 Introduction

In Section 1.2.3, we have studied the importance of VSLAM in the field of autonomous navigation by robots because VSLAM can estimate the robot pose and 3D scene structure accurately. However, VSLAM has some inherent problems such as cumulative drift, losing camera track, unknown scale [10], etc. In Section 1.2.4, we have also seen that VI-SLAM can alleviate some of these problems due to the complementary nature of IMU with the camera. In general, VI-SLAM is expected to produce better pose estimation than VSLAM, but this depends on the quality of the data that the IMU sensor generates from the motion [18]. In practice, noisy IMU measurements can reduce the quality of combined measurements. Therefore, the system requires intelligence to validate the IMU measurements before fusing them with any other sensors.

The use of multiple heterogeneous mobile robots on a big mission is advantageous because a complicated task can be broken down and allocated to multiple robots based on their capabilities to reduce the mission completion time. A collaborative SLAM framework helps multiple robots to navigate in an unknown environment in cooperation, generates a global environmental map, and shares information among the robots. Information sharing leads to the rapid exploration of the region and results in a more reliable system because the system can still function even if one or more robots become inoperable. The major challenges of any collaborative framework are communication among participating robots, information availability among all robots, fusing partial maps generated by individual robots, avoiding losing track, the reuse of map information, handling network delays, etc.

In this spirit, we propose a collaborative VI-SLAM framework based on the research work in [109], CORB2I-SLAM, where each participating robot is treated as a client and accompanied by at least one visual sensor (e.g., monocular/stereo/RGB-D camera) and may carry an IMU. These robots have limited processing capabilities and navigate using either VIO or VO on a local map depending on the availability of the IMU. The framework contains a centralized server with higher processing and larger storage capability. The server receives information from all participating robots and executes all computationally complex tasks for VSLAM, e.g., loop closing, map management and merging, and optimization. The optimization is in the form of Bundle Adjustment [110], where all the camera poses and the 3D structure are further globally refined through reducing the average reprojection error, as explained in Appendix A.2.5. The central server shares optimized information with the respective robots whenever required. The rest of the chapter uses robots, clients, or agents synonymously to indicate the participating robots. Our proposed system uses ROS [111] based message passing.

This chapter is organized as follows. Section 4.2 describes the proposed collaborative framework and the contributions. Section 4.3 presents the experimental results. Finally, Section 4.4 concludes the chapter.

## 4.2 CORB2I-SLAM Framework

We present the architecture of our proposed framework in Fig. 4.1. Every client robot

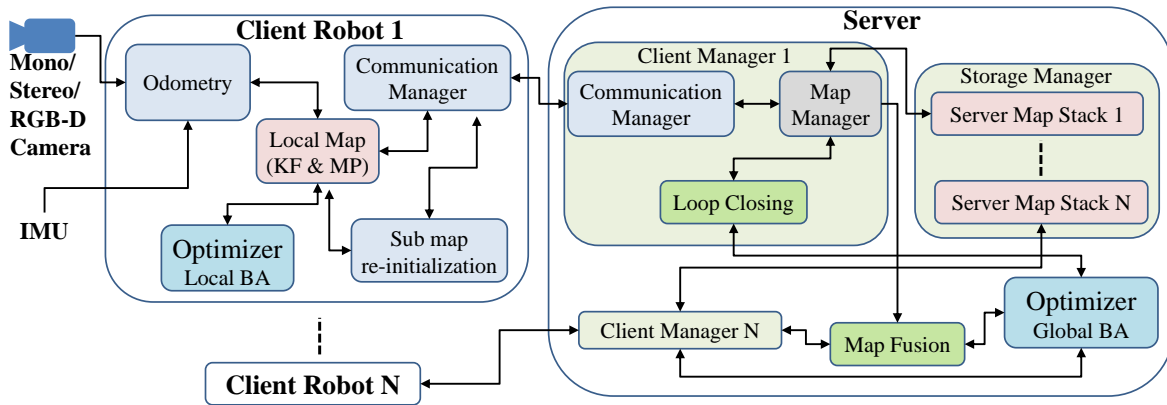


FIGURE 4.1: CORB2I-SLAM architecture.

is equipped with at least one monocular/stereo/RGB-D camera, an IMU (optional), a processing unit, a small memory unit, and a wireless module. The central server contains high-performance computation capabilities along with large storage and wireless communication capabilities. The system configuration does not assume any specific

configuration of client robots, so any type of UGV or UAV can participate as a client. If IMU is available, any agent must try to initialize with VIO and continue with VIO if the biases of the accelerometer and gyroscope, gravity direction, and velocities are estimated correctly; otherwise, VO estimation follows. In this context, we assume that the environment is feature-rich and has sufficient illumination.

Each client executes ORB VO/VIO on its local map, adhering to a design comparable to ORB-SLAM2 [15]. In our current system, the local map structure adopts a sparse map, avoiding of a dense map formulation on the client to mitigate computational load. Consequently, the map structure, KFs, and MPs follow conventions similar to ORB-SLAM2, differing only in their globally unique identifiers. Initially, every client receives a distinctive client identification number (client ID, e.g.,  $Client_1$ ,  $Client_2$ , etc.) from the server and initializes a local map. Representing the immediate vicinity of the client, the local map, containing a fixed number (N) of KFs, prompts clients to offload the map structure, inclusive of all KFs and MPs, to the server. Simultaneously, they discard old KFs and MPs not part of the local map. To minimize communication overhead, messages are exclusively designed for visual data, resulting in the server map comprising solely visual data.

The server allocates a dedicated client manager for each client to handle data management for associated clients. This management includes receiving new KFs and MPs from the corresponding client and storing them in the appropriate stack. The server dispatches past KFs and maps back to a client when it detects the client approaching a location near a past location and the client's local map lacking the past location. If the client identifies matches with these past KFs, the server executes a loop closer. We utilize an analogous transmission protocol for transferring data between a client and the server, akin to the approach employed in CCM-SLAM [21]. A novel re-initialization procedure, incorporating a globally unique map, is implemented when any client experiences track-loss. All computationally intensive algorithms, such as intra-map and inter-map place recognition, map fusion, and global Bundle Adjustment [110], run on the server. This design links the collaborative functionalities of the framework to the server's performance, which diminishes as the number of active clients increases. Nevertheless, the real-time operations of the client robots remain independent of the server, ensuring consistent efficiency for the client robots.

### 4.2.1 Notations

The position and rotation of the world frame  $W$  relative to the  $i$ -th camera frame can be described by the rigid body transformation  ${}^W\mathbf{T}_{C_i} \in SE(3)$ , where  $SE(3)$  is the special Euclidean group [5],  ${}^W\mathbf{R}_{C_i}$  is the corresponding rotation matrix and  ${}^W\mathbf{t}_{C_i}$  is

the corresponding translation vector. A landmark is represented as a 3D point  $\mathbf{x} \in \mathbb{R}^3$  with its image projection as  $\mathbf{u} = \varphi(\mathbf{x})$  where  $\mathbf{u} \in \mathbb{R}^2$ .

### 4.2.2 Client Robot

We start with the description of a client robot, where we take a sample client to look into the internal modules. Fig 4.2 present a modular design of a client, where the modules are described in the following sections.

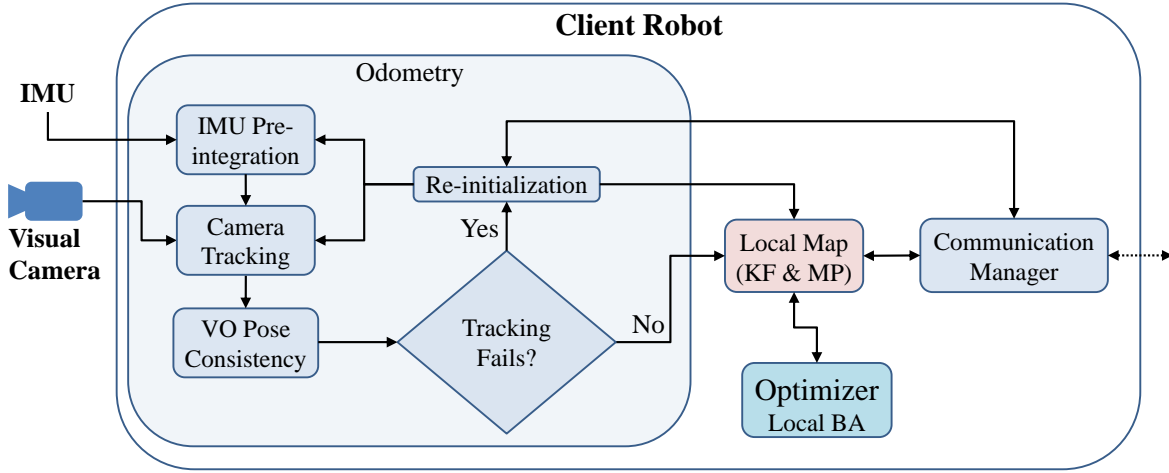


FIGURE 4.2: Internal modular design of a client in CORB2I-SLAM.

#### 4.2.2.1 Odometry

We have explored the process of estimating camera pose from IMU motion and the benefits of leveraging IMU measurements for this purpose, such as mitigating tracking failures in rapid or erratic camera movements (discussed in Section 1.2.4). Consequently, we develop a self-localization system based on VIO. However, we also maintain the option for VO in cases where the IMU is unavailable or its estimation is unreliable. The integration of IMU and visual estimations in the VIO system is tightly coupled [79]. To facilitate this, we adopt the approach of pre-integrating IMU measurements, as proposed in [79], which is briefly explained below.

#### IMU Pre-integration

ORB-SLAM2 [15] is a feature-based VSLAM system that incorporates the concept of KF into the map structure. Consequently, the motion estimation of the current camera frame is always performed with respect to the last KF. Since the IMU frequency is usually much higher than that of the visual camera, a one-to-one matching is not feasible. Therefore, it is necessary to extract the IMU data captured from the last KF



to the current frame and integrate them to estimate the small motion of the current frame from the last KF. This integration procedure of IMU data is referred to as pre-integration.

When considering the current frame as the latest KF, pre-integration of the IMU measurements between two consecutive KFs is required to calculate the motion of the latest KF from the previous one. We employ a manifold-based pre-integration approach to transform IMU measurement data into visual KF constraints to estimate the camera pose, as proposed in [79]. Therefore, for a detailed understanding of using IMU measurements to estimate camera poses through manifold-based pre-integration, we refer to [79].

### Camera Tracking

In the present system, the initialization of the VO tracking is initiated by extracting ORB features from two frames: a reference frame and the current frame. Their poses are then estimated either by homography or by a fundamental matrix, following a similar approach to ORB-SLAM2. After a successful initialization, a map structure is created with KFs and MPs, where the MPs constitute the visual structure of the scene. Inspired by CVI-SLAM [26], we design the initialization of VIO. To prevent large IMU pre-integration, we maintain a distance of 5 frames between two consecutive KFs. The visual structure is optimized using Bundle Adjustment with 20 KFs, and the gyroscope bias is initialized using a linear least squares approach [112]. Linear least squares is a method for fitting a mathematical or statistical model to data in cases where the model provides the idealized value for any data point. The ‘linear’ term is introduced because the idealized value is expressed linearly in terms of the unknown parameters of the model.

The unknown scaling parameter for the metric scale, velocity, and gravity direction are estimated linearly. Upon obtaining reliable estimations of gravity direction and velocity, we assume correct IMU estimation, scale the visual structure, and align it with the gravity direction. If IMU estimation fails three times consecutively, we temporarily declare the IMU unstable. In such cases, we proceed with pure VO estimation as in ORB-SLAM2. The client system continuously checks for nonlinear client motion using estimated poses of the KFs from VO. When non-linear motion is detected, IMU estimations are reiterated (as described in Section 4.2.2.1), followed by VIO initialization if stable IMU estimations are achieved; otherwise, the IMU is assumed to be permanently unstable.

We extract ORB features on every incoming frame and integrate the IMU measurements accumulated since the last KF to estimate the motion model of the current frame. This motion model aids in predicting the pose of the incoming frame, and a guided search is performed to find 2D correspondences by projecting the 3D map points into the current frame. We perform a two-step frame alignment: (i) the initial alignment is based on the matched correspondences, and (ii) further matching correspondences are searched on the frame by projecting other map points. The aligned frame is finally optimized using motion-only BA on a local window. The current frame is selected as a new KF if one of the following conditions is satisfied, where the parameter values are experimentally determined.

- (a) The current frame is 20 frames apart from the last KF.
- (b) The current frame observes fewer than 15 old MPs.
- (c) 2D key points cover less than 40% of the image area.

In the case of VO, the process follows the procedure outlined in ORB-SLAM2, and the current frame is chosen as a new KF when one of the following conditions is met, with parameter values determined experimentally:

- (a) The current frame is 20 frames apart from the last KF.
- (b) The current frame observes fewer than 70 close MPs.
- (c) The current frame observes fewer than 80% MPs than the last KF.
- (d) 2D key points cover less than 40% of the image area.

Every client periodically sends the newly created KFs and MPs to the server and subsequently deletes older KFs and MPs that do not belong to the local map.

### VO Pose Consistency

It is noted that, VIO estimates the initial camera motion from IMU pre-integration. Whereas the estimated poses using VO are purely based on visual features, providing a geometric interpretation. Consequently, poses may become erroneous when visual features lack geometric richness. To address this, we evaluate the poses of each frame through a heuristically proposed verification step to ensure correct camera tracking in the case of VO estimation.

**Scene Geometry Consistency:** Once the detected points lack sufficient geometric information to convey the structure, the estimated pose is likely to be erroneous.

We measure the structural continuity of the reconstructed visual structure to measure geometric consistency, as proposed in [16]. First, we extract edges from the RGB image and reconstruct the longest edge in 3D using sampled points on the selected edge and the estimated pose of the frame. Subsequently, we examine the depth continuity of the reconstructed 3D points using their positional coordinates. In this context, we endeavor to extract the line by fitting a line equation, as explained in [113]–[115], and consider the 3D line segment continuous if all the 3D points satisfy the line equation. The underlying assumption is that a line segment would be continuous in 3D if it exhibits continuity in 2D. Once the depth continuity is disrupted, we consider that the pose is not accurate enough for tracking.

**Pose Observability:** Camera pose observability, initially proposed in ORB-Atlas [69], is utilized in a modified form. Camera pose estimation tends to deteriorate when tracked features have a very high depth. The 2D motions of such features on consecutive images become negligible and fail to encode the true motion of the camera. Therefore, we use the uncertainty of observing a 3D point  $\mathbf{x}_i$  from a camera  $C_k$ , denoted as  $\Psi_{\mathbf{x}_i, C_k}$ . This uncertainty is proportional to the observational depth of  $\mathbf{x}_i$ , encoding a higher depth point with a higher uncertainty of observability.

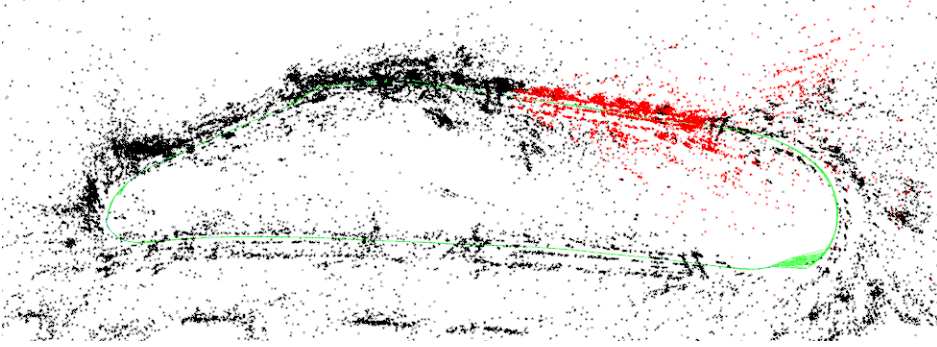
The estimated six-degrees-of-freedom (DoF) camera pose of the  $k^{th}$  frame is represented as  ${}^W\hat{\mathbf{T}}_{C_k}$ . We express the uncertainty of this estimated pose with an unbiased Gaussian vector of six parameters,  $\boldsymbol{\rho}_{C_k}$ , defining the Lie algebra approximation of  ${}^W\mathbf{T}_{C_k}$  around  ${}^W\hat{\mathbf{T}}_{C_k}$ , as given in Equ. (4.1).

$$\begin{aligned} {}^W\mathbf{T}_{C_k} &= \exp(\boldsymbol{\rho}_{C_k}) \oplus {}^W\hat{\mathbf{T}}_{C_k} \\ \boldsymbol{\rho}_{C_k} &= (x, y, z, \omega_x, \omega_y, \omega_z) \sim \mathcal{N}(0, \mathbf{Cov}_{C_k}) \\ \mathbf{Cov}_{C_k} &\approx \left( \sum_{\mathbf{x}_i} \mathbf{J}_{\mathbf{x}_i, C_k}^T \Psi_{\mathbf{x}_i, C_k} \mathbf{J}_{\mathbf{x}_i, C_k} \right)^{-1} \end{aligned} \quad (4.1)$$

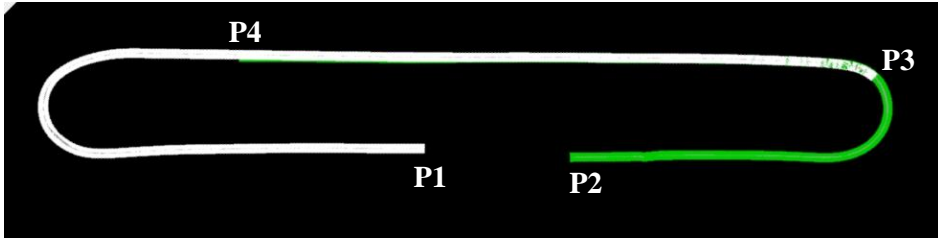
where,  $m$  map points are visible on camera  $C_k$ .  $\mathbf{Cov}_{C_k}$  is the covariance matrix that represents the observability accuracy of camera  $C_k$ , and  $\mathbf{J}_{\mathbf{x}_i, C_k}$  is the Jacobian of the observability measurement of camera  $C_k$  for point  $\mathbf{x}_i$ . Translation is often the most weakly estimated parameter, as described in ORB-Atlas. Consequently, we attempt to obtain an error estimation of translation only with the diagonal values of  $\mathbf{Cov}_{C_k}$ . We consider the camera tracking to be lost either when we find the number of tracked points below the threshold or when we find the pose consistency to be very low, as described in Equ. (4.1). Fig. 4.3 illustrates an example of the utilization of pose consistency evaluation. Fig. 4.3(a) shows the GPS GT path on the Google



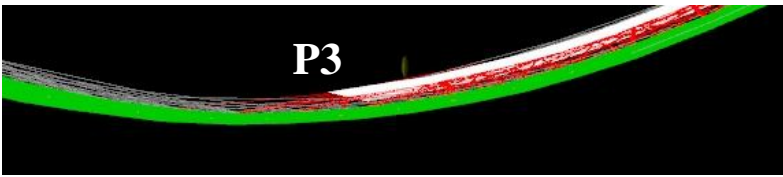
(a)



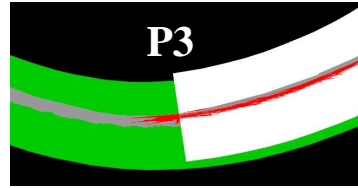
(b)



(c)



(d)



(e)

FIGURE 4.3: Improved accuracy with pose-consistency evaluation: (a) GPS ground truth path for Malaga 07 dataset [116] on Google Maps, (b) Map is generated using ORB-SLAM2 [15], (c) Merged camera track with our pose-consistency evaluation, where white and green denote two individual camera tracks, (d–e) Magnified views showcasing map fusion accuracy at location P3.

map from the Malaga 07 [116] sequence, and Fig. 4.3(b) displays the erroneous path estimation using ORB-SLAM2, where a straight road is estimated as a bent road. In Fig. 4.3(c), the camera track estimation using the proposed method is presented, where P3 indicates the starting position, signifying that SLAM was initialized at P3, moved towards position P4 (white path), took a u-turn, and the pose consistency failed at position P1, leading to track-loss mode. The proposed pose consistency prevents the system from continuing with erroneous pose estimations that can decrease overall accuracy. We further describe the re-initialization process after losing the camera track and demonstrate the overall improvement in accuracy for the example in Fig. 4.3.

### Re-Initialization

A client may lose track of an incoming frame either due to erroneous pose estimation in VO or due to the reduction in the number of 3D-2D matched features. The client attempts to relocalize with its own local map for a short duration of 2 seconds, as in many cases we found that a sudden jerk creates track loss and the client regains track immediately after the jerk. The client enters into a track-loss mode if it is unable to regain track within the specified time. In such a scenario, the client has two options: (i) persist in attempting relocalization or (ii) reinitialize from the beginning and resume. If the client chooses the first option, autonomy is greatly affected because the client can only start navigation after relocalization, and it can impact the completion time of the entire mission. If the client chooses the second option, autonomy is restored, but the trajectory would not be continuous, and there would be extra overhead to fuse multiple sub-maps. We opt for the second option, where our autonomy is unaffected.

Fig. 4.4 shows the message flow diagram after a client loses camera track. The client first tries to relocalize for 2 seconds and immediately sends a track-lost notification message to the server once the relocalization fails. The track-lost notification message contains the last KF id of the local map. The server creates a new map structure for that client in the corresponding map stack upon receiving the track-lost message and waits until the last KF is received. The server sends an acknowledgment message with the next available client id to the client immediately after the last KF is received. The client again reinitializes from the beginning with the new client id and a new local map and continues with VIO or VO based on its previous configuration. The ‘Client Robot’ and the ‘Server’ blocks of Fig. 4.4 contain these corresponding descriptions. To avoid ambiguity, each local map uses a globally unique identifier, which is the associated client id.

Let us revisit the example in Fig. 4.3, where the agent goes into track-loss mode at

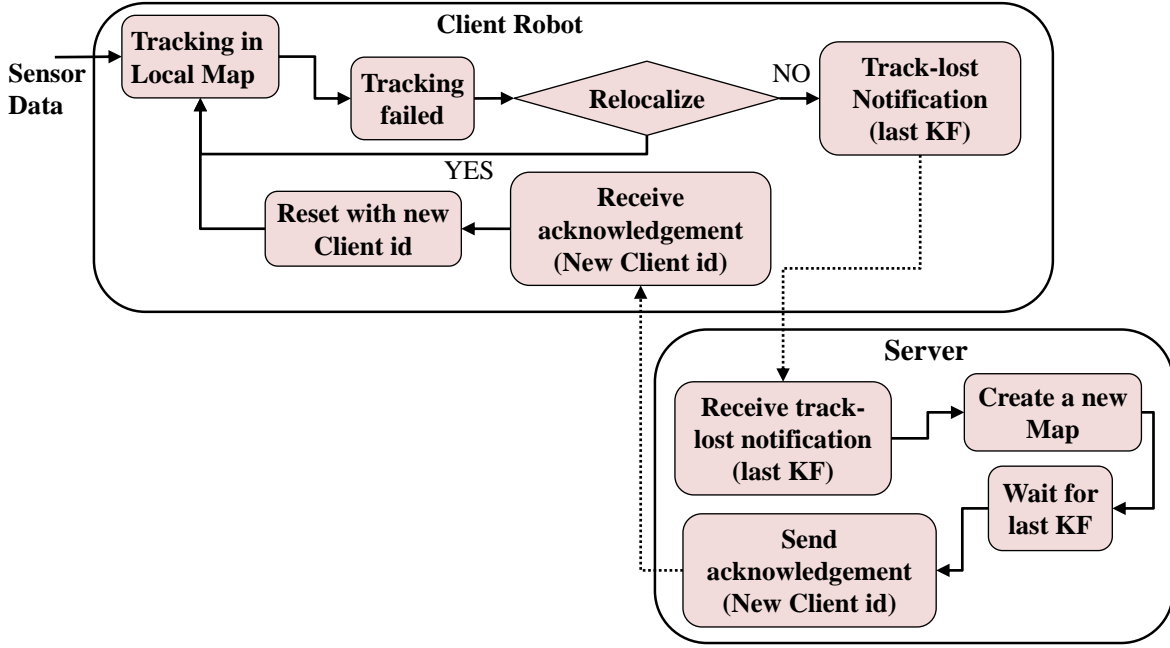


FIGURE 4.4: Information flow after camera tracking failure.

position P1, as shown in Fig. 4.3(c). The agent is reinitialized at position P2 and continues tracking. Fig. 4.3(c) shows the camera track (green path) after the re-initialization. The agent moves towards position P3, and the map-matching module finds matches between these two maps when the agent arrives at position P3. The map fusion module (Section 4.2.5.3) merges these two maps, and Fig. 4.3(c) shows the overlapped regions from position P3 to position P4. The overall accuracy of our estimated camera track is greatly improved from the ORB-SLAM2 estimation, because the straight road is estimated as a straight road. However, the camera track becomes discontinuous between positions P1 and P2. The camera track enhancement and discontinuity are realized pictorially in Fig. 4.3(a)–(c).

### 4.2.3 Map Structure and Optimization

The map structure of each client consists of KFs and MPs, where each KF and MP is uniquely numbered to identify them without any ambiguity. A KF is identified as  $KF_{x_i}$ , where  $x$  and  $i$  represent the client number and KF number, respectively. A MP is identified as  $MP_{y_j}$  in the map structure, where  $y$  and  $j$  represent the client number and MP numbers, respectively. The local map on the client consists of the nearest  $N$  number of KFs from the current location of the camera and is periodically updated by the insertion of new KFs. Now, the local map structure has KFs with two different types of connected constraints in the case of VO and VIO. We follow a similar structure as proposed in CCM-SLAM [21] for VO estimation. In the case of VIO, the

KFs have connected constraints with IMU observations between consecutive KFs and covisibility constraints with common MPs visibilities.

The local map structure on a client is periodically refined with local BA. The local BA runs on a local window of a fixed number of KFs, which is smaller than the local map size to ensure valid IMU constraints within a small boundary. The local BA does not optimize poses for KFs that are already updated and inserted into the local map from the server.

#### 4.2.4 Communication Manager (Server and Client)

The communication manager serves as the interface for communication between the client and server. In this scenario, the communication manager converts any information to a ROS message and vice versa for sending and receiving. The communication manager ensures the successful delivery of every message using an acknowledgment mechanism and retransmits messages if it does not receive proper acknowledgments within a given time span. The communication manager is designed to send KFs and MPs periodically: (i) it transmits complete information for new KFs (e.g., 2D features, associated feature descriptors, and 3D map points) and MPs, and (ii) transmits only the updated information for old KFs and MPs. Thus, the communication manager always maintains control over the network bandwidth.

#### 4.2.5 Server System

The server functions as a central system with robust computing capabilities, primarily dedicated to executing processor-intensive tasks. Its core responsibilities include organizing and storing client maps, identifying loop closures within individual maps, finding matches across multiple maps for fusion, and optimizing maps through global Bundle Adjustment. A detailed modular block diagram of the server is illustrated in Fig. 4.5. The server features multiple client managers, each linked to a specific client for communication and mapping operations. Additionally, a singular storage manager oversees map stacks, providing selective access to client managers for necessary map modifications. The server incorporates a map fusion module, facilitating the matching of KFs from various clients and subsequent map fusion upon identification of matches. The Optimizer module is in charge of executing global BA after each loop closure and map fusion. In summary, the server, equipped with high computing capabilities, manages and optimizes client maps, facilitates loop closures and map fusion, as depicted in the detailed block diagram in Fig. 4.5. Further modular details of the server are presented below.



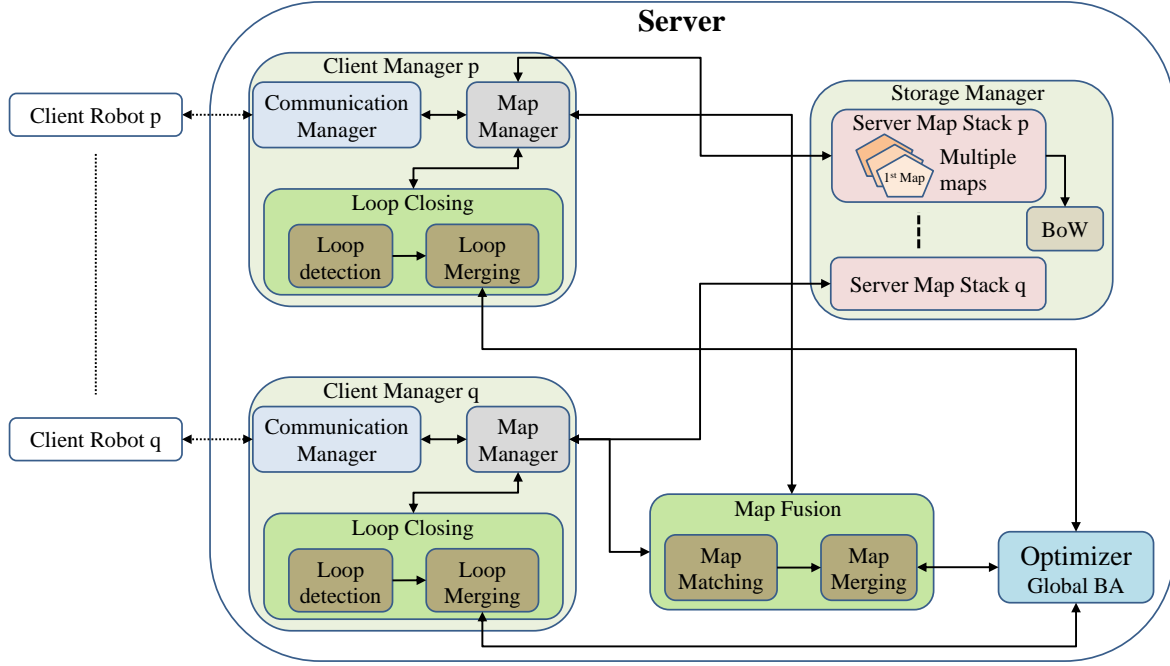


FIGURE 4.5: Internal modular design of the Server in CORB2I-SLAM.

#### 4.2.5.1 Client Manager

The server has separate client managers for all connected clients. A client manager mainly communicates with the corresponding client for receiving and sending map information. The client manager has a map manager, which is the interface for storing and retrieving the corresponding map. The client manager contains an intra-map place recognizer module, which checks the matches between non-neighboring KFs of the map using the Bag-of-Words (BoW) [117] technique. These matches between KFs indicate the existence of a loop in the map. The detected loops are adjusted by establishing new covisibility edges between the corresponding KFs, followed by a global BA. The server system consists of a client manager for each client, a storage manager, and map fusion and optimization modules.

#### Map Structure

The server has separate map stacks for each client and stores received maps into the corresponding map stacks. The KFs of map structures in the server have only covisibility constraints because the IMU constraints are not transmitted to the server. The server optimizes maps and sends back the updated map to the corresponding clients whenever necessary. Therefore, when an old KF is sent back by the server and is inserted into the local map on the client, it only contains covisibility constraints. Thus, the local map of a client can also be updated with the updated pose of an old KF by the server.



The global BA on the server is a vision-only BA because the server map contains only covisibility constraints. The scale is kept fixed in the global BA when any contributing client runs VIO.

#### 4.2.5.2 Storage Manager

The storage manager stores the maps of every client, allowing the access to the map manager of the corresponding client manager for necessary modifications to the map. The map manager generates a BoW model for every KF, where BoW is a way of representing image features in a vocabulary tree that discretizes a binary descriptor space, speeding up correspondences for geometrical verification. Therefore, BoW helps in finding places by establishing geometrical matches between features from multiple KFs. We utilize BoW for place recognition in detecting loops in a map or identifying matches between two different maps.

#### 4.2.5.3 Map Fusion

The place recognizer module utilizes the place-recognition technique, BoW, and generates a pair of matched KFs along with the associated matched MPs from two different maps, either from the same client or from multiple clients. The fusion module is well aware of each client's sensors for estimating VIO or VO; therefore, it consistently fuses maps from a non-metric scale to a metric scale.

Let us assume  $F^s$  and  $F^d$  are the matched KFs from the maps  $M^s$  and  $M^d$ . The matched MPs generate one set of 3D–3D point correspondences from  $M^s$  to  $M^d$  and two sets of 3D–2D point correspondences from  $M^s$  to  $F^d$  and from  $M^d$  to  $F^s$ . There can be three situations: (1) both the maps are in the metric scale, (2) only map  $M^d$  is in the metric scale, and (3) no maps are in the metric scale.

**Case 1:** The 3D–3D correspondence set generates a  $SE(3)$  transformation [5], [13]  ${}^{M^d}\mathbf{T}\mathbf{1}_{M^s}$  and the 3D–2D point correspondence sets generate two more  $SE(3)$  transformations  ${}^{M^d}\mathbf{T}\mathbf{2}_{M^s}$  and  ${}^{M^s}\mathbf{T}\mathbf{3}_{M^d}$ . Finally, an average  $SE(3)$  transformation is formulated by rotation averaging [118] and translation averaging [119]. The rotation averaging problem is to find the best average rotation from several estimations of a single rotation and can be formulated as in Equ. (4.2).

$$\arg \min_{R \in SO(3)} \sum_{i=1}^n dist(R_i, R)^e \quad (4.2)$$

where,  $R_1 \dots R_n$  are the multiple estimations of the rotation in the rotation space  $SO(3)$  (the group of all 3-dimensional rotations),  $e$  is an exponent ( $e \geq 1$ ), and

$dist(P, Q)$  represents a distance function between two rotations  $P$  and  $Q$ . The translation averaging is to find the best average translation from multiple estimations of a single translation. We use a bilinear formulation with a rotation-assisted Iterative Re-weighted Least Squares scheme for translation averaging, as proposed in [119].

**Case 2:**  $M^d$  is only aligned in metric scale, meaning 3D–3D correspondences are not in the same scale. Therefore, we calculated the scale difference from  $M^s$  to  $M^d$  using Equ. (4.3).

$$s = \frac{1}{n} \sum_{i,j \in \psi} \frac{E_{M_i^d, M_j^d}}{E_{M_i^s, M_j^s}} \quad (4.3)$$

where,  $\psi$  is the 3D–3D point correspondence set, and  $E_{M_i^x, M_j^x}$  denotes the Euclidean distance between 3D points  $i$  and  $j$  in map  $M^x$ . The scaled map  $\hat{M}^s = sM^s$  and  $M^d$  generate a  $SE(3)$  transformation [5], [13]  ${}^{M^d}\mathbf{T}_{\hat{M}^s}$  similar to the previous case. The 3D–2D point correspondence sets generate two more  $Sim(3)$  [13] transformations,  ${}^{M^d}\mathbf{S}_{2M^s}$  and  ${}^{M^s}\mathbf{S}_{3M^d}$ , and we calculate an average  $Sim(3)$  transformation by rotation averaging and translation averaging.

**Case 3:** We follow a similar process as proposed in CCM-SLAM [21].

These calculated transformations in all three cases allow us to create a new map structure  $M^f$ , where map  $M^s$  is inserted after using the transformation and  $M^d$  enters directly. Both clients obtain access to the fused map  $M^f$ . Global BA [110] runs after map fusion. Fig. 4.3(c)–(e) show an accurate map fusion using the proposed map fusion module. Fig. 4.3(d)–(e) are the magnified views of the location P3 from different view points. The alignment of the green and white camera tracks ensures an accurate map fusion.

#### 4.2.6 Communication Bandwidth

Any newly created KF and MP generated by a client are promptly shared with the server, while re-transmission of older KFs and MPs takes place when there are updates in poses. The sizes of these messages are detailed in Table 4.1. For a new KF, the message comprises the entire data structure, encompassing 2D features, associated feature descriptors, and 3D map points. The average size for a new KF, considering 1000 feature points, is 56 KB. Similarly, a message for a new MP also includes the complete data structure, with an average size of approximately 200 bytes. In the event of re-transmitting an old KF or MP between the client and server, the sizes are 148 bytes and 52 bytes, respectively, since such re-transmissions exclude old 2D feature points. Additional communication messages between the server and the client are of negligible size, as message passing occurs only in response to specific events. This efficient data

sharing mechanism ensures minimal communication overhead between the client and server, facilitating seamless collaboration in the multi-robot mapping framework.

TABLE 4.1: Size of Communication Messages

Message Type	Message Description	Average Size
New KF	Whole KF data structures (2D features, feature descriptors, 3D map points)	56 KB (considering 1000 feature points)
New MP	Whole MP data structures	200 bytes
Old KF	Updated Poses	148 bytes
Old MP	Updated Position	52 bytes

### 4.3 Experimental Results

We extensively evaluate CORB2I-SLAM on open sequences (EuRoC [120], Freiburg2 [121]) as well as in real autonomy, conducting a total of four categories of experiments. Table 4.2 provides a broad description of all the categories.

TABLE 4.2: Experimental Descriptions

Experimental Objective	Dataset	Camera	Type	Features of the selected data
Evaluating Single agent accuracy	EuRoC [120]	Stereo	Indoor	Single MAV runs multiple times, normal motions and fast motions
Evaluating map merging for homogeneous camera setup	EuRoC	Stereo	Indoor	Single MAV runs multiple times, normal motions and fast motions
Evaluating map merging for heterogeneous camera setup	Freiburg2 [121]	RGB-D	Indoor	Handheld camera motions around a office desk
Map fusion in real autonomy	Real flight	Monocular	Outdoor	Single UAV flies multiple times, fast motion

Each client utilizes a standard laptop with an Intel Core i5-5200 (four cores @ 2.2 GHz) and 4 GB of RAM when executing open sequences. The server employs another laptop equipped with an Intel Core i7-8750H (12 cores @ 2.20 GHz) and 16 GB of RAM. We observed that this server performs without any delay with six active clients, but experiences lag when the number of active clients increases further. In real autonomy,

testing is conducted on a Tarot drone featuring an NVidia Jetson TX2 board paired with an Intel RealSense D435i RGB-D camera. The server remains the same laptop in real autonomy, and communication takes place over a dedicated 4G wireless network. Experimental parameter values are assigned empirically, setting  $N = 30$  KFs in the local map for on-board VIO/VO calculation and 15 KFs for local BA in all our experiments. Errors are estimated as the Root Mean Square of Absolute Translation Error (RMS-ATE) and presented as the average value of 10 executions.

### 4.3.1 Experiment 1: Evaluating Single-Agent Accuracy

We evaluate the fundamental accuracy of CORB2I-SLAM on a single agent, evaluating CORB2I-SLAM both with and without the inclusion of an IMU. We calculate the RMS-ATE and compare it with the SoA VIO or VO-based methods. The details of these comparisons are presented in Table 4.3, where the values for CVI-SLAM [26] are obtained from the author’s publication due to its closed-source nature. Prior to RMS-

TABLE 4.3: The RMS-ATE (meter) of single agent for experiments on EuRoC sequences [120].

EuRoC [120] Sequence No.	VINS [80] (MO + IM)	CCM [21] * (MO)	CVI [26] (MO + IM)	ORB-SLAM3 [84] (ST + IM)	CORB2I (ST)	CORB2I (ST + IM)
MH01	0.120	0.113	0.085	0.036	<b>0.034</b>	0.035
MH02	0.120	0.089	0.063	<b>0.033</b>	0.037	0.034
MH03	0.102	0.078	0.065	<b>0.035</b>	0.036	0.036
MH04	0.155	0.138	0.293	0.051	0.133	<b>0.045</b>
MH05	0.136	0.129	0.081	0.082	0.078	<b>0.059</b>
V103	0.190	–	–	0.024	0.048	<b>0.023</b>
V203	0.220	–	–	0.024	0.129	<b>0.022</b>

\*: Trajectories are scaled off-line because the metric scale is not present,  
MO: Monocular, ST: Stereo, IM: IMU.

ATE calculation, the estimated trajectories are aligned using an  $SE(3)$  transformation [5], [13]. The symbol ‘–’ denotes instances where the open-source implementation either does not support a specific configuration or is absent from the author’s publication. Vision-only CORB2I-SLAM experienced three tracking failures while executing the V203 sequence, resulting in the creation of new maps. Consequently, the value in Table 4.3 is reported after the merging of all four maps. ORB-SLAM3 [84] exhibits marginally superior accuracy in certain cases, attributed to its more effective loop corrections. The accuracy of CORB2I-SLAM (without IMU) indicates that the inclusion of an IMU is not universally beneficial, as a noisy IMU measurement can amplify localization noise. However, the CORB2I-SLAM framework is adaptive to detect and

reject highly noisy IMU measurements. In conclusion, we find that the performance of CORB2I-SLAM is either comparable to ORB-SLAM3 or surpasses SoA methods.

### 4.3.2 Experiment 2: Evaluating Map Fusion Accuracy for Homogeneous Camera

We evaluate the accuracy of map fusion using EuRoC sequences [120], where multiple clients operate concurrently. Table 4.4 provides a comparative analysis with SoA methods, revealing a notable improvement in accuracy with the merged map. This

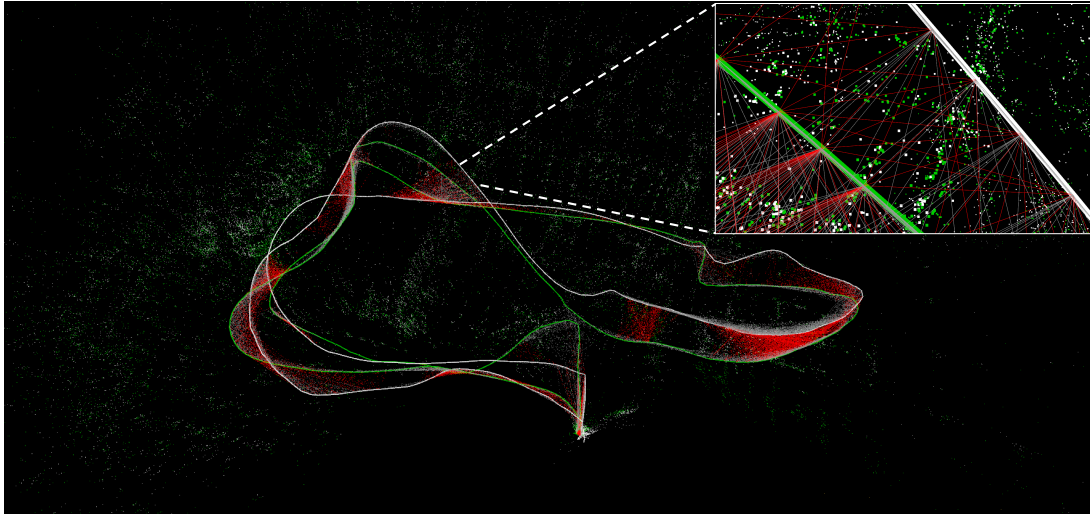
TABLE 4.4: The RMS-ATE (meter) of multiple agents for experiments on EuRoC sequences [120].

EuRoC [120] Sequence No.	VINS [80] † (MO + IM)	CCM [21] * (MO)	CVI [26] (MO + IM)	ORB-SLAM3 [84] † (ST + IM)	CORB2I (ST)	CORB2I (ST + IM)
MH01,02	0.159	0.097	0.050	0.035	0.036	<b>0.028</b>
MH01,02, 03	0.192	0.092	–	0.037	0.035	<b>0.029</b>
MH01,02, 03,04	0.239	0.079	–	0.051	0.069	<b>0.034</b>
MH01,02, 03,04,05	0.278	–	–	0.086	0.052	<b>0.035</b>

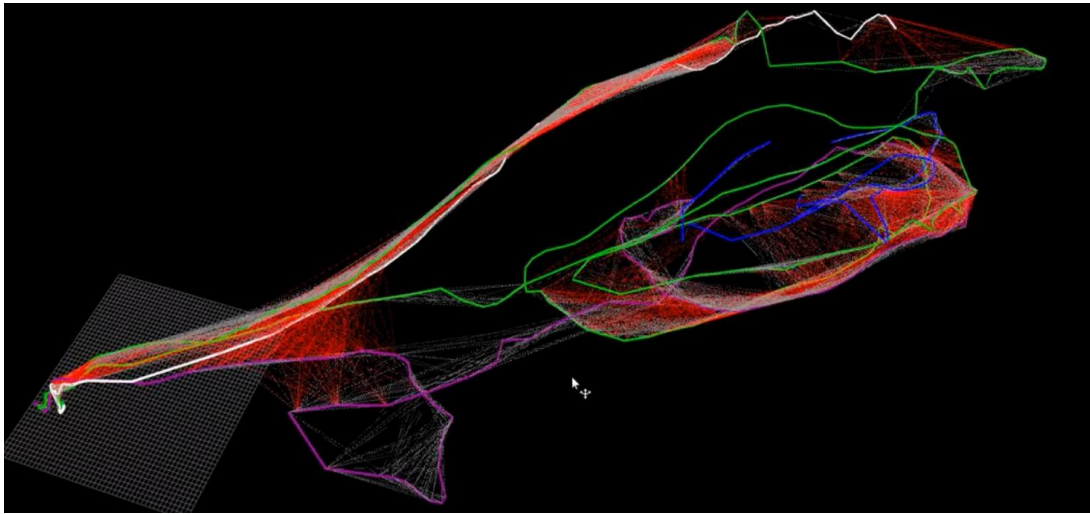
†: the sequences are run sequentially because the system is designed for a single agent,  
\*: Trajectories are scaled off-line because the metric scale is not present, **MO**: Monocular, **ST**: Stereo, **IM**: IMU.

enhancement is attributed to our innovative multi-constrained map fusion approach and a robustly constrained loop-closure correction. Collaborative information sharing among clients contributes to more precise localization. The comparison unequivocally demonstrates an increase in accuracy through collaboration, as compared to the values in Table 4.3.

Fig. 4.6 visually depicts the results of two experiments pictorially. Fig. 4.6(a) presents a snapshot of the first experiment on EuRoC MH04 and MH05 sequences, running VIO on two different clients. The camera trajectories and MPs of MH04 and MH05 are represented in white and green, respectively. The camera trajectories are generated with a series of KFs from the corresponding clients. Any two KFs are connected with a covisibility edge only if both KFs observe at least a single MP. Two types of covisibility edges are depicted: white edges represent connected KFs from one client, while red edges represent connections between KFs from multiple clients. The inset provides the magnified view of the merged map. Fig. 4.6(b) displays a snapshot of the second experiment on EuRoC MH01 and MH02 sequences running on two different



(a)



(b)

FIGURE 4.6: Snapshots of the CORB2I-SLAM map fusion experiment on EuRoC sequences [120]: (a) Map fusion among multiple agents, (b) Map fusion among sub-maps of a single agent and multiple maps of multiple agents.

clients. The client executing the MH01 sequence has limited computing power and fails to track twice due to frame skipping, but reinitializes immediately. The camera trajectories of sub-maps are shown in white, blue, and purple. CORB2I-SLAM merges all sub-maps.

### 4.3.3 Experiment 3: Evaluating Map Fusion Accuracy for Heterogeneous Camera

Experiment 3 comprises two distinct tests, namely Test 1 and Test 2, utilizing sequences from the Freiburg2 dataset [121]. In both tests, involving two clients, namely  $Client_1$  and  $Client_2$ , the same data sequences are employed, sequence  $fr2/xyz$  on  $Client_1$  and sequence  $fr2/rpy$  on  $Client_2$ . In Test 1,  $Client_1$  executes VO solely on monocular images, specifically selecting RGB images from RGB-D data. Conversely, in Test 2, VO is conducted on RGB-D images. Quantitative details of these experiments are outlined in Table 4.5. The computed RMS-ATE values for Test 1 and Test 2

TABLE 4.5: Quantitative details for experiment on Freiburg2 sequences.

	Test 1		Test 2	
	$Client_1$	$Client_2$	$Client_1$	$Client_2$
Freiburg2 [121] Sequence Names	$fr2/xyz$	$fr2/rpy$	$fr2/xyz$	$fr2/rpy$
Sensor Types	<b>Monocular</b>	RGB-D	<b>RGB-D</b>	RGB-D
Merging time offset (sec)	<b>31.84</b>		<b>46.27</b>	
Single agent RMS-ATE (m)	0.002438	0.033917	0.004600	0.033917
CORB2I-SLAM RMS-ATE (m)	<b>0.008357</b>		<b>0.034113</b>	

are 0.008357 and 0.034113, respectively, indicating a noteworthy improvement in localization accuracy for Test 1 in comparison to Test 2. This heightened accuracy in Test 1 can be attributed to the early occurrence of map fusion, transpiring at 31.84 seconds, in contrast to Test 2 where map fusion takes place at 46.27 seconds, as indicated by the ‘Merging time offset’ in Table 4.5. These results underscore the substantial enhancement in localization estimation associated with quicker map fusion or efficient information sharing in the collaborative framework. The findings highlight the significance of timely collaboration and information exchange in multi-robot mapping scenarios.



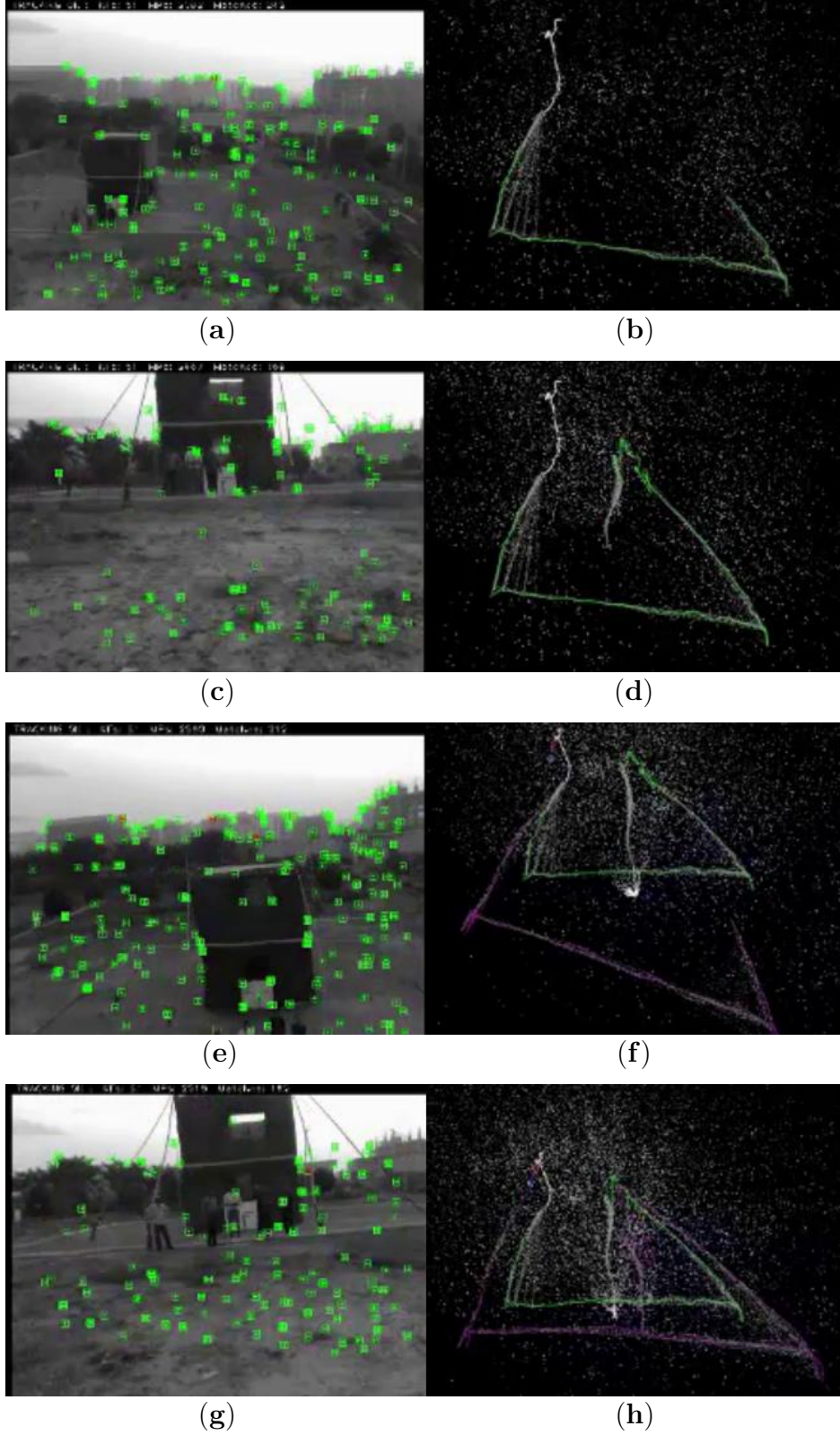


FIGURE 4.7: Test on real autonomy: Each column shows the camera view and corresponding map, (a, c) display camera views of  $Client_1$ , (e, g) show camera views of  $Client_2$ , (b, d) illustrate the camera trajectories of  $Client_1$ , (f) depicts the camera trajectories of both  $Client_1$  and  $Client_2$  before fusion, (h) presents the entire fused map of  $Client_1$  and  $Client_2$ .



#### 4.3.4 Experiment 4: Evaluating Map Fusion in Real Autonomy

This experiment aims to validate the performance of the proposed system on a real robotic platform, emphasizing on real-time operation, a primary objective for collaborative SLAM in actual robot applications. A Tarot drone, navigating through pre-determined GPS way points outdoors, was employed for the experiment. Due to the observed inaccuracies in GPS point-based navigation, GT verification was omitted. Two clients were run sequentially on a single drone, starting 3 meters apart with each trajectory spanning approximately 55 meters. The experiment evaluated the proposed adaptive VIO or VO selection mechanism. A noisy MEMS-based IMU was employed to test the framework, with unsuccessful IMU-based initializations leading to continuous use of monocular VO. Opting for monocular over RGB-D accommodated the need for long-range depth, surpassing the sensing capabilities of the RealSense D435i. Despite multiple camera tracking failures due to fast rotations, successful re-initializations were achieved. Notably, the framework fused sub-maps of *Client*<sub>2</sub> with the largest sub-map of *Client*<sub>1</sub> after 11.3 seconds of *Client*<sub>2</sub> execution, demonstrating the system’s real-time capabilities on an actual robotic platform. Fig. 4.7 shows the intermediate trajectories and the fused trajectories with MPs. In summary, this experiment successfully validates the proposed system’s real-time performance on a Tarot drone, emphasizing its adaptability to real-world challenges, such as GPS inaccuracies and fast rotations, while showcasing its robust sub-map fusion capabilities between clients.

#### 4.3.5 Execution Time

We evaluate CORB2I-SLAM execution on EuRoC MH01 to MH04 sequences. The on-board execution of any client is independent of the total number of participating clients; therefore, the execution time is similar for single and multiple clients. The tracking thread takes on average 36 ms, the mapping thread takes 240 ms to 245 ms, and communication takes about 0.28 ms. Server performance changes with the number of KFs to be processed for an operation. Fig. 4.8 shows the average execution time for computing the transformation for a loop closing and map merging.

### 4.4 Conclusions

We present a novel architecture of a centralized collaborative SLAM framework for heterogeneous vision sensors and inertial sensors. In this centralized architecture, client robots are considered with limited computation capabilities and a central server with

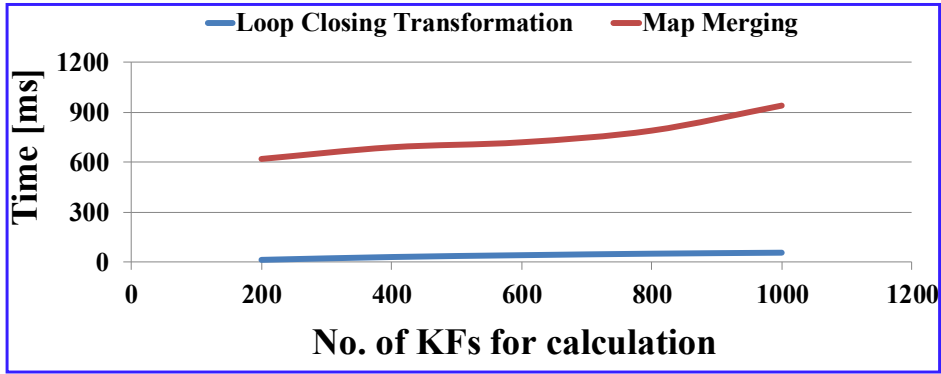


FIGURE 4.8: Server execution time with respect to the number of KFs to be processed. The values are tested on EuRoC MH01 to MH04 sequences with an average of 5 executions.

higher computation capabilities. The framework allows the clients to run either VO or VIO independently without any server dependency, and it is designed to adapt to detect noisy inertial sensors and exclude them in pose estimation. We proposed a new criterion to estimate the accuracy of poses and reinitialize with a sub-map in the case of tracking being inaccurate. The sub-maps can be fused into a single map once a match is found among multiple maps. We also propose a novel map-merging procedure between a non-metric scale map and a metric scale map that produces better accuracy compared to SoA techniques. We evaluate our proposed framework extensively on open datasets as well as in real flight and show better accuracy. The CORB2I-SLAM may not have better accuracy on sudden illumination changes or low-light conditions. We keep such enhancements in the scope of future work.

The proposed collaborative VI-SLAM framework and our proposed obstacle detection algorithms from RGB-D sensor (details are in Chapter 3) together are capable to support multiple clients to navigate autonomously. We have studied some of the limitations related to RGB-D sensing technology in Section 3.4, where operating depth range is a major problem for sensing fast moving obstacles from safe distances. This motivate us to explore other sensors with long range accurate sensing for dynamic obstacles detection. The next chapter shall talk about the extension of our previous obstacle detection system to alleviate some of the problems.

## Chapter 5

# Obstacle Detection and Tracking using 3D LiDAR

### 5.1 Introduction

Autonomous driving and navigation of robots have made significant progress in the last couple of decades, with researchers contributing scientific advancements in various research domains like computer vision, machine learning, path planning, and control theory, etc. Autonomous navigation broadly associates three main modules:

- Perception Module: This module utilizes multiple sensors to capture the surroundings and create an internal model. The internal model helps to understand obstacles and their states (i.e., static or dynamic, dimensions, velocities, etc.).
- Planning Module: The planning module is responsible for generating a feasible path from source to destination, avoiding occupied spaces, and generating motion plans for the estimated paths.
- Control Module: This module generates control commands based on motion plans to move the robot.

In this work, our primary focus is on the perception module, specifically on the estimation of a mobile robot's surroundings in terms of obstacles. Mobile robots often encounter the need to comprehend and track multiple dynamic obstacles in complex and cluttered environments, requiring the ability to respond promptly to avert potential collisions. Researchers predominantly emphasize vision-based sensors such as stereo cameras [34]–[40], RGB-D cameras [2], [4], [41], [51], [52], [55]–[57], and LiDAR [3], [122]–[125] for perceiving the surroundings. Chapter 2 provides a comprehensive literature survey on estimating obstacles using stereo and RGB-D cameras. In Chapter 3, we propose an efficient solution for obstacle detection and tracking utilizing only depth images from RGB-D sensors, referred to as EODT [2]. Chapter 4 focuses on deploying

multiple heterogeneous robots with diverse vision sensors [109], enabling collaborative creation of the perception of the surrounding environment rapidly. Stereo cameras exhibit limitations, including a small base-line, being sensitive to calibration errors, susceptibility to illumination changes, and challenges in night driving scenarios [126]. Conversely, RGB-D cameras face constraints such as a limited operating depth range (typically less than 3 meters), depth corruption in sunlight, intense lighting conditions, low light scenarios, sudden illumination changes [127]–[129], etc.

In this context, LiDAR emerges as the most accurate and reliable sensor due to its insensitivity to weather conditions, capability to operate under both sunlight and poor illumination conditions, and its proficiency in providing precise depth estimations over long ranges. A 2D LiDAR emits a single beam of light on a horizontal plane and collects data along the X and Y axes. Conversely, a 3D LiDAR emits multiple beams of light along the X, Y, and Z axes. Typically, LiDAR sensors spin to gather comprehensive information about their surroundings. 2D LiDAR sensors are best suited for detection activities, while 3D LiDAR sensors are suitable for landscape mapping and scanning purposes. It is crucial for obstacle detection and processing to be both accurate and computationally efficient for real-time onboard execution. However, existing approaches have limitations in effectively handling obstacles across diverse scenarios.

Leveraging the advantages of LiDAR sensors, we introduce a dynamic obstacle detection system utilizing 3D LiDAR technology based on the research findings presented in [130]. In our present work, we focus on a 3D LiDAR sensor mounted on a mobile robot, capturing 3D measurements in the form of point clouds. These point clouds are then processed by the perception module to generate a consistent and meaningful representation of the surrounding environment. Traditional point cloud-based approaches typically utilize raw sensor data, providing accurate representations but demanding substantial memory and computational power [126]. Consequently, a key objective of this work is to mitigate computational intensity by circumventing high-computing algorithms like ground plane segmentation and 3D clustering. The existing approach in EODT [2] formulates u-depth map and restricted v-depth map representations from depth images of an RGB-D camera. These multiple depth map representations produce accurate obstacle estimation with reduced processing time. However, the technique has never been explored for LiDAR point clouds. Inspired by [2], in this work, we adapt and apply these depth map representations to 3D LiDAR point clouds for enhancing obstacle detection, estimation, and tracking using 3D LiDAR point clouds. The system actively tracks obstacles positioned in front of the autonomous robot, posing potential obstructions to navigation. Additionally, the proposed system incorporates a self-localization module to precisely localize the robot and estimate obstacle positions

within a fixed world coordinate frame.

The rest of this chapter is organized as follows: Section 5.2 presents a literature survey of obstacle detection systems using LiDARs. Section 5.3 describes the architecture of the proposed system and contributions. Section 5.4 presents experimental evaluations on various datasets and real-time execution. Finally, Section 5.5 concludes the chapter.

## 5.2 Literature Survey

We conducted a literature survey on obstacle detection using stereo and RGB-D cameras in Section 2.1. Consequently, we now present a literature survey specifically focusing on obstacle detection utilizing only LiDAR sensors.

Lanxiang *et al.* [122] present an obstacle detection system for UAVs employing 2D LiDAR. Their method corrects LiDAR-derived point clouds effectively, utilizing a clustering algorithm to identify obstacles in rectified point clouds based on relative distance and density. Alireza *et al.* [126] propose an obstacle detection system using 3D LiDAR. In this approach, captured point clouds are transformed into dense point clouds, and a piece-wise surface fitting algorithm estimates a finite set of surfaces fitting the road. Subsequently, voxelization is applied to the remaining point clouds, and discriminative analysis based on 2D counters is used to identify static and dynamic obstacles. However, a major drawback of the system is its reliance on computationally intensive processes, including road segmentation using dense point clouds. Desheng *et al.* [123] present an obstacle detection and tracking method for AGVs. They segment the road from point clouds and use an eight-neighbor clustering algorithm to identify obstacles, subsequently tracking them with a Kalman filter. Chien *et al.* [3] introduce an autonomous robot system utilizing both LiDAR and a camera. LiDAR-derived point clouds are employed for ground removal and subsequent clustering. The clustered point clouds are projected onto the 2D camera frame to identify the region of interest (ROI) for obstacles. Guidong *et al.* [131] compared geometric-based and deep learning-based obstacle detection methods. The geometric-based method employs geometric and morphological operations on 3D points to retrieve obstacles. Marius and Florin [125] present a facet-based shaped obstacle detection system, capturing point clouds from a 64-layer LiDAR, detecting ground points using a normal-based geometric approach, and identifying obstacles using an RBNN-based clustering method. Recently, Heng and Xiaodong present a vehicle detection and tracking system using LiDAR point clouds. Their approach involves preprocessing captured point clouds to filter out noise,

proper alignment in a single coordinate frame, clustering into smaller groups, and vehicle identification using a trained classifier by support vector machine. The Kalman filter and global nearest neighbor algorithms are employed for vehicle tracking. Very recently, Pengwei *et al.* [1] detail an obstacle detection system for intelligent vehicles. The captured point clouds are initially filtered, followed by ground plane segmentation and obstacle identification using an improved DBSCAN-based point cloud clustering algorithm. It is noteworthy that all existing systems rely on computationally intensive modules such as ground plane segmentation and clustering for obstacle detection, rendering them unsuitable for vehicles with limited computational capabilities.

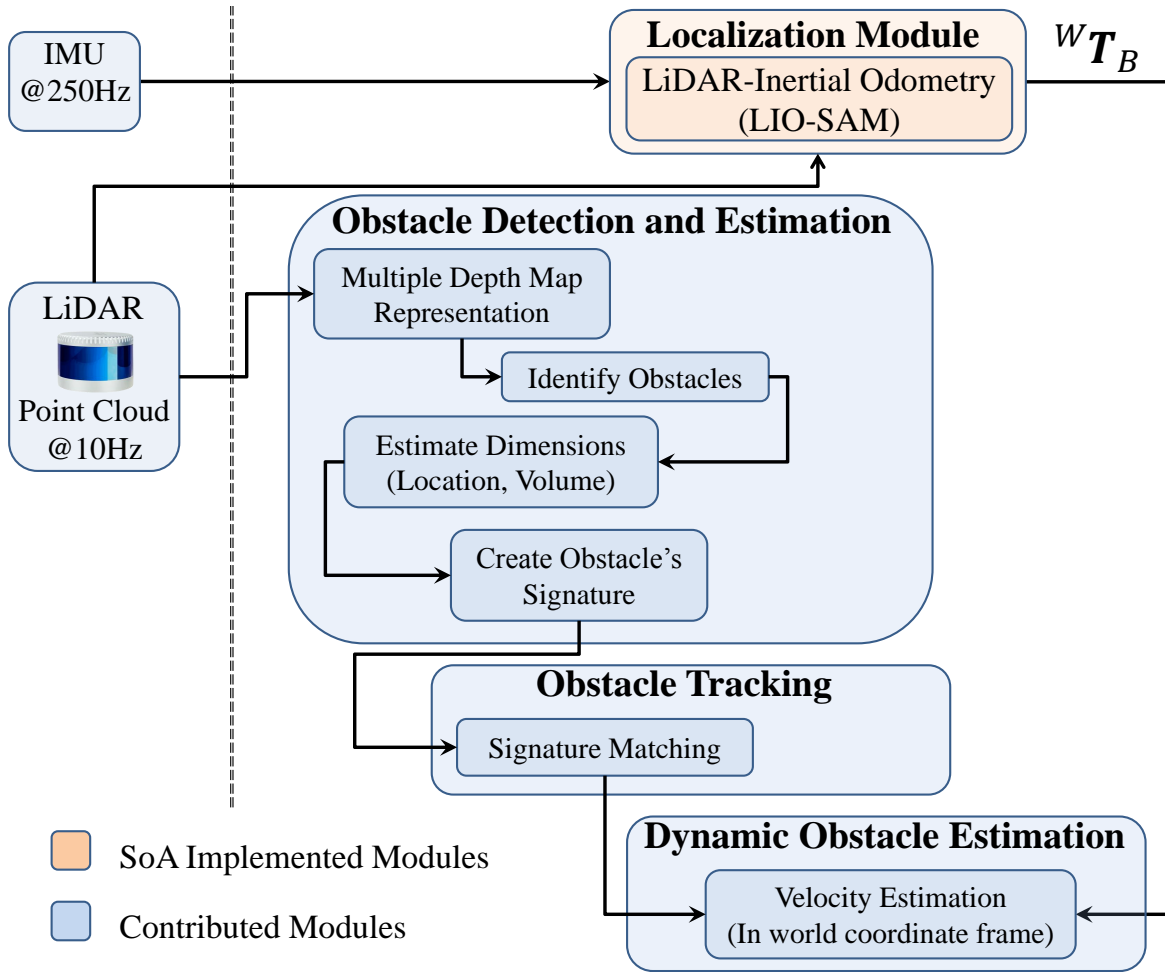


FIGURE 5.1: Block diagram of the proposed system for dynamic obstacle detection and tracking using 3D LiDAR.

## 5.3 System Design

Fig. 5.1 illustrates our proposed system, which takes inputs from two sensors, Velodyne 3D LiDAR and IMU, and has four main modules, a self-localization module, an obstacle detection and estimation module, an obstacle tracking module, and a dynamic obstacle estimation module. We use the state-of-the-art LiDAR-inertial based odometry estimation system, LIO-SAM [132], as the self-localization module. It receives IMU measurements and captured point clouds from the LiDAR and estimates the robot poses (transformations of the base\_link coordinate frame) with respect to the fixed world coordinate frame  $W$ . LIO-SAM achieves this by tightly-coupled LiDAR-inertial odometry estimation through smoothing and mapping and we refer to [132] for a detailed description of LIO-SAM. Let  ${}^W\mathbf{T}_B$  denotes the estimated robot pose at any given time instance by the localization module.

The proposed system employs multiple coordinate frames for capturing different sensor data. We assume that point clouds are captured using Velodyne 3D LiDAR, captured point clouds are rectified and aligned with the Velodyne coordinate frame  $L$ . The gravity direction and biases of the accelerometer and gyroscope are estimated in the base\_link coordinate frame  $B$ , representing the coordinate frame of the IMU. Additionally, the X-axis of the Velodyne coordinate frame  $L$  is assumed to point towards the frontal direction of the robot. Fig. 5.2 displays the coordinate frames on a Husky robot [102] simulated in the Gazebo environment [95] and visualized in the Rviz visualizer [99]. The red arm represents the X-axis and it is pointing towards the front direction of the Husky. A fixed transformation exists from the Velodyne coor-

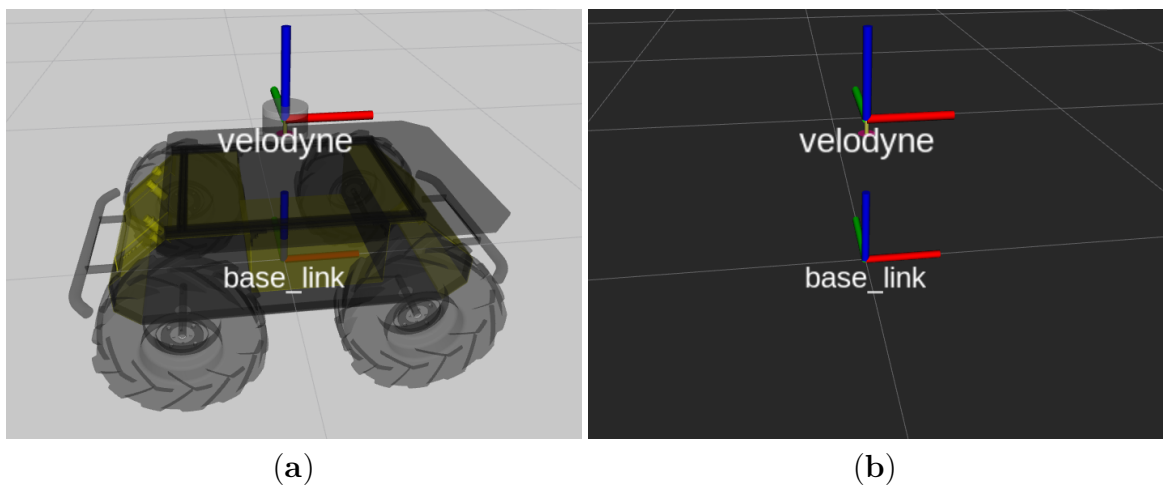


FIGURE 5.2: (a) Visualization of coordinate frames (base\_link and Velodyne) on a Husky robot [102] in Rviz [99], (b) Isolated display of coordinate frames without the robot for clarity, with red, green, and blue arms representing the X, Y, and Z axes, respectively.



dinate frame  $L$  to the base\_link coordinate frame  $B$ , and denoted as  ${}^B\mathbf{T}_L$ . We use the (X, Y, Z) notation to represent a 3D coordinate where red, green, and blue colors represent the X, Y, and Z axes, respectively.

The contributed modules of the present system include (i) an obstacle detection and estimation module, (ii) an obstacle tracking module, and (iii) a dynamic obstacle estimation module.

### 5.3.1 Obstacle Detection and Estimation

We introduced EODT, an obstacle detection and tracking system using an RGB-D sensor, in Chapter 3. In this chapter, we utilize a u-depth map and a newly proposed restricted v-depth map for obstacle detection and tracking. Influenced by the advantages of LiDAR over RGB-D camera, we extend our prior work, EODT, and introduce u-depth and restricted v-depth representations derived from LiDAR point clouds. The primary objectives are to enhance accuracy and reduce processing time. Unlike EODT, which formulates u-depth and restricted v-depth representations from depth images, our present method utilizes LiDAR point clouds to derive these depth representations without having any standard camera setup. This represents a significant contribution in this work. We discuss the details of u-depth and restricted v-depth formulations in the following two sub-sections and provide insights into detecting and estimating obstacles using these multiple depth representations.

#### 5.3.1.1 Virtual Camera Coordinate Representation

A u-depth map is a column-wise histogram representation of depth values taken from a depth image. In the case of EODT, the system employs an RGB-D camera, where the u-depth maps are constructed on the same image plane as the depth image. The present system includes only a 3D Velodyne LiDAR sensor, which has a horizontal FOV of  $360^\circ$ , therefore, it captures a  $360^\circ$  view around the sensor in the form of 3D point clouds.

In the absence of any camera sensor in the present system, we create a virtual camera to form u-depth and restricted v-depth representations and denote the virtual camera coordinate frame as  $C_v$ . The camera coordinates follow the right-hand rule, where the viewing direction is along the Z-axis. The origin of the coordinate frame  $C_v$  coincides with the origin of the coordinate frame  $L$ , implying that the coordinate frame  $C_v$  can only be obtained by rotating the coordinate frame  $L$ . Fig. 5.3 illustrates the rotations pictorially for better understanding. Fig. 5.3(a) depicts the coordinate frame  $L$ , in which we apply a rotation of  $+90^\circ$  along the X-axis, which rotates the

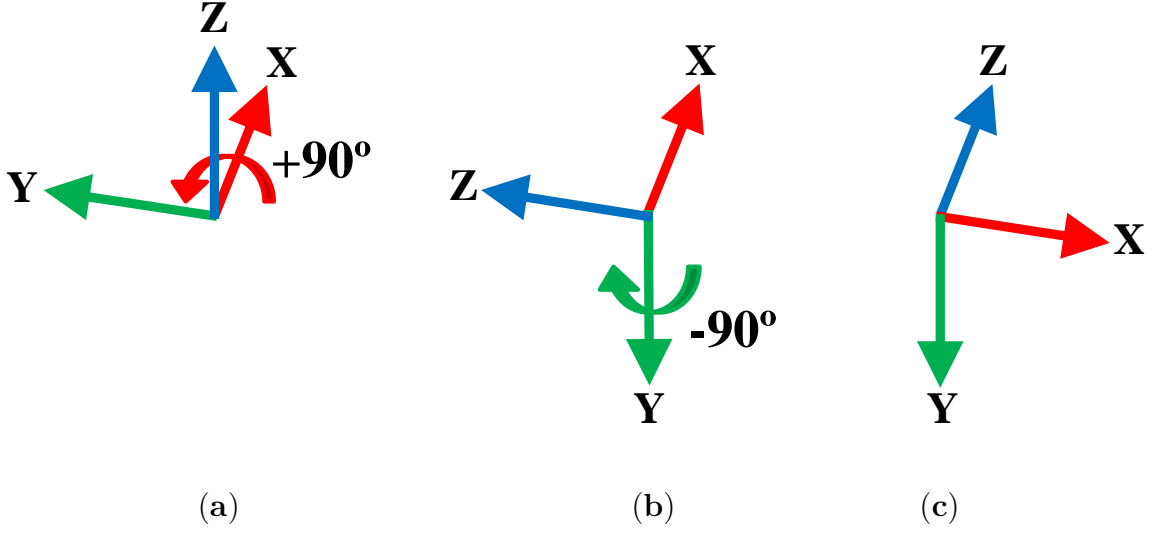


FIGURE 5.3: Transformation from the Velodyne coordinate frame  $L$  to the virtual camera coordinate frame  $C_v$ : (a) The Velodyne coordinate frame  $L$ , (b) An intermediate coordinate frame achieved with  $+90^\circ$  rotation along the X-axis of  $L$ , (c) The virtual camera coordinate frame  $C_v$  is obtained by rotating the intermediate coordinate frame  $-90^\circ$  along the Y-axis.

YZ plane and changes the directions of the Y and Z axes. This rotation creates an intermediate coordinate frame, as shown in Fig. 5.3(b). Thereafter, we again apply another rotation of  $-90^\circ$  along the Y-axis of the intermediate coordinate frame, which rotates the XZ plane of the intermediate coordinate frame and changes the directions of the X and Z axes. This rotation finally creates the camera coordinate frame  $C_v$ , as shown in Fig. 5.3(c). Let us assume  ${}^L PC$  is the point cloud captured in the coordinate frame  $L$ , and  ${}^{C_v} T_L$  is the transformation from the coordinate frame  $L$  to the coordinate frame  $C_v$ . The transformation  ${}^{C_v} T_L$  allows the captured point clouds  ${}^L PC$  to transform from the coordinate frame  $L$  to the coordinate frame  $C_v$  using Equ. (5.1).

$${}^{C_v} PC = {}^{C_v} T_L ({}^L PC) \quad (5.1)$$

where,  ${}^{C_v} PC$  is the transformed point cloud into the virtual camera coordinate frame  $C_v$ .

The camera coordinate frame assumption is based on the axes of the coordinate frame  $L$  and the robot's moving direction. We consider only a single camera with a wide horizontal FOV, looking towards the forward motion of the robot because we consider detecting obstacles in front of the robot that may obstruct its motion. We create both the u-depth and restricted v-depth representations on the corresponding 2D image plane. Any obstacle present on the back side of the robot remains undetected in the present system. The virtual camera coordinate frame can be considered in different ways to view towards different directions as per the environmental requirements, or even

multiple virtual cameras, looking in multiple directions, can also be considered to detect obstacles in multiple directions together.

First, we consider a virtual 2D image plane with a 2D coordinate frame  $I_v$  for the virtual camera coordinate frame  $C_v$ . The resolution of the image plane is  $h_{I_v} \times w_{I_v}$ . The resolution of the corresponding u-depth representation is  $n \times w_{I_v}$ , where  $n$  is the number of histogram bins. The depth range of each histogram bin is  $\frac{Max_d}{n}$ , where  $Max_d$  is the maximum depth range inside which obstacles should be detected. The value of  $Max_d$  must not be greater than the maximum range of the LiDAR sensor. Let us also assume, the focal lengths of the virtual camera are  $f_h$  and  $f_v$  in the horizontal and vertical directions of the 2D virtual image plane, respectively, and  $(c_h, c_v)$  is the principal point of the virtual camera. We can choose the values of  $h_{I_v}$ ,  $w_{I_v}$ ,  $f_h$ , and  $f_v$  freely as per the required FOV because the camera is a virtual one. The horizontal FOV ( $FOV_h$ ) and vertical FOV ( $FOV_v$ ) calculations follow Equ. (5.2), where a smaller focal length means a greater FOV.

$$\begin{aligned} FOV_h &= 2 \times \arctan\left(\frac{w_{I_v}}{2f_h}\right) \\ FOV_v &= 2 \times \arctan\left(\frac{h_{I_v}}{2f_v}\right) \end{aligned} \quad (5.2)$$

### 5.3.1.2 U-depth Representation

Now, we describe the formulation of u-depth representation on this virtual image plane and subsequently identify obstacles with their dimensions. In the generation process of u-depth representation, we first project 3D points from the point cloud  ${}^{C_v}PC$  to the 2D virtual image plane and calculate the row index ( $D_r$ ) and column index ( $D_c$ ) of each 3D point. We only accept the points that come within the FOV of the virtual camera, meaning, they are projected within the virtual image dimensions. In the u-depth representation, the column index of the 3D point must be the same as  $D_c$  because of column-wise histogram representation, but the row index ( $D'_r$ ) is calculated from the depth of the 3D point. Finally, the value of the location  $(D'_r, D_c)$  in the u-depth representation is incremented. This entire calculation is performed on all the 3D points present in the point cloud  ${}^{C_v}PC$ , and the u-depth representation is normalized between 0 to 255 before further processing. The detailed formulation of the u-depth representation from the point cloud  ${}^{C_v}PC$  at the time instance  $t$  is presented in Algorithm 3: U-Depth Representation( ), where step 4 and 5 perform the 3D-2D projection, step 6 accepts the points that are within the FOV of the virtual camera, step 7

calculates the row index of the u-depth representation, and finally, step 8 increments the value of the calculated pixel location.

---

**Algorithm 3** : U-Depth Representation( )
 

---

**Input:**  ${}^{C_v}PC$ ,  $h_{I_v}$ ,  $w_{I_v}$ ,  $n$ ,  $Max_d$ ,  $f_h$ ,  $f_v$ ,  $c_h$ ,  $c_v$

**Output:**  $Udepth_{(n \times w_{I_v})}$

```

1: Initialization :  $Udepth_{(n \times w_{I_v})} \leftarrow 0$ ,  $Scale = \frac{255}{h_{I_v}}$ 
2: for  $i = 1$  to  $|{}^{C_v}PC|$  do
3:   if  $(0 < {}^{C_v}PC[i].Z \leq Max_d)$  then
4:      $D_c = \text{roundoff}(\frac{({}^{C_v}PC[i].X)f_h}{{}^{C_v}PC[i].Z} + c_h + 1)$ 
5:      $D_r = \text{roundoff}(\frac{({}^{C_v}PC[i].Y)f_v}{{}^{C_v}PC[i].Z} + c_v + 1)$ 
6:     if  $(1 \leq D_c \leq w_{I_v}) \ \& \ (1 \leq D_r \leq h_{I_v})$  then
7:        $D'_r = \lfloor \frac{(n-1){}^{C_v}PC[i].Z}{Max_d} + 1 \rfloor$ 
8:        $Udepth(D'_r, D_c) = Udepth(D'_r, D_c) + 1$ 
9:     end if
10:  end if
11: end for
12:  $Udepth_{(n \times w_{I_v})} = Udepth_{(n \times w_{I_v})} \times Scale$ 
13: return  $Udepth$ 

```

---

In general, the 3D points on any obstacle attain depth values that are very near to each other. Therefore, they contribute to the same bins in the u-depth representation, and the values of those bins become high and visible as white horizontal patches in the u-depth representation. The presented u-depth representation in Algorithm 3: U-Depth Representation( ) contains histogram bins in a growing order from top to bottom, *i.e.*, a row  $i$  corresponds to a smaller depth than a row  $j$  in the u-depth representation,  $\forall i < j$ . Therefore, any closer obstacle contributes to the upper rows and creates a white patch on the upper rows. The sizes of the white patches correspond to the dimensions of the corresponding obstacles, and we refer to Section 3.2.2.2 for a detailed discussion on the relationship between the 2D size of a white patch and the corresponding 3D size of the obstacle. We perform dynamic binary thresholding as proposed in EODT to make the white patches prominent.

A point cloud from LiDAR is significantly sparser than a point cloud formed from an RGB-D image, and because of this fact, the white patches in a u-depth representation from a LiDAR point cloud are often more broken than white patches in a u-depth map from a depth image. Fig. 5.4 shows this scenario with an example. Fig. 5.4(a) shows a snapshot of a LiDAR point cloud, where a small obstacle is present at the front. Fig. 5.4(b) shows the corresponding thresholded u-depth map, and the obstacle

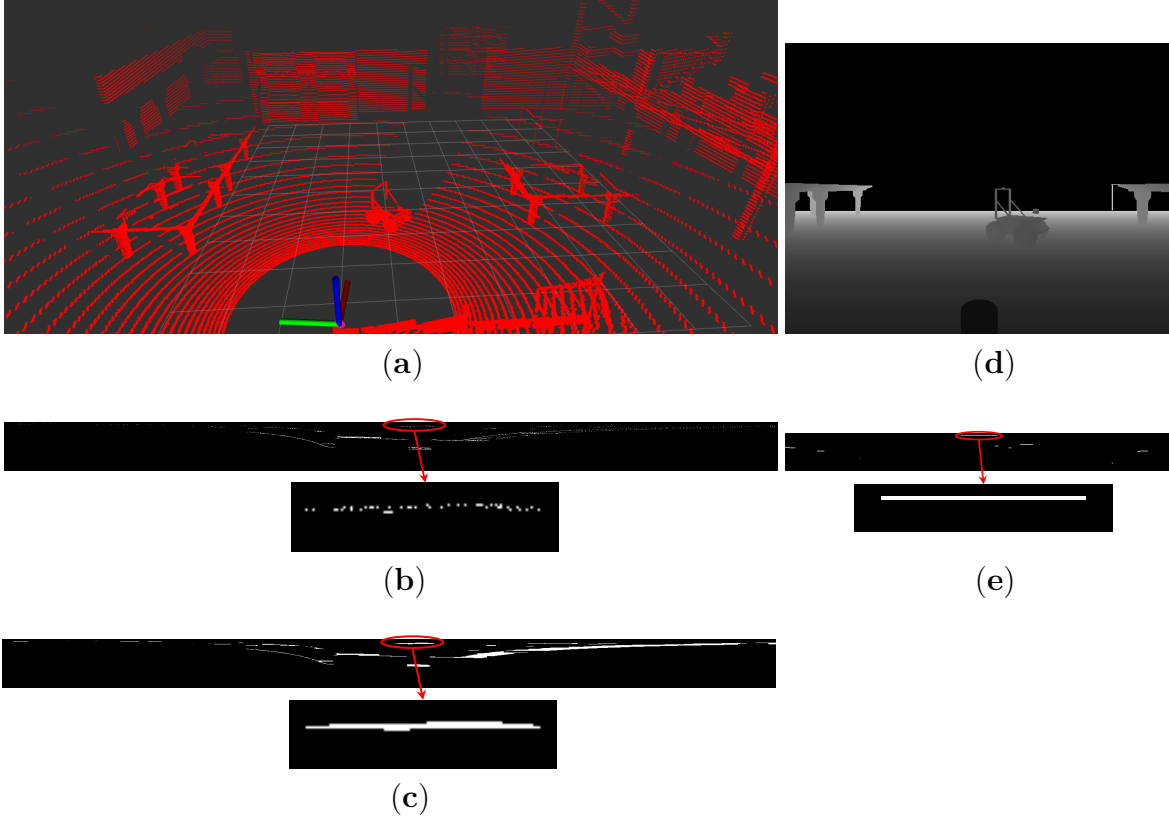


FIGURE 5.4: Comparison of u-depth formation using LiDAR point cloud (a–c) and RGB-D camera (d–e): (a) A sample snapshot of a LiDAR point cloud, (b) Corresponding thresholded u-depth, (c) Corresponding u-depth after closing operation, (d) Depth image of same view from RGB-D camera, (e) Corresponding thresholded u-depth map.

(within the red ellipse) is visible as multiple broken patches. The magnified view in Fig. 5.4(b) shows the broken patches clearly. Fig. 5.4(d) shows the depth image of the same view, which is captured with an RGB-D sensor, and Fig. 5.4(e) is the corresponding thresholded u-depth map. The same object appears as a continuous white path on the thresholded u-depth map, as shown in the magnified view of Fig. 5.4(e). This is a major problem of creating u-depth representations using LiDAR point clouds, and we overcome this issue by using a closing operation [96] with a large horizontal structuring element. We perform a closing operation with a  $(3 \times 50)$  structuring element, which makes these broken patches continuous, as shown in Fig. 5.4(c). Fig. 5.5 shows the visualization of all the steps of making the u-depth representations from the popular KITTI data sequence [133]. Fig. 5.5(a) shows an RGB snapshot of the KITTI data sequence, and Fig. 5.5(b) shows the corresponding point cloud in the Rviz [99] visualizer where the vehicle moving direction is the X-axis (in red) of the Velodyne coordinate frame  $L$ . Fig. 5.5(c) is the virtual depth representation created from the point cloud, Fig. 5.5(d) is the corresponding u-depth representation, and Fig. 5.5(e)

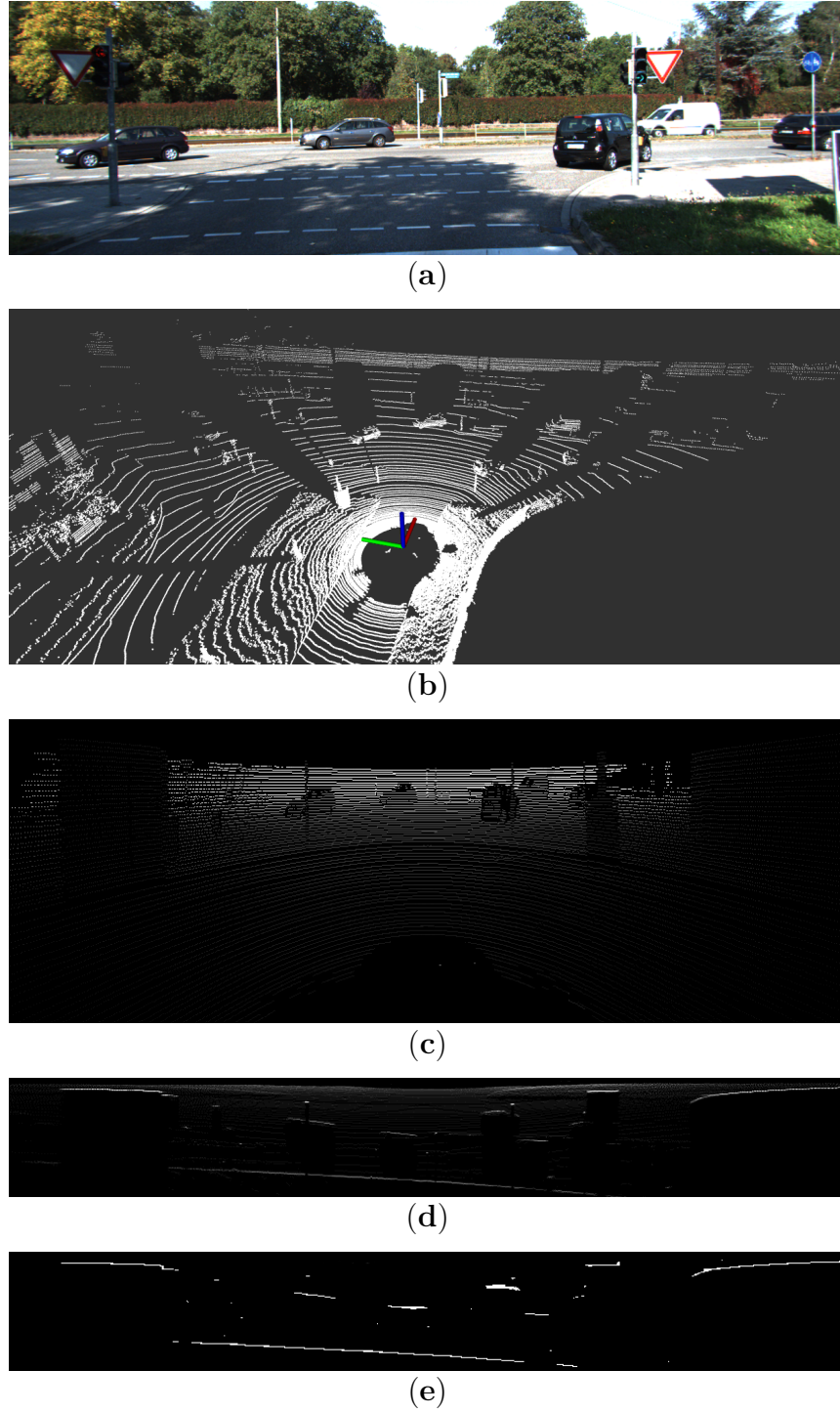


FIGURE 5.5: U-depth representation from a point cloud: (a) An RGB snapshot from the KITTI data sequence [133], (b) A snapshot of the corresponding LiDAR point cloud, (c) A virtual depth representation created using the point cloud, (d) Corresponding u-depth representation, (e) Corresponding u-depth representation after thresholding and closing operations.

is the corresponding u-depth representation after thresholding and closing operations. The obstacles are visible as horizontal white patches in Fig. 5.5(e). The proposed system does not create any virtual depth representation as shown in Fig. 5.5(c) to avoid unnecessary processing; rather, it generates the u-depth representation directly using Algorithm 3: U-Depth Representation( ), and Fig. 5.5(c) is presented only for better understanding.

Once we get prominent continuous white patches, we segment out the white patches using component analysis [97]. The segmentation process provides us with a bounding box around every white patch, and each bounding box provides the corresponding row and column ranges. Let us assume  $u_t$  and  $u_b$  are the top and bottom row indexes, respectively, and  $u_l$  and  $u_r$  are the left and right column indexes, respectively, for an obstacle. We can calculate the minimum depth ( $d_{min}$ ) and the maximum depth ( $d_{max}$ ) for the segmented obstacle from the corresponding row indexes of the u-depth representation using Equ. (5.3).

$$\begin{aligned} d_{min} &= (u_t - 1) \times \frac{Max_d}{(n - 1)} \\ d_{max} &= (u_b - 1) \times \frac{Max_d}{(n - 1)} \end{aligned} \quad (5.3)$$

The column indexes of the bounding box around the segmented patch provide the width of the corresponding obstacle on the virtual image plane, but the height of the obstacle can be erroneous in the case where the height is directly derived from the row indexes of the bounding box around the segmented patch. We have shown a detailed explanation of such erroneous estimation for EODT in Section 3.2.2.2 of Chapter 3 and proposed a restricted v-depth map representation for accurate height estimation of the corresponding obstacle. Therefore, we consider the restricted v-depth representation using the LiDAR point cloud for accurate height estimation of the obstacles.

### 5.3.1.3 Restricted V-depth Representation

A v-depth map representation is a row-wise histogram representation of depth values taken from a depth image, mainly representing the ground-plane as a curved white patch, but obstacles are not easily identifiable. A restricted v-depth representation is also a row-wise histogram representation of depth values taken from a depth image but restricted within a specific column range and a specific depth range. Fig. 5.6 presents a visual comparison between v-depth and restricted v-depth representations created from a LiDAR point cloud, where the curved patch on Fig. 5.6(b) is the ground plane,



and all vertical white patches on the curved patch are the obstacles. The restricted

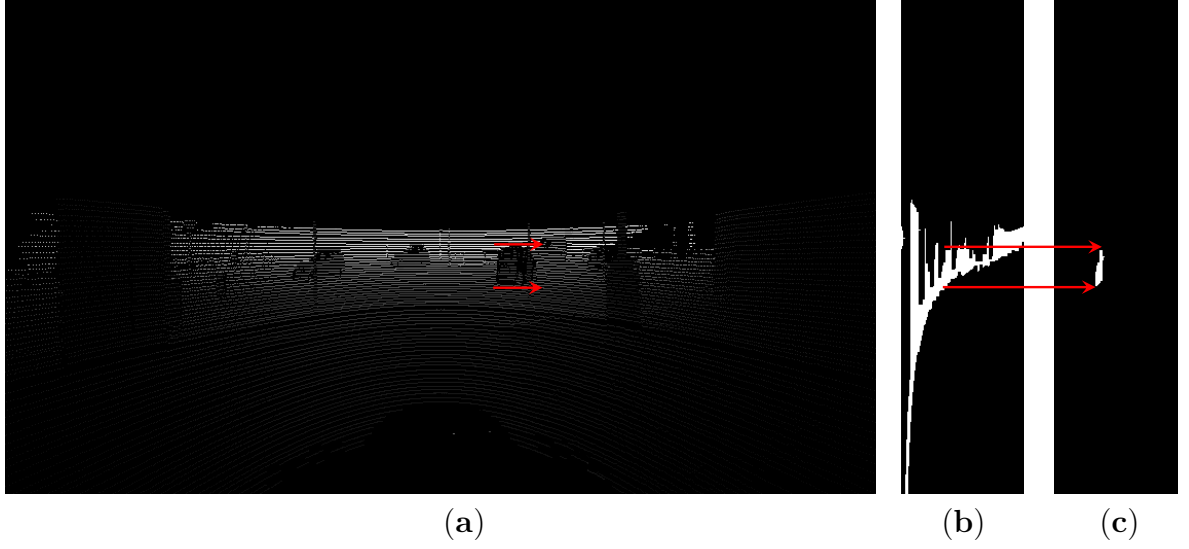


FIGURE 5.6: Comparison between v-depth and restricted v-depth representations on the KITTI data sequence [133]: (a) Formulated virtual depth image from the LiDAR point cloud. The red arrows indicate the height of an obstacle, (b) Corresponding v-depth representation, (c) Corresponding restricted v-depth representation. The red arrows from (b) to (c) indicate the specific obstacle segmented out from the v-depth representation.

v-depth representation, Fig. 5.6(c), segments out the particular obstacle where the height of the obstacle can easily be estimated from the corresponding patch height. We formulate separate restricted v-depth representations for every individual obstacle detected in Section 5.3.1.2 and estimate their height individually.

The concept of constructing restricted v-depth representation is similar to the u-depth formulation. In restricted v-depth, we only accept the points that reside between the provided obstacle depths, project the 3D points to the virtual image plane, and calculate the row index ( $D_{row}$ ) and column index ( $D_{col}$ ) of each projected 3D point. We only accept the points where the column indexes of the projected points are between the left column ( $u_l$ ) and right column ( $u_r$ ) of the obstacle. We further reject the points if the row indexes of the projected points go outside of the virtual image plane. In the restricted v-depth representation, the row index of the 3D point must be the same as  $D_{row}$  because of row-wise histogram representation, but the column index ( $D'_{col}$ ) is calculated from the depth of the 3D point. Finally, the location ( $D_{row}, D'_{col}$ ) in the restricted v-depth map is made entirely white by filling it with 255. Algorithm 4: Restricted V-Depth Representation( ) explains the details of the procedure to construct a thresholded restricted v-depth representation from the point cloud  $C_v PC$ , where step 3 checks the depth of the point within the provided obstacle depths, step 4 and 5 perform the 3D-2D projection, step 6 checks the column index is between obstacle width and

row index is between the virtual image plane, step 7 calculates the column index of the restricted v-depth representation, and finally, step 8 updates the value in the calculated pixel location. We also perform morphological operations on the thresholded v-depth representation similar to the thresholded u-depth representation; a closing operation with a  $(50 \times 3)$  structuring element to generate continuous vertical white patches, and component analysis to segment and get the dimensions of the patches. Hereafter in this chapter, we use the terms ‘u-depth representation’ and ‘v-depth representation’ to represent ‘closed thresholded u-depth representation’ and ‘closed thresholded restricted v-depth representation’, respectively.

---

**Algorithm 4** : Restricted V-Depth Representation( )
 

---

**Input:**  ${}^{C_v}PC$ ,  $u_l$ ,  $u_r$ ,  $h_{I_v}$ ,  $n$ ,  $d_{min}$ ,  $d_{max}$ ,  $Max_d$ ,  $f_h$ ,  $f_v$ ,  $c_h$ ,  $c_v$

**Output:**  $Vdepth_{(h_{I_v} \times n)}$

```

1: Initialization :  $Vdepth_{(h_{I_v} \times n)} \leftarrow 0$ 
2: for  $i = 1$  to  $|{}^{C_v}PC|$  do
3:   if  $(d_{min} \leq {}^{C_v}PC[i].Z \leq d_{max})$  then
4:      $D_{col} = \text{roundoff}(\frac{({}^{C_v}PC[i].X)f_h}{{}^{C_v}PC[i].Z} + c_h + 1)$ 
5:      $D_{row} = \text{roundoff}(\frac{-({}^{C_v}PC[i].Y)f_v}{{}^{C_v}PC[i].Z} + c_v + 1)$ 
6:     if  $(u_l \leq D_{col} \leq u_r) \ \& \ (1 \leq D_{row} \leq h_{I_v})$  then
7:        $D'_{col} = \lfloor \frac{(n-1){}^{C_v}PC[i].Z}{Max_d} + 1 \rfloor$ 
8:        $Vdepth(D_{row}, D'_{col}) = 255$ 
9:     end if
10:  end if
11: end for
12: return  $Vdepth$ 

```

---

Let us assume  $v_t$  and  $v_b$  are the top and bottom row indexes, respectively, of a patch in the v-depth representation of a given obstacle; then, the obstacle spans from columns  $u_l$  to  $u_r$  and rows  $v_t$  to  $v_b$  in the corresponding virtual depth image. Fig. 5.7 shows the relations of estimating the values of  $u_l$  to  $u_r$  from the u-depth representation and the values of  $v_t$  and  $v_b$  from the restricted v-depth representation. The presence of a very few number of obstacles within the given column range and depth range (a single prominent white patch in the example of Fig. 5.7(c)) on the restricted v-depth representation makes the height estimation easy and accurate, and this is the main advantage of a restricted v-depth map.

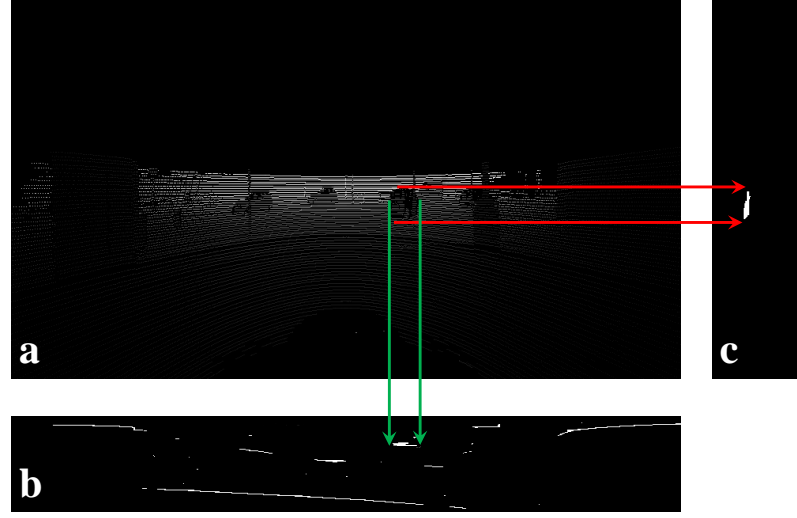


FIGURE 5.7: Estimation of dimensions of an obstacle using u-depth and restricted v-depth representation on the KITTI data sequence [133]: (a) Formulated virtual depth image from the LiDAR point cloud, (b) Corresponding u-depth representation. The green arrows indicate estimating the width of the obstacle from the u-depth representation, (c) Corresponding restricted v-depth representation. The red arrows indicate estimating the height of the obstacle from the restricted v-depth map.

#### 5.3.1.4 3D Dimension Estimation

Now we create an obstacle point cloud,  ${}^{C_v}PC_{Obj}$ , by extracting 3D points that project within the columns  $u_l$  to  $u_r$ , and rows  $v_t$  to  $v_b$  and having depth values between  $d_{min}$  and  $d_{max}$  from the point cloud  ${}^{C_v}PC$ . The point cloud  ${}^{C_v}PC_{Obj}$  represents the obstacle in the coordinate frame  $C_v$ . We first estimate an axis-aligned 3D bounding box around the point cloud  ${}^{C_v}PC_{Obj}$ . The axis-aligned bounding box is not very efficient because the volume of an axis-aligned bounding box is usually much bigger than the exact volume of the obstacle. Fig. 5.8(a) shows an example of an axis-aligned 3D bounding box around an obstacle, where the volume of the bounding box is much bigger than the obstacle. Therefore, we create an oriented 3D bounding box around the point cloud  ${}^{C_v}PC_{Obj}$ . The orientation of the 3D bounding box is estimated such that it attains the minimum volume but includes all the points from the point cloud  ${}^{C_v}PC_{Obj}$ . Fig. 5.8(b) shows the corresponding oriented 3D bounding box, where the dimensions of the bounding box firmly fit the obstacle. The dimensions of the bounding box represent the dimensions of the obstacle. Therefore, the obstacle representation in the proposed system is in the form of positions and dimensions of obstacles. Occupancy grid map is another well-established form of obstacle representation, and we present a comparison of our representation with an occupancy grid map in the section below.

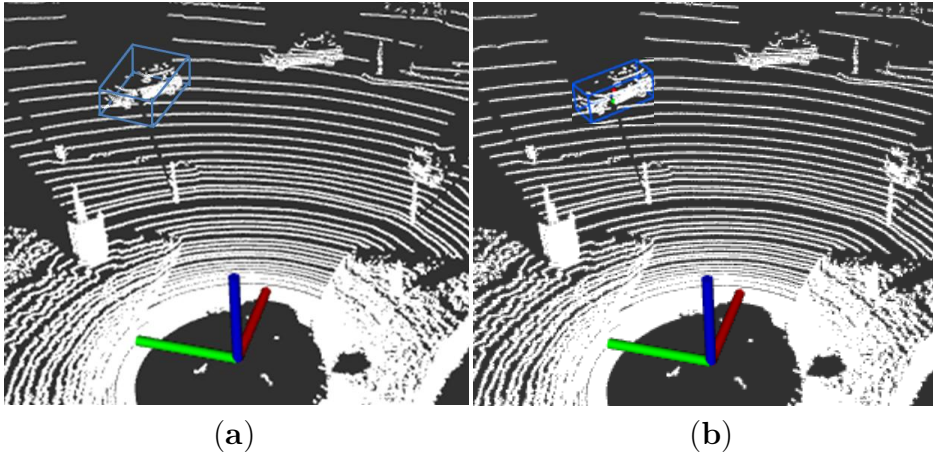


FIGURE 5.8: (a) An axis-aligned bounding box, initially estimated, is displayed in blue, (b) The corresponding oriented bounding box is shown in blue.

### 5.3.1.5 3D Occupancy Map

An occupancy grid map is a representation of the visible space in three categories: occupied, unoccupied, and unexplored. Path planning algorithms usually use this occupancy information for collision-free path estimation. In this context, Octomap [134] is a popular tool for representing the 3D occupancy grid map, where the entire 3D space is divided into small 3D cubic volumes, usually called 3D voxels. Octomap is a hierarchical octree-based approach that uses probabilistic occupancy estimation, and each node of an octree represents a voxel. A voxel is recursively subdivided into eight sub-voxels until a given minimum voxel size is reached. Therefore, the leaf nodes of an octree represent the smallest voxels. The minimum size of the voxel depends on the user-defined resolution. Each voxel can obtain any one of the three options: (i) occupied, (ii) free, and (iii) unexplored or unknown. The accuracy and processing time both increase with smaller voxels. Fig. 5.9 shows an Rviz snapshot of the 3D occupancy map of the KITTI data sequence [133] using Octomap, where the magnified view shows each obstacle is represented by multiple consecutive occupied voxels.

The proposed system represents each obstacle with a single bounding box, which means a single 3D voxel for each obstacle. Using a single voxel to represent an obstacle is a little erroneous at the obstacle boundaries, but processing time is much faster compared with a standard occupancy map (e.g., Octomap). Let us understand the situation with an obstacle (the car), as shown in Fig. 5.10(a), (b). The GT dimensions of the car are provided as follows: the height is 1.4921722 m, the width is 1.6669273 m, and the length is 4.5 m. These dimensions generate a volume of 11.193041594 cubic meters. The Octomap uses 99 occupied voxels to represent the obstacle, as shown in Fig. 5.10(c). This means it creates the corresponding hierarchical

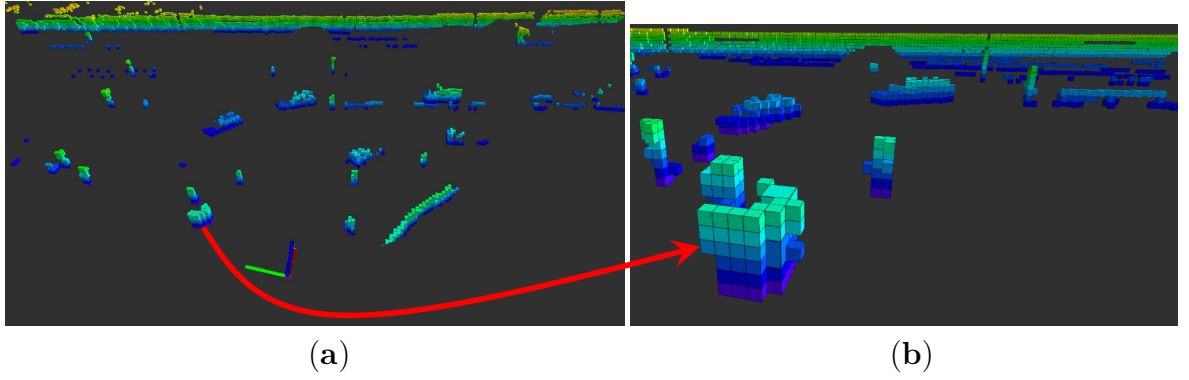


FIGURE 5.9: (a) 3D occupancy map representation using Octomap [134], (b) A magnified view shows an obstacle represented with multiple consecutive occupied 3D voxels.

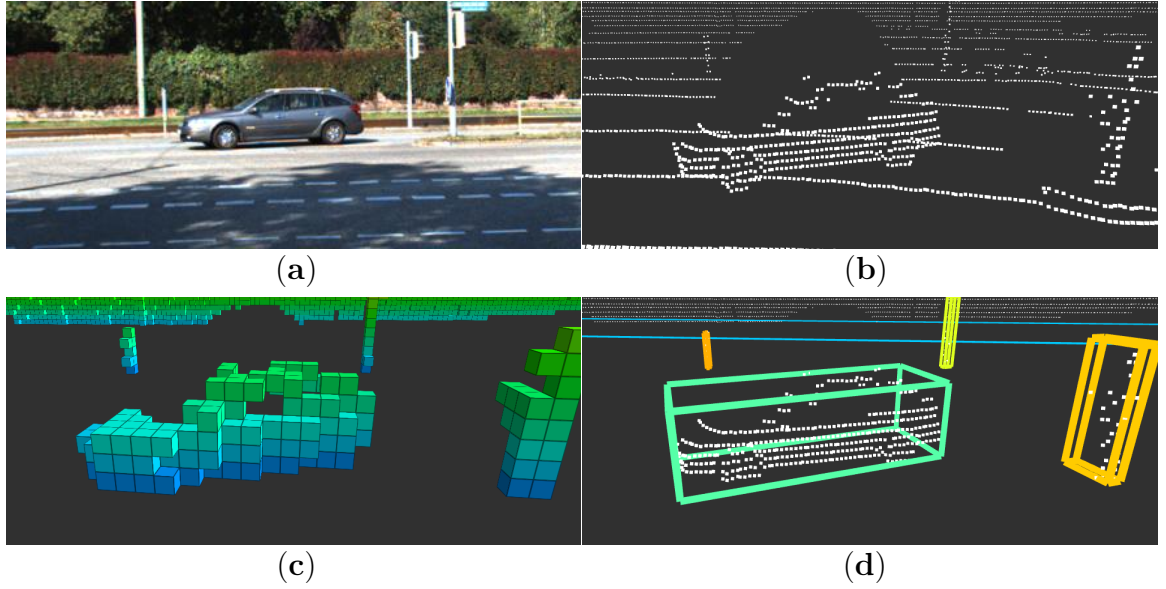


FIGURE 5.10: Comparison of occupancy representation: (a) The car is the dynamic obstacle from the KITTI data sequence [133], (b) Corresponding point cloud, (c) Occupancy representation using Octomap, (d) Occupancy representation using the proposed method.

octree data structure with all the voxels and processes the entire tree whenever it is required. However, the proposed method uses a single voxel to represent the entire car, as shown in Fig. 5.10(d), and processes it much faster. The Octomap represents the shape of the car more accurately compared to the proposed method. The estimated volume of the car using the proposed method is 11.57347 cubic meters.

Fig. 5.11 shows an execution time comparison between Octomap and the proposed method on the KITTI data sequence [133]. The timing details of Octomap represent the times for generating a new octree structure with a resolution of 0.3 m, where each point cloud is provided after removing the ground points. The average time for Octomap is about 1.258 seconds per point cloud for generating the octree structure. However, the timing details of the proposed method represent the total time for estimating the

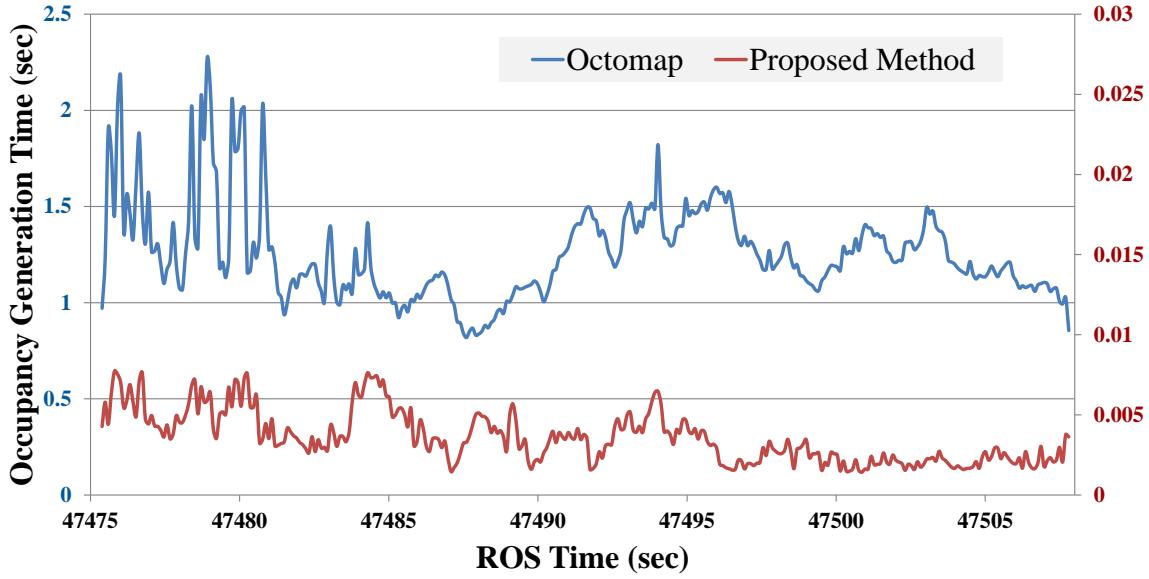


FIGURE 5.11: Time comparison for creating occupancy information from point clouds between Octomap (left vertical axis) and the proposed method (right vertical axis).

dimensions and oriented 3D bounding boxes for all obstacles in each point cloud, where each point cloud contains about 14 obstacles on average. The proposed method takes about 3.577 milliseconds on average to estimate the occupied regions from each point cloud. The comparison shows the advantage of using a single voxel for each obstacle. The proposed system can easily be modified to represent a single obstacle with multiple 3D voxels for a more accurate representation at the cost of increased processing time. The standard occupancy map uses a fixed resolution, which produces fixed-size voxels, but the proposed system creates bounding boxes that fit the corresponding obstacles firmly. Therefore, the sizes of the bounding boxes are not fixed, and the sizes are dynamically estimated from the point cloud. Once the obstacles are represented, they are tracked on subsequent point clouds. Therefore, we concentrate on the tracking in the section below.

### 5.3.2 Obstacle Tracking

Tracking is the task that associates the detected obstacles in the subsequent captured point clouds. The associations help to estimate the states (static/dynamic) of the detected obstacles and further predict the future location of dynamic obstacles within a predicted zone. We found the proposed tracking mechanism in EODT (Section 3.2.3) is adequate and performs equally well with our proposed obstacle detection method from LiDAR point clouds. Therefore, we follow a similar approach for tracking multiple obstacles in subsequent captured point clouds as presented in EODT. In the present



work, we create signatures for every obstacle from the u-depth representation. Let us name the obstacle point cloud  ${}^{C_v}PC_{Obj}$  as obstacle A. The signature of the obstacle A is considered as  $\{\Phi_A, \Phi_{A_l}, \Phi_{A_r}, Vis_A\}$ , where  $\Phi_A$  is a set of ordered vectors formed using contour pixels of A in u-depth representation,  $\Phi_{A_l}$  is the left most pixel of A in u-depth representation,  $\Phi_{A_r}$  is the right most pixel of A in u-depth representation, and  $Vis_A$  represents partial or full visibility of A. The system calculates probable zones around every obstacle and assumes the obstacle motion is always within the probable zones. Therefore, the system looks for matching signatures within the estimated probable zones in the subsequent u-depth representations. The matching is based on similarity checking between signatures, where a dissimilarity cost is calculated based on obstacle dimensions, positions, vector length, and vector direction. We consider a match once the dissimilarity cost is below a threshold. Once the match is found for any obstacle, the signature of the obstacle is updated with the latest signature. We refer to Section 3.2.3 for a detailed description of the tracking procedure that the current proposed method follows. Once an obstacle is successfully tracked in five consecutive u-depth representations, we transform the positional estimations of the obstacle to the world coordinate frame  $W$ . We first use the transformation  ${}^{C_v}T_L$  to transform the point cloud  ${}^{C_v}PC_{Obj}$  to the coordinate frame  $L$  using Equ. (5.4) and then finally to the world coordinate frame  $W$  using Equ. (5.5).

$${}^LPC_{Obj} = ({}^{C_v}T_L)^{-1} ({}^{C_v}PC_{Obj}) \quad (5.4)$$

$${}^WPC_{Obj} = {}^W T_B ({}^B T_L ({}^LPC_{Obj})) \quad (5.5)$$

Fig. 5.12 is the Rviz [99] snapshot that shows detected obstacles with their bounding boxes for the dataset presented in Fig. 5.5. We identify dynamic obstacles using the estimated velocity profiles of the centroids in the world coordinate frame  $W$ .

### 5.3.3 Dynamic Obstacle Estimation

We have described the process of detecting and estimating obstacles from LiDAR point clouds and thereafter tracking those obstacles in multiple point clouds. With the tracking process, we can identify the location of any obstacle in the temporal scale. Now we look forward to identify the dynamic obstacles with the temporal information of any obstacle.

We first transform the locations of all obstacles into the fixed world coordinate frame  $W$ . Afterward, we create a profile using the temporal positional information of each obstacle and calculate the positional changes from the profile. Once we find the



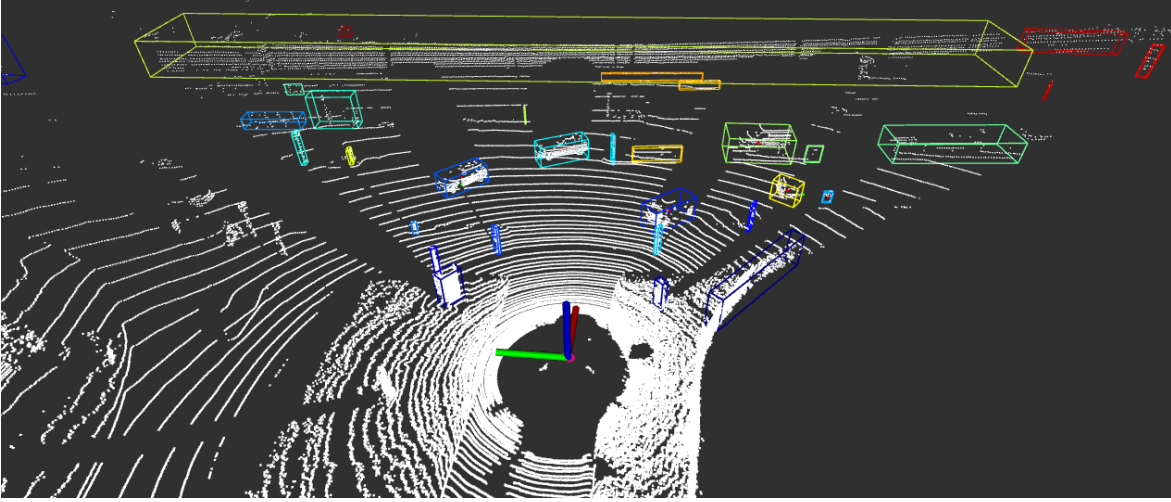


FIGURE 5.12: Obstacle detection using the proposed method on the point cloud from Fig. 5.5. An Rviz [99] snapshot displays 3D bounding boxes of detected obstacles in the coordinate frame  $W$ , with the pose of the coordinate frame  $L$ .

positional changes above a given threshold of any obstacle, we consider the obstacle as dynamic. We estimate the velocities of all the dynamic obstacles from their temporal positional information.

## 5.4 Experimental Results

We use a laptop with an Intel Core i5-10310U, eight cores @ 1.7 GHz, and 8 GB of RAM for experimental purposes. We implement the proposed method in C++ with the support of the Robot Operating System [111]. We evaluate the current system with multiple open and self-captured datasets to establish the capability and applicability of the system. Table 5.1 shows a detailed configuration of all the datasets; first, we use simulated data to highlight the effectiveness of LiDAR over an RGB-D sensor and compare the results with SoA methods. We also use the popular open dataset on ground vehicles, KITTI [133], and compare the results of the proposed system with SoA methods. Additionally, we evaluate the proposed system in real-time data capture and execution. Furthermore, we demonstrate the capabilities of the presented system with an open aerial data sequence [136], [137].

Table 5.2 shows the values of the parameters that are set for u-depth and v-depth formulations in all of our experiments, where these values are chosen empirically. We use the same values as in EODT for all of our tracking parameters. We consider EODT, FO3D2D [3], and 3DPCO [1] to compare with the present system in multiple evaluations, where we implement FO3D2D and 3DPCO by following the details in the manuscript to the best of our understanding.

TABLE 5.1: Configurations of all experimental datasets

Datasets	Type	Mounted on	RGB-D Sensor	LiDAR (No. of Channels, Range, Horizontal FOV, Vertical FOV)	Obstacle Description (Dynamic)
Gazebo Simulation [95]	Indoor	Husky Robot [102]	RealSense D400 [43]	Velodyne (64, 100 m, 360°, ±15°)	Single
KITTI [133]	Outdoor	A Car	–	Velodyne HDL-64E (64, 120 m, 360°, 26.8°)	Multiple
Self-Captured (Real H/W Platform)	Indoor	TurtleBot3 Waffle Pi [135]	–	CE30-D (16, 0.4-30 m, 60°, 4°)	Single
Open Data [136], [137]	Outdoor	DJI Matrice 100 UAV	–	Velodyne VLP 16 Puck Lite (16, 100 m, 360°, ±15°)	No Dynamic Obstacle

TABLE 5.2: Parameter Values in the Proposed System

$h_{I_v}$	$w_{I_v}$	$n$	$Max_d$	$f_h$	$f_v$	$c_h$	$c_v$	$FOV_h$	$FOV_v$
480	848	120	100	200.1	411.1	424.0	240.0	129.47°	60.55°

#### 5.4.1 Experiment 1: Evaluation on Simulated Dataset

In this experiment, we present an evaluation to demonstrate the accuracy improvement with the usage of LiDAR over an RGB-D camera. Additionally, we compare the accuracy of the proposed method with a SoA method that uses LiDAR point clouds for obstacle estimation. We create an indoor environment in the Gazebo simulation environment [95], where two Husky robots [102] are placed. A snapshot of the Gazebo simulation environment is shown in Fig. 5.13(a). Each Husky robot carries a Velodyne LiDAR and a RealSense D400 [43] RGB-D camera, as depicted in Fig. 5.13(b). The LiDAR has 64 channels, a range of 100 meters, with a vertical FOV of ± 15°, and a capture rate of 10 Hz. The RealSense has a range of 0.105–6 meters with a capture rate of 30 Hz, but the noise in depth estimation increases rapidly beyond 3 meters. Both Huskies follow a predefined way point-based path. The GT positions (origin of the base\_link coordinate frame) of both Huskies are recorded in the world coordinate frame  $W$ . Subsequently, the GT relative positions (origin of the base\_link coordinate frame) of the second Husky from the base\_link coordinate frame of the first Husky are evaluated.

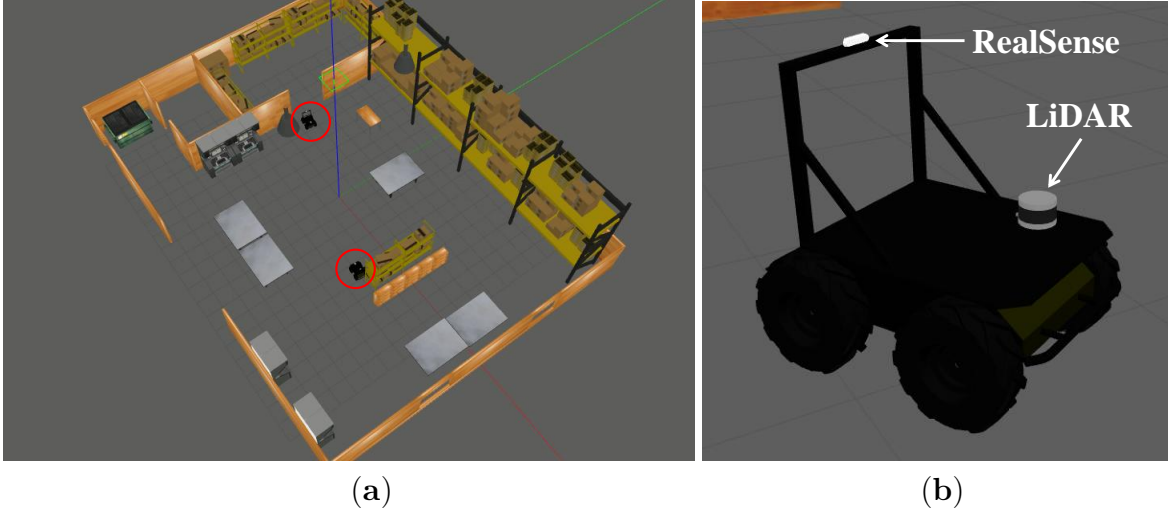
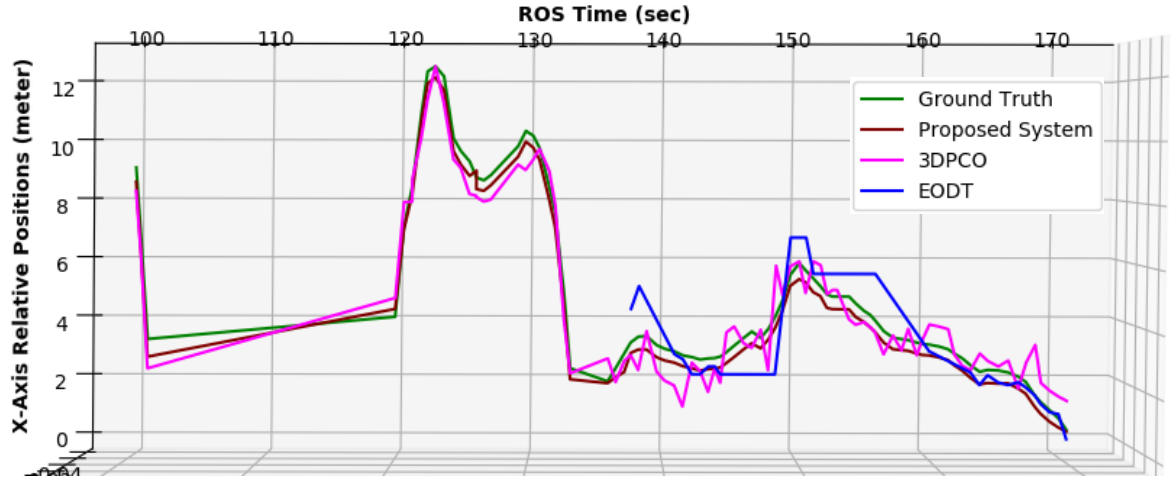


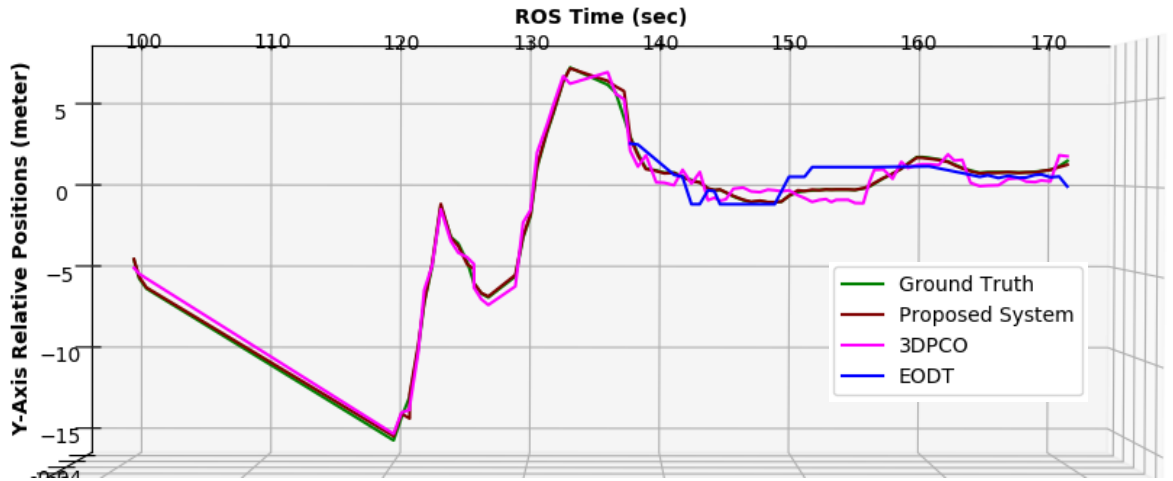
FIGURE 5.13: (a) The simulated environment in Gazebo [95] with two Husky robots [102], circled in red, (b) A Husky robot equipped with a Velodyne LiDAR and a RealSense D400 [43] RGB-D camera.

We estimate the relative position of the second Husky from the `base_link` coordinate frame of the first Husky using our proposed system, as well as 3DPCO and EODT, where EODT takes input from an RGB-D camera, but 3DPCO takes input from a 3D LiDAR. Here, the estimated positions are the centroids of the bounding parallelepiped. Fig. 5.14 shows the estimation plots with the GT in the X, Y, and Z axes. The simulated RGB-D sensor has a depth range of up to 6 meters, whereas LiDAR has a depth range of 100 meters. Therefore, the LiDAR point clouds capture any obstacles from a long range. The estimation curves of 3DPCO and the proposed system in Fig. 5.14 are present from the beginning, indicating that the second Husky is detected from the beginning. But the estimation curve of EODT starts at ROS time 137.118 seconds, indicating that EODT is unable to detect the second Husky between ROS time 101.181–137.118 seconds, as the second Husky was far apart and able to detect that as an obstacle only when the second Husky is close enough (near to 4 meters). Therefore, the long range of LiDAR allows for early obstacle detection and aids in safe navigation.

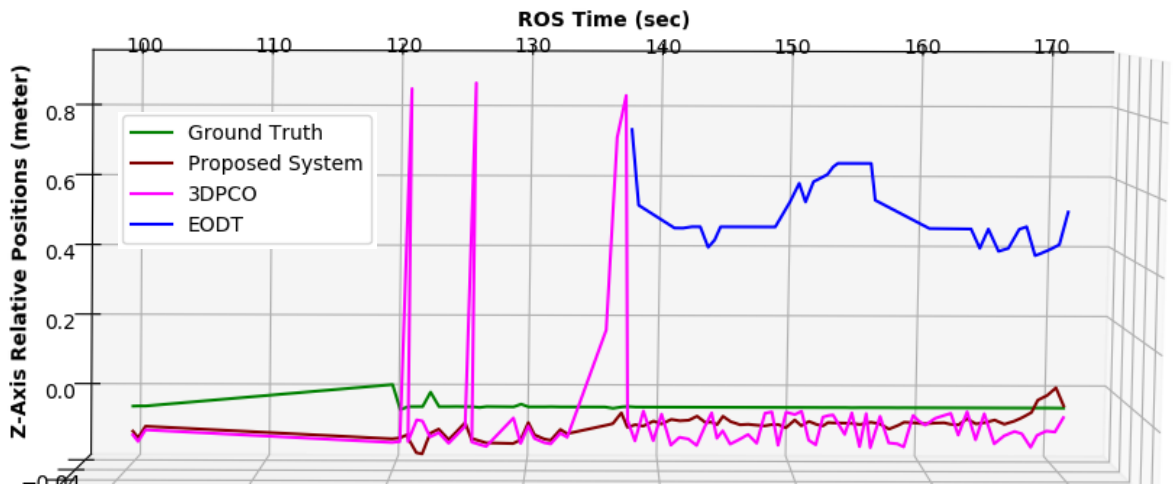
The estimation error of EODT is the noisiest among the compared methods in Fig. 5.14, and this is due to the noisy depth estimation of the RGB-D sensor. The estimation of 3DPCO is improved compared to EODT due to accurate depth estimation with the LiDAR. We notice that the estimation errors of 3DPCO increase when the obstacle is closer, and this is due to noisy ground plane segmentation. The estimation errors of the present system are lower compared to 3DPCO and EODT due to accurate dimension estimation of obstacles. The estimation error in the Z-axis (opposite to the gravity vector) is the highest because the visually estimated centroid always differs



(a)



(b)



(c)

FIGURE 5.14: Comparison plots of the relative positions of the second Husky from the first Husky among the present system, 3DPCO, EODT, and the ground truth. The curve closer to the ground truth indicates better estimation. (a) X-axis, (b) Y-axis, (c) Z-axis.

from the origin of the base\_link coordinate frame. The comparison graphs conclude that the accuracy of the proposed system, which uses LiDAR point clouds, is improved compared to 3DPCO.

### 5.4.2 Experiment 2: Evaluation on the KITTI datasets

We extensively evaluate the proposed system using the ‘Object Tracking Evaluation’ data sequences under the KITTI Vision Benchmark Suite [133]. The dataset consists of point clouds captured with a Velodyne HDL-64E LiDAR, and synchronized images are captured with stereo cameras. The LiDAR has 64 channels with equally spaced angular subdivisions, where the vertical FOV is  $26.8^\circ$ , and the capture rate is 10 Hz. The maximum range of the LiDAR is about 120 meters. The datasets record multiple static and dynamic objects such as cars, pedestrians, cyclists, trees, walls, poles, etc., on the roads and highways of urban and rural areas. The 3D GT positions and dimensions of moving objects that are visible on the stereo image plane are provided for every captured frame.

#### 5.4.2.1 Quantitative Evaluation

We use four data sequences from each of the ‘City’, ‘Residential’, and ‘Road’ categories to evaluate the dynamic obstacle detection and tracking performance of the proposed system. Table 5.3 presents the details of the dynamic obstacle evaluation as well

TABLE 5.3: Comparison on the Dynamic Obstacle Detection on KITTI Object Tracking Dataset [133]

Sequences (2011_09_26_drive_)	No. of Frames	No. of Dynamic Obstacles	FO3D2D [3]			3DPCO [1]			Our Method		
			TP	FP	FN	TP	FP	FN	TP	FP	FN
City	0001	114	15	1	0	15	0	0	15	0	0
	0009	453	97	4	2	96	3	3	98	2	1
	0014	320	41	1	0	41	1	0	41	0	0
	0059	379	59	3	1	59	2	1	60	1	0
Residential	0022	806	61	3	2	61	2	2	63	1	0
	0023	480	162	11	5	161	9	6	167	2	0
	0035	137	36	4	1	35	4	2	37	0	0
	0036	809	88	5	1	86	4	3	89	1	0
Road	0015	303	36	3	0	36	2	0	36	0	0
	0028	435	10	0	0	10	0	0	10	0	0
	0032	396	28	1	0	28	1	0	28	0	0
	0070	426	8	0	0	8	0	0	8	0	0

**TP:** True Positive, **FP:** False Positive, **FN:** False Negative.

as the comparison with FO3D2D [3] and 3DPCO [1]. ‘True Positive’ indicates the

number of dynamic obstacles detected correctly, ‘False Positive’ indicates the number of wrongly detected dynamic obstacles, and ‘False Negative’ indicates the number of missed dynamic obstacles. Therefore, the summation of ‘True Positive’ and ‘False Negative’ equals the actual number of dynamic obstacles, which must be equal to the GT. A bigger value of ‘True Positive’ and smaller values of both ‘False Negative’ and ‘False Positive’ represent more accurate estimations. The proposed method shows improvements in dynamic obstacle detection, with correct detection in most cases. However, ‘False Positive’ cases occur due to errors in the self-localization module. The proposed method also detects static obstacles with equal accuracy and shows a significant improvement in the detection of very thin obstacles like poles on the road compared to existing methods like FO3D2D and 3DPCO. Fig. 5.15 shows one such example, where the red arrows connect the 2D image to the corresponding detected poles as obstacles in the 3D point clouds.

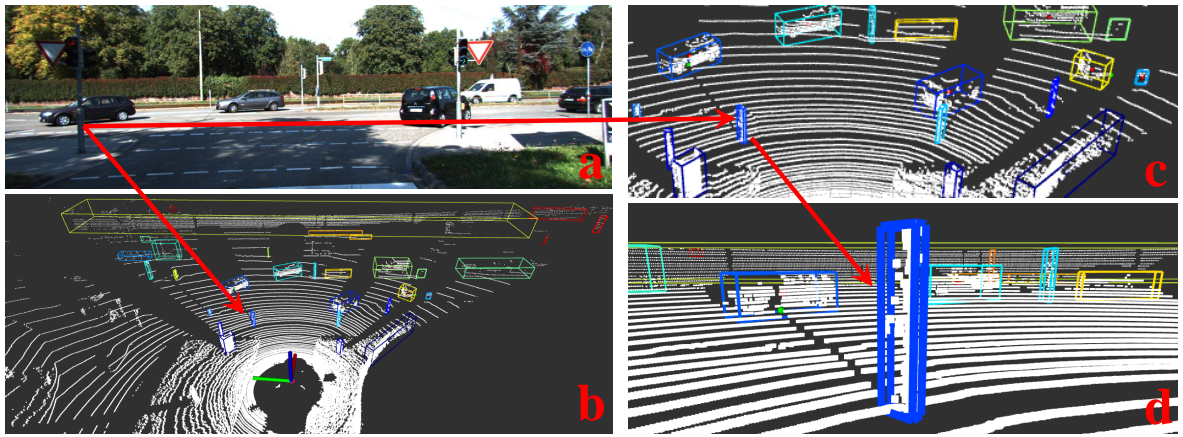


FIGURE 5.15: Thin obstacle detection using the proposed method: (a) An RGB snapshot from the KITTI dataset [133], (b) The corresponding Rviz [99] snapshot displays the 3D bounding boxes of the detected obstacles, (c) A magnified view containing multiple poles on the road as detected obstacles, (d) Another magnified view showing a pole as an obstacle.

The arrows indicate the corresponding pole in other images.

We also evaluated the estimation accuracy of dynamic obstacles, focusing on centroid estimation in the Velodyne coordinate frame  $L$  with GT. Fig. 5.16 shows the estimated and GT positions of the centroid of a dynamic obstacle (a car) with ID 2 from the ‘2011\_09\_26\_drive\_0014’ data sequence [133]. The positions of the centroids are estimated accurately in the XY plane, visible from the top view (Fig. 5.16(a)). However, there is a significant error in the Z direction (height estimation), visible from the side view (Fig. 5.16(b)). The ground-truth heights of all dynamic obstacles are considered slightly above the ground plane, whereas the estimated heights are half of



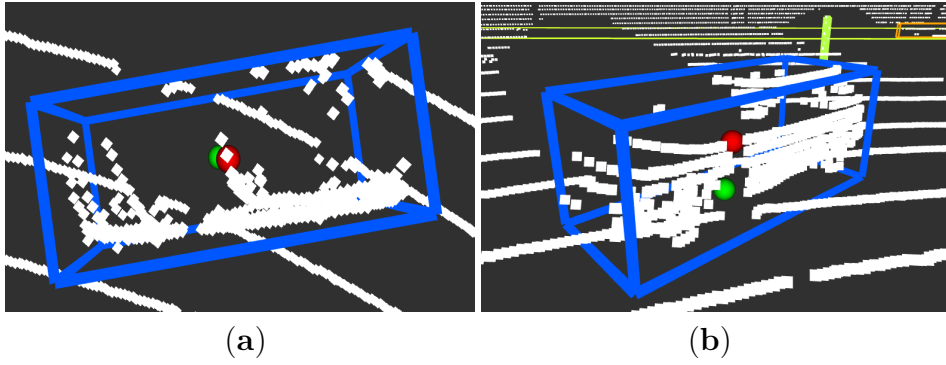


FIGURE 5.16: A visual comparison between the estimated centroid (in red) and the ground truth centroid (in green): (a) Top view, (b) Side view.

the heights of the corresponding estimated 3D bounding boxes. This conceptual difference is the main reason for the estimation error in the Z direction. Therefore, we concentrate only on the X and Y axes. Fig. 5.17 and Fig. 5.18 depict comparisons of the X and Y components, respectively, of the estimated centroids by FO3D2D [3], 3DPCO [1], and the proposed method against the GT. The estimation curves in both

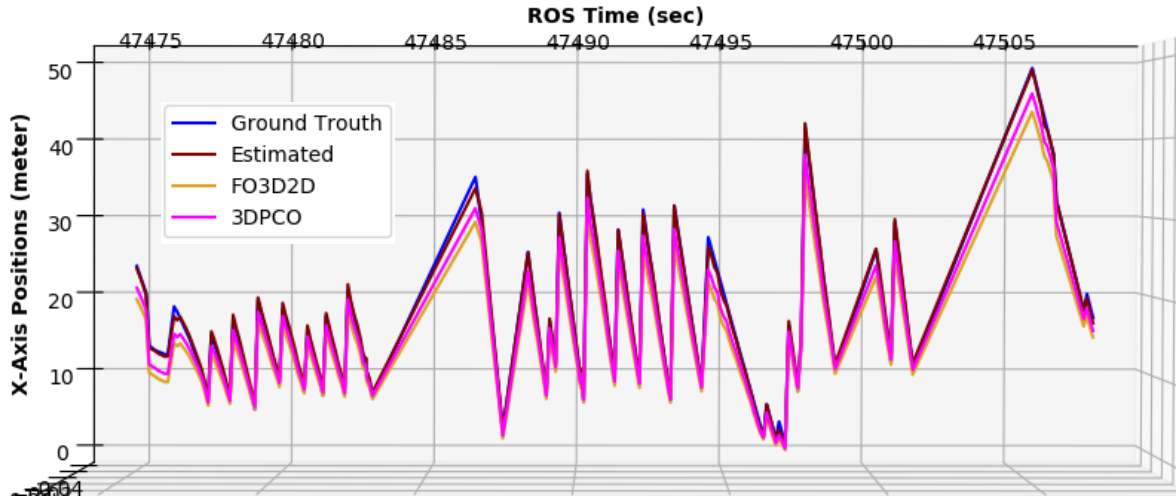


FIGURE 5.17: A X-axis comparison plot with the ground truth. The curve closer to the ground truth indicates better estimation.

figures indicate that, in most cases, the estimation accuracy of 3DPCO exceeds that of FO3D2D in both X and Y components. The proposed method exhibits the highest accuracy compared to FO3D2D and 3DPCO. Therefore, we consider only 3DPCO in our next comparison. The maximum errors in the X and Y components of the proposed method are 0.4 and 0.5 meters, respectively. Fig. 5.19 shows the error plot of the Euclidean distance estimation of the centroids by 3DPCO and the proposed method. The estimation error by 3DPCO is at most 2.4 meters, whereas the estimation error by the proposed methods is at most 0.9 meters. The error in the Z-axis contributes to



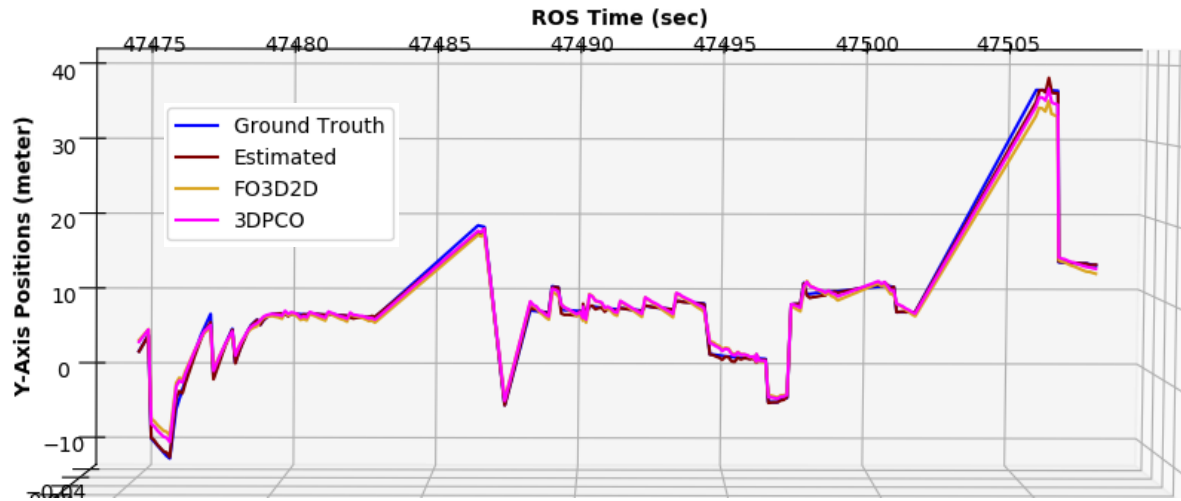


FIGURE 5.18: A Y-axis comparison plot with the ground truth. The curve closer to the ground truth indicates better estimation.

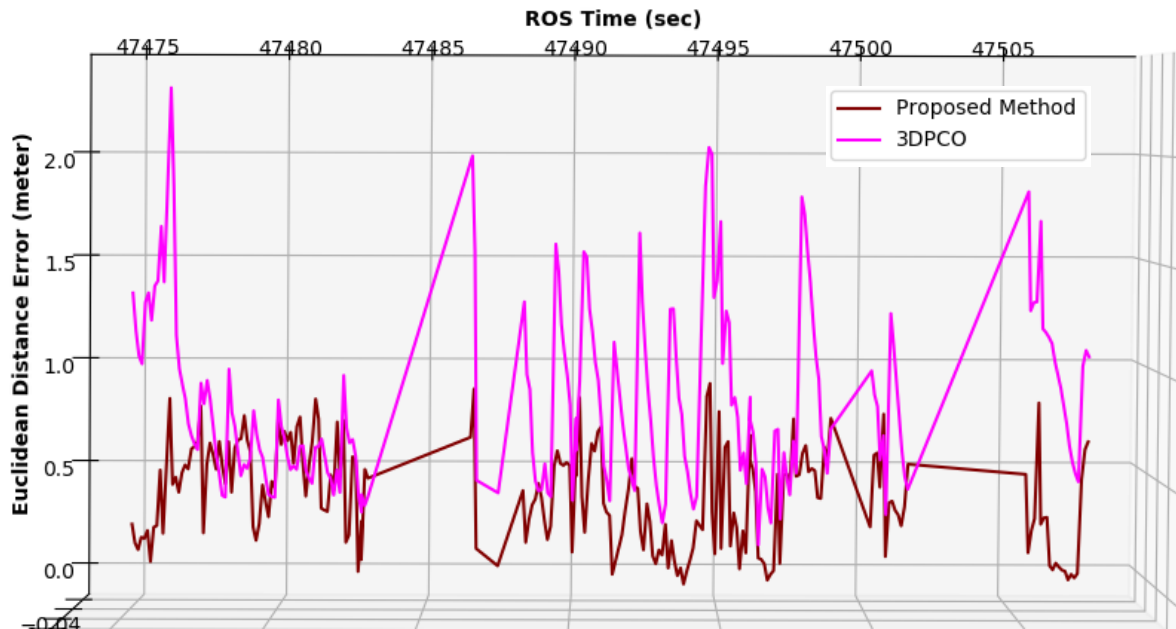


FIGURE 5.19: Error plot of Euclidean distance estimation.

the major portion of the Euclidean distance estimation. Therefore, we can conclude that the proposed method performs better than FO3D2D and 3DPCO.

### 5.4.3 Experiment 3: Evaluation with Real-Time Robot Motion

We evaluate the proposed method with real-time robotic motion, employing two TurtleBot3 Waffle Pi [135] for our experiments, with each TurtleBot3 carrying an OpenMANIPULATOR-X [138]. We attach a CE30-D Solid State Time of Flight (ToF) Infrared 3D LiDAR on one of the TurtleBot3, and the arrangement is shown in Fig. 5.20. The

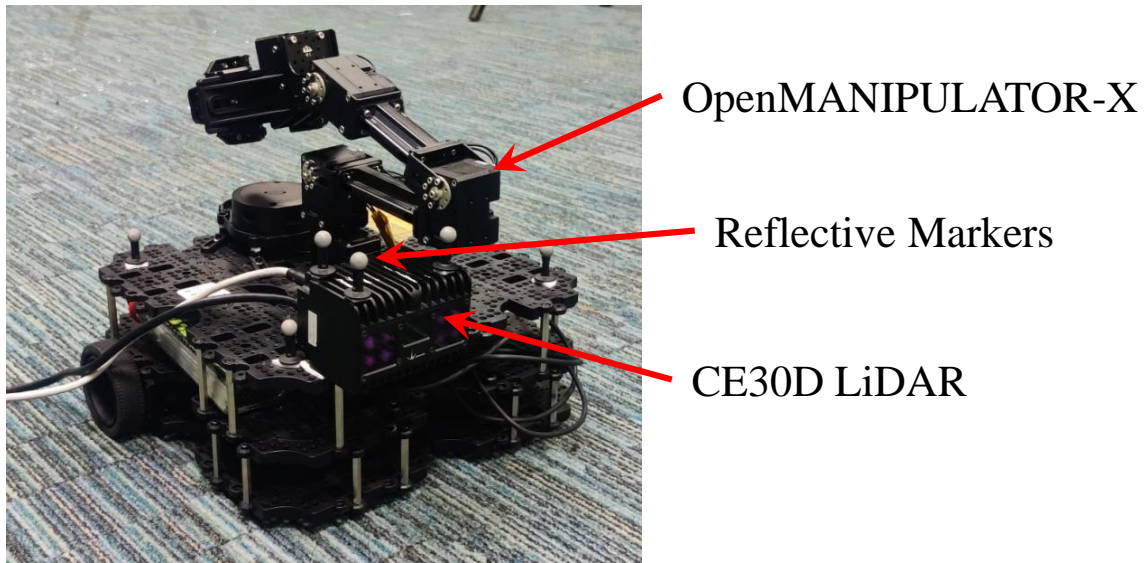


FIGURE 5.20: Experimental Setup – A TurtleBot3 Waffle Pi [135] equipped with a CE30-D Solid State 3D LiDAR.

CE30-D LiDAR has a horizontal and vertical FOV of  $60^\circ$  and  $4^\circ$ , respectively, with an operating depth range of around 0.4–30 meters. The small FOV allows us to capture the point cloud in very narrow regions, but the system configuration requires no changes to adopt the small FOV. We move both robots with velocity command-based navigation, connecting them to a laptop over a wireless network. Fig. 5.21 shows the time-stamped images of robot motions, where the second robot passing from the left to the right as a dynamic obstacle. The first robot carries the CE30-D LiDAR and faces towards the second robot while both are in motion. The CE30-D LiDAR is connected to a laptop (the same laptop as mentioned in Section 5.4) over wired Ethernet and performs the entire computation of the proposed method because the robot does not carry any other computation board with enough processing power and memory. We use the OptiTrack motion capture system [139] with 8 cameras to capture the GT, where optical reflective markers are attached on the CE30-D LiDAR as well as the four



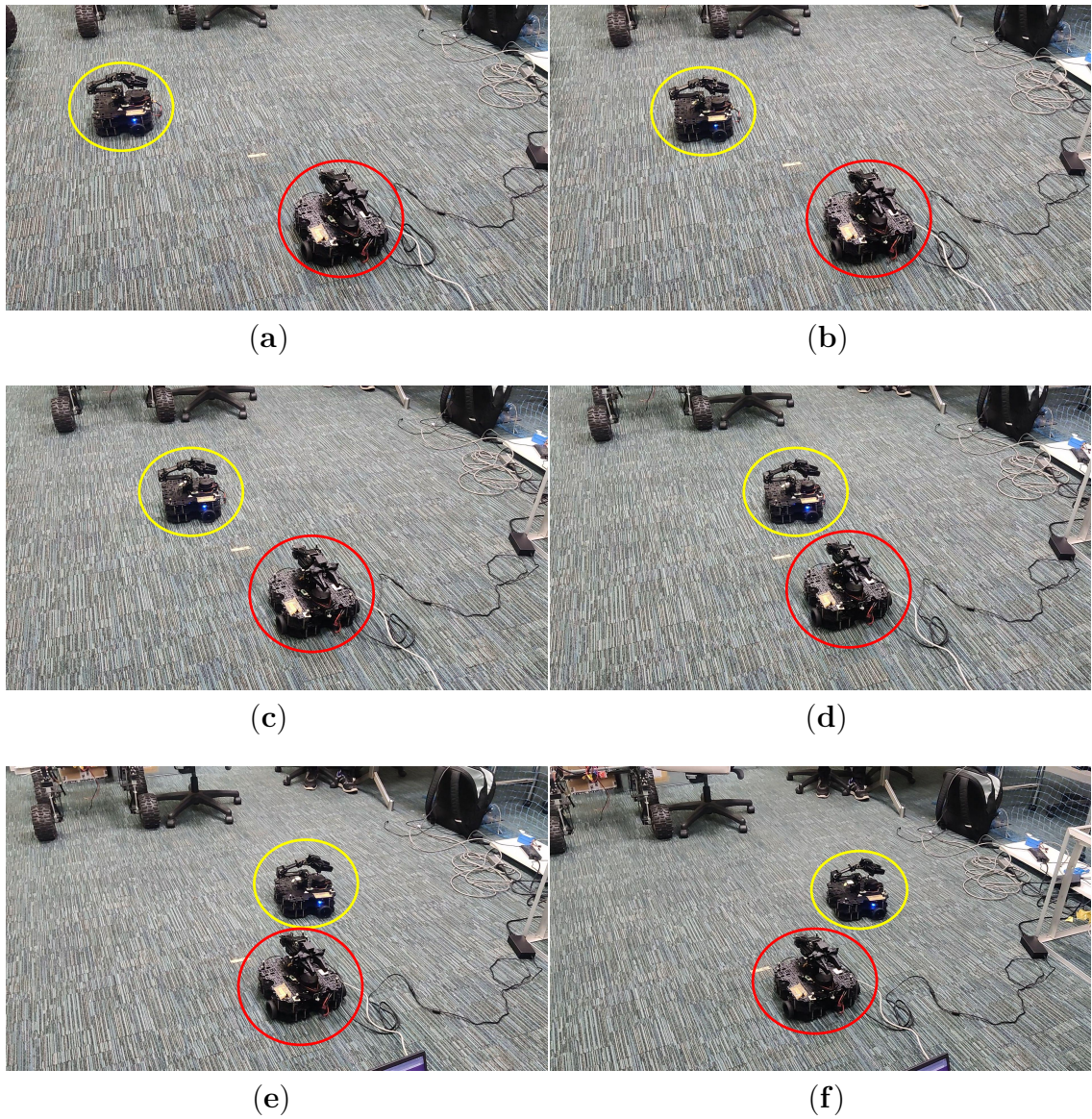


FIGURE 5.21: Time-sampled snapshots of the experimental environments, where two robots are in motion. The first and the second robots are encircled in red and yellow, respectively.

corners of both robots. The OptiTrack motion capture system directly provides the GT poses of the CE30-D LiDAR and the poses of the second robot.

Fig. 5.22 shows a snapshot of the performance of the proposed system on the point cloud captured by the CE30-D LiDAR. The dynamic obstacle (the second robot) is

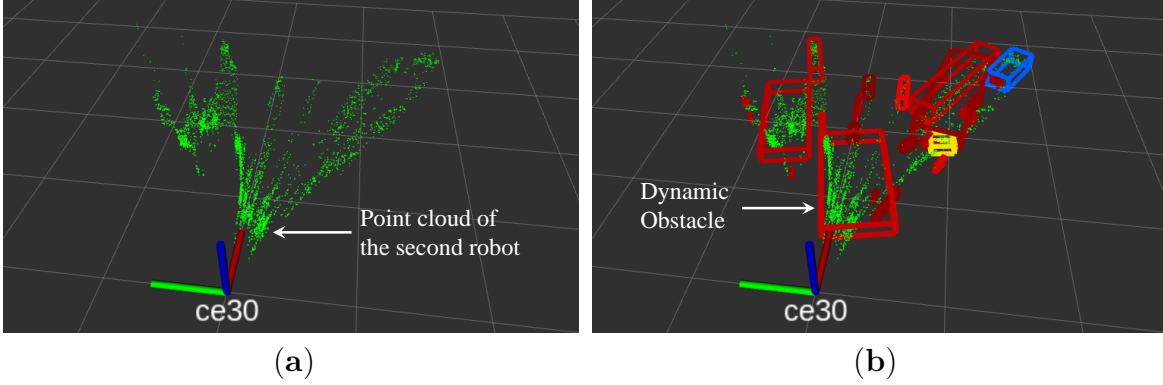
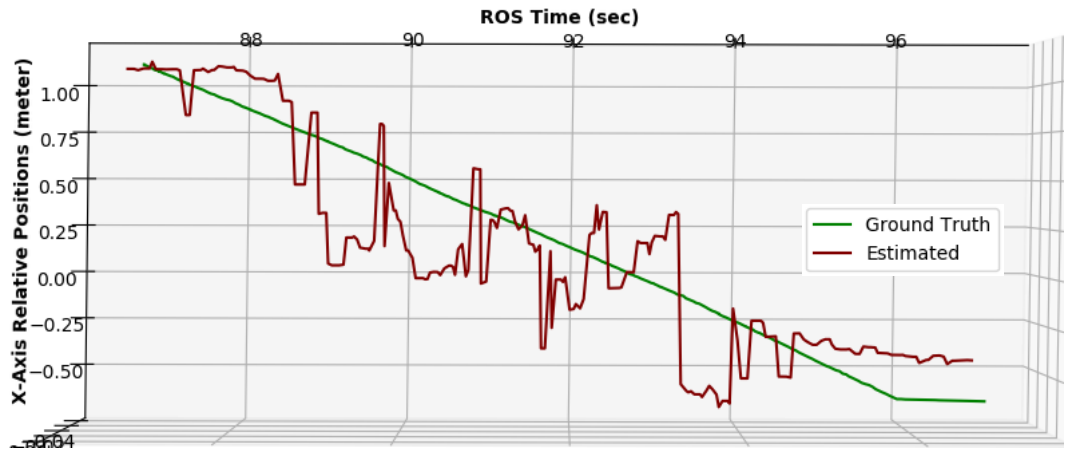


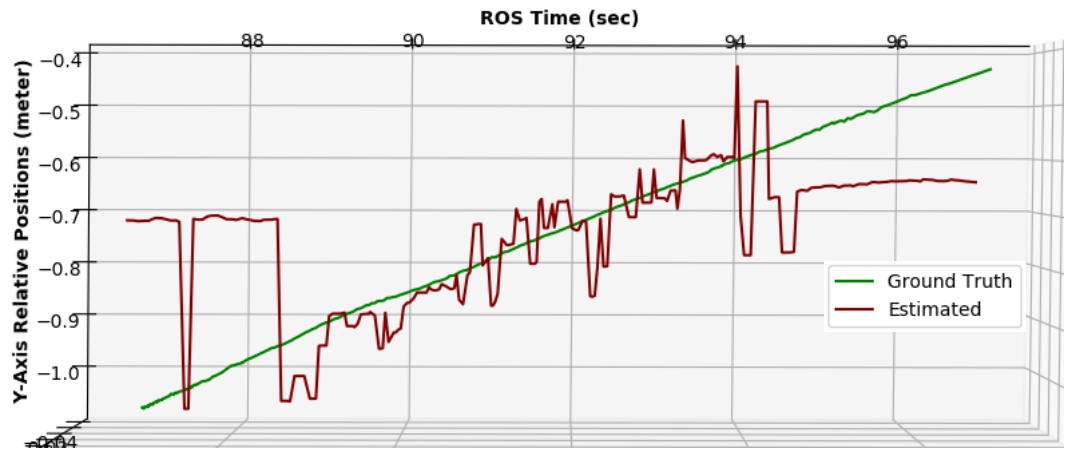
FIGURE 5.22: (a) Captured point cloud is shown in Rviz [99], (b) Detected obstacles on the point cloud by the proposed method.

the closest obstacle to the LiDAR coordinate (shown as “ce30” in Fig. 5.22). It is interesting to notice that the estimated dimensions of the dynamic obstacle are bigger than the actual ones, as depicted in Fig. 5.22(b). The reason for such noisy estimation is the noisy point cloud captured by the CE30-D LiDAR, which is shown in Fig. 5.22(a).

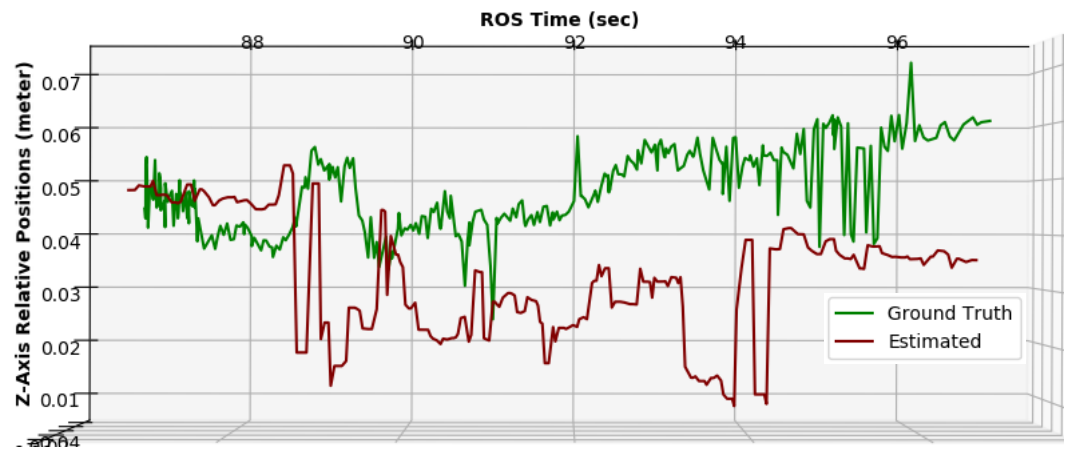
We have evaluated the estimation accuracy of the dynamic obstacles (the second robot in Fig. 5.21) in terms of centroid estimation in the Velodyne coordinate frame  $L$  with the GT. Fig. 5.23 presents our estimation accuracy, where we estimate the accuracy of the relative positions of the dynamic obstacle in all three axes. Fig. 5.24 provides an accuracy comparison in relative distance measurements. The point cloud captured with the CE30-D LiDAR is much noisier compared to the point cloud captured with the Velodyne LiDAR used in the KITTI dataset [133], as evident from the noisy point trails at the edge boundary of every obstacle and this can be viewed in Fig. 5.22(a). These noisy points impact the estimations, especially when the obstacle is partially visible at the edge boundaries (before the 89<sup>th</sup> second and after the 95<sup>th</sup> second). This aspect is visualized in the estimation curve of Fig. 5.23(b). The estimation error is at most 0.7 meters, considering all axes.



(a)



(b)



(c)

FIGURE 5.23: Comparison plots of the estimated relative positions of the dynamic obstacle (the second robot) from the first robot by the present system. The curve closer to the ground truth indicates better estimation. (a) X-axis, (b) Y-axis, (c) Z-axis.



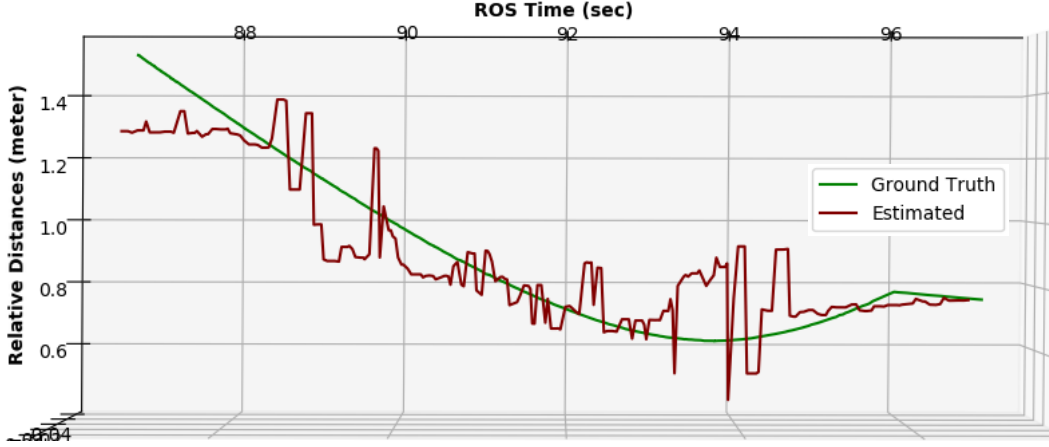


FIGURE 5.24: Relative distance comparison plots for the dynamic obstacle (second robot) from the first robot. Closeness to the ground truth curve indicates better estimation.

#### 5.4.4 Experiment 4: Evaluation on Aerial LiDAR Open Dataset

The proposed system does not assume any specific type of robot motion and is equally applicable for UAVs. Therefore, we evaluate the capability and performance of the proposed system with open data [136], [137], which was captured using a Velodyne VLP 16 Puck Lite LiDAR mounted on a DJI Matrice 100 UAV for checking inventory in a warehouse environment. The VLP 16 Puck Lite is a compact, lightweight sensor and is ideal for drones. It has a  $\pm 15^\circ$  vertical FOV with 16 equally spaced channels and a range of 100 meters. The operating environment is a warehouse, where industrial materials are piled up, containing only static obstacles. The dataset provides GT for self-localization but does not include GT for obstacle location. Therefore, we are unable to make any comparison with the GT for obstacle estimation. Fig. 5.25 provides a visual representation of the estimated obstacles, where all frontal obstacles are detected. Multiple views in Fig. 5.25 demonstrate an accurate obstacle estimation. This experiment establishes the capability of the proposed system in obstacle detection with less dense point clouds and UAV-type motion.

#### 5.4.5 Execution Time

We compare the execution times of different modules in our proposed system with EODT and FO3D2D [3], where all three systems are implemented as multi-threaded applications among eight cores. We chose EODT and FO3D2D from the SoA for timing comparison because EODT is the fastest method for obstacle estimation using an RGB-D camera, and we found FO3D2D to be the fastest method for obstacle estimation using LiDAR point clouds, among other SoA methods. Table 5.4 shows the

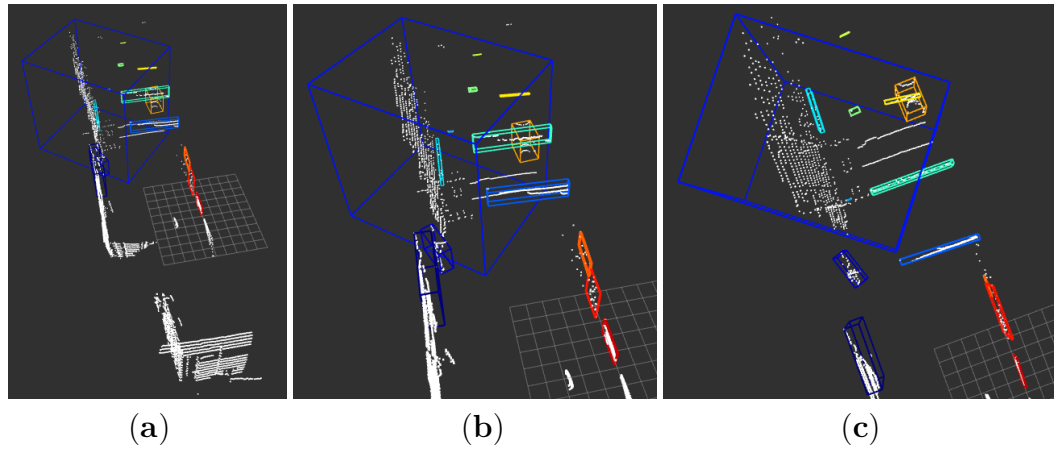


FIGURE 5.25: An Rviz [99] view of detected obstacles at a specific time instance with the dataset in [136]: (a)–(b) Side views, (c) Top view. Multiple views establish accurate obstacle estimation with UAV motion.

average running time of different modules on the ‘2011\_09\_26\_drive\_0014’ of KITTI Object Tracking data sequence [133], where ‘–’ means not applicable.

TABLE 5.4: Comparison of different methods in terms of execution Time (milliseconds).

Modules	EODT	FO3D2D [3]	Proposed Method
<b>Ground Plane Segmentation (a)</b> (No. of Data Points)	–	14.2177 (120530)	–
<b>3D Point Clustering (b)</b> (No. of Data Points)	–	682.4080 (23488)	–
<b>Multi-depth Representation (c)</b> (No. of Data Points)	0.9270 (465750)	–	0.3090 (120530)
<b>Tracking For Single Obstacle (d)</b>	0.2530	4.2330	0.2610
<b>No. of Obstacles Per Frame (e)</b>	05	18	14
<b>Total Time Per Frame</b> <b>= a + b + c + (d × e)</b>	2.1920	772.8197	3.9630

EODT works with depth images, and the resolution of the depth image is  $375 \times 1242$  in the given data sequence. Therefore, it processes more than 465 thousand pixels per image, but the algorithms take less than one millisecond to create multiple depth representations even after processing such a huge number of pixels per frame. The simplistic linear algorithms with multi-threaded implementations allow us to achieve such quick processing times.



FO3D2D takes LiDAR point clouds as input, where the point cloud size is about 120 thousand on average. This means, there is a 3.86 fold reduction in the total amount of data to be processed in the case of LiDAR point cloud compared to depth image. The FO3D2D performs ground plane removal at the beginning using a filtering method, which takes about 14.2 milliseconds on average. The next module, i.e., 3D point clustering, takes about 682.4 milliseconds on average, and this module accounts for about 88.3% of the total processing time. These details indicate that obstacle estimation with LiDAR point clouds takes much more time even with fewer data compared to obstacle estimation from depth images. The ground plane segmentation and the 3D point clustering modules are mainly responsible for such behavior. We found that FO3D2D runs at 1.3 Hz in our implementation.

In our proposed method, we use LiDAR point clouds, where the number of points to process is about 120 thousand (similar to FO3D2D). Our proposed method is free from ground plane removal and 3D point clustering, which are the major time-consuming parts of existing LiDAR-based obstacle estimation methods. The time complexities of our proposed Algorithm 3: U-Depth Representation( ) and Algorithm 4: Restricted V-Depth Representation( ) are similar to those of EODT, and therefore, these algorithms jointly take about 0.3 milliseconds to process 120 thousand 3D data points. The major time-consuming module in the present system is still the formation of the multiple depth representation, as shown in the last column of Table 5.4. The average number of obstacles (both static and dynamic) per frame is the highest in FO3D2D because the system tracks all the obstacles around  $360^\circ$ , whereas the average number of obstacles in EODT is the lowest because of the small stereo depth range. The proposed system tracks obstacles that are present on the frontal side of the vehicle within a horizontal FOV of  $129.47^\circ$ . The average number of obstacles per frame is significantly higher in the proposed method compared with EODT, which increases the per-frame tracking time and results in a slight increment in total time in the proposed method. We achieve about 0.3 to 0.7 milliseconds on average for a single obstacle to detect and track, which is a significant improvement from the SoA methods. The LiDAR frequency is 10 Hz, and that implies, we get 100 milliseconds to process each captured point cloud. The proposed system takes about 7 to 10 milliseconds to track obstacles, considering there are 14 obstacles in every point cloud. Comparing the performance reported in Table 5.3 and Table 5.4, we may conclude that the performance of the proposed method is superior to the SoA methods.

## 5.5 Conclusions

This work presents a 3D obstacle detection and tracking system using 3D LiDAR point clouds, which aids in the autonomous navigation of robots in dynamic environments. In this work, we utilize u-depth and restricted v-depth representations from LiDAR point clouds to detect obstacles and estimate their locations and dimensions. The multiple depth representations enable us to bypass computation-intensive modules, achieving a faster method compared to SoA methods. The concept of u-depth and restricted v-depth representations from LiDAR point clouds in the proposed method yields satisfactory results, making it suitable for real-time applications. The proposed system performs well in various scenarios, including indoor and outdoor environments, multiple static and moving obstacles, and obstacles of different shapes and sizes. It undergoes evaluation with multiple open datasets and self-captured simulated datasets. Additionally, the proposed system is evaluated through real-time executions with data captured using robotic platforms. The proposed method shows a significant improvement over existing methods in dynamic obstacle detection, demonstrating its ability to detect very thin obstacles, such as slender poles on the road. Our future goal is to integrate the proposed method with other necessary modules for autonomous navigation in cluttered environments.

With this proposed obstacle estimation method using LiDAR, the system exhibits remarkable improvement from [2] and underscores the efficacy of LiDAR over visual cameras. The same efficiency is also demonstrated for the self-localization task in [132]. Given the advantages of LiDAR over visual cameras, we aim to develop an inspection application, wherein multiple UAVs perform autonomous navigation in a collaborative way using LiDAR sensors. The next chapter will present a system for collaborative inspection that utilizes LiDAR sensors to facilitate accurate autonomous navigation.



## Chapter 6

# Autonomous Aircraft Inspection using Collaborative UAVs

### 6.1 Introduction

In recent years, research on UAVs has rapidly expanded, and UAVs are now being developed for a myriad of purposes, including search and rescue operations in natural disasters, mine operations and safety monitoring, photography, and many more. In Chapter 4, we have presented a framework for collaborative VI-SLAM for multi-robot localization. Additionally, in Chapter 5, we introduced a fast obstacle detection technique using LiDAR, highlighting LiDAR's advantages over cameras in terms of accuracy, operating range, and resistance to environmental effects. The availability of these novel techniques motivates us to create a collaborative self-localization framework using LiDAR. With this objective, we present an autonomous visual inspection system with multiple UAVs. The system proposes a novel idea for collaborative self-localization using LiDAR sensor based on research work [140]. Participating UAVs utilize the presented obstacle detection techniques employing LiDAR.

To this end, one interesting field that has emerged is Flight Inspection using UAVs. When it comes to operational safety in civil aviation, flight inspection plays an important role in ensuring the proper functioning of navigation and sensing equipment. Flight inspection encompasses various dimensions, such as special inspection, production inspection, periodic inspection, and pre-flight inspection. An external pre-flight assessment is crucial for mitigating airplane flight hazards by identifying damaged or defective external sensing and control components. Our focus lies in pre-flight visual inspection using collaborative UAVs, an area with limited existing works. Flight inspection using UAVs can be divided into two parts: the air part and the ground part. This work focuses only on the air part. Most of the literatures available involve UAVs either manually controlled by a pilot or following a predefined inspection route [141]–[145]. This limits the effectiveness and speed of the UAV. Moreover, manual control can be

challenging for a pilot to navigate certain areas of the aircraft with limited mobility, such as under the fuselage and on top or under the main and tail wings. A potential error by the UAV pilot may result in substantial repair costs. Another enhancement lies in using multiple UAVs to complete inspections in reduced time. Manual control of multiple UAVs by a pilot is challenging, requiring more human resources and proper coordination. Hence, autonomous collaborative UAVs are the best choice for scenarios where navigation should work in any weather, lighting conditions, or GPS denied areas.

In this work, we propose a novel framework deploying multiple UAVs for the aircraft inspection task. UAVs can take off from any location near the aircraft and register with a single coordinate frame. Participating UAVs distribute the entire inspection task among them based on their locations and autonomously navigate to the corresponding inspection points sequentially. Each UAV is equipped with a front-facing RGB-D camera, a Velodyne 3D LiDAR with 64 channels, and one IMU. UAVs navigate using LiDAR and IMU measurements, while the inspection process uses measurements from the RGB-D camera. All UAVs navigate and perform the inspection task independently onboard.

The rest of this chapter is organized as follows: Section 6.2 presents a literature survey of pre-flight inspection tasks using robots. Section 6.3 describes the design of the proposed system and our contributions. Section 6.4 presents a series of experimental evaluations on self-captured simulated datasets. Finally, Section 6.5 concludes the chapter.

## 6.2 Background Study

The primary objective of using UAVs for pre-flight inspection is to significantly reduce the inspection time where a human can take hours to complete the task. A system capable of executing required tasks repeatedly and reliably at much higher frequencies, while accurately localizing itself in relation to the aircraft, is essential. This ensures repeatability and makes the whole system more generalized. Silberberg *et al.* [141] employed a Coverage Path Planning (CPP) approach in which the algorithm generates some initial way-points for maximum coverage of the aircraft. Subsequently, offline path planning is performed to complete the inspection task. Miranda *et al.* [142] have presented a UAV-based inspection for exterior screws, where manually defined way-points are used for the UAV to traverse and capture images. These images are then processed by a Convolutional Neural Network (CNN) for object detection. Maeda *et al.* [146] have utilized two subsidiary UAVs to assist the localization of a main UAV performing the inspection task, though the UAVs were not operational in parallel. Ruiqian *et al.* [147]

have utilized ArUco markers for positioning UAV and carrying out the inspection task. Boyong He *et al.* [143] have employed a safety pilot with GPS for localization and cascade Region-based Convolution Neural Network (RCNN) [148] for inspecting the aircraft's surface. Dario Cazzato *et al.* [144] have utilized a static pre-planned trajectory and have employed FAST and ORB [149] point features for feature matching. Konstantinos Malandrakis *et al.* [145] also have described a similar offline trajectory generation for conducting Non-Destructive Testing (NDT) using ultraviolet lights to inspect the surface of the aircraft.

Turning our attention to the collaborative localization aspect, we have presented a collaborative VSLAM framework in Chapter 4 with heterogeneous cameras that operates on a client-server based model and is able to handle noisy IMU measurements, but LiDAR data is excluded. The proposed system identifies matches when multiple robots observe common landmarks. In all SoA collaborative SLAM systems studied in Chapter 2 and Chapter 4, a common landmark must be viewed by participating robots to merge their individual maps. However, this requirement impacts the final completion time of the systems, making this technology impractical for high-throughput systems. Consequently, there is a demand for a robust system that can accurately register multiple maps of participating UAVs in a single coordinate frame, even when UAVs start from different locations, do not observe any common landmarks, and collaboratively execute the inspection task in parallel. The system should also be capable of online path planning while avoiding dynamic obstacles. The proposed work aims to address these shortcomings by adopting a different approach, as discussed in the upcoming sections.

## 6.3 System Design

Fig. 6.1 depicts our proposed system, where an autonomous UAV is equipped with three main types of sensors: an IMU, a Velodyne 3D LiDAR, and a RGB-D camera. The system comprises multiple modules, as illustrated in the block diagram, where our implemented modules are highlighted in blue. We assume that (i) the gravity direction and biases of the accelerometer and gyroscope are estimated in the base\_link coordinate frame  $B$  of the UAV, (ii) point clouds, captured using the LiDAR, are rectified and aligned with the Velodyne coordinate frame  $L$ , (iii) point clouds captured using the RGB-D camera, are also rectified and aligned in the camera coordinate frame  $C$ . The sensors are placed in fixed positions on the UAVs, leading to fixed transformations among all the coordinate frames. Let us consider that  ${}^B\mathbf{T}_L$  is the fixed transformation from the coordinate frame  $L$  to the coordinate frame  $B$ , and  ${}^B\mathbf{T}_C$

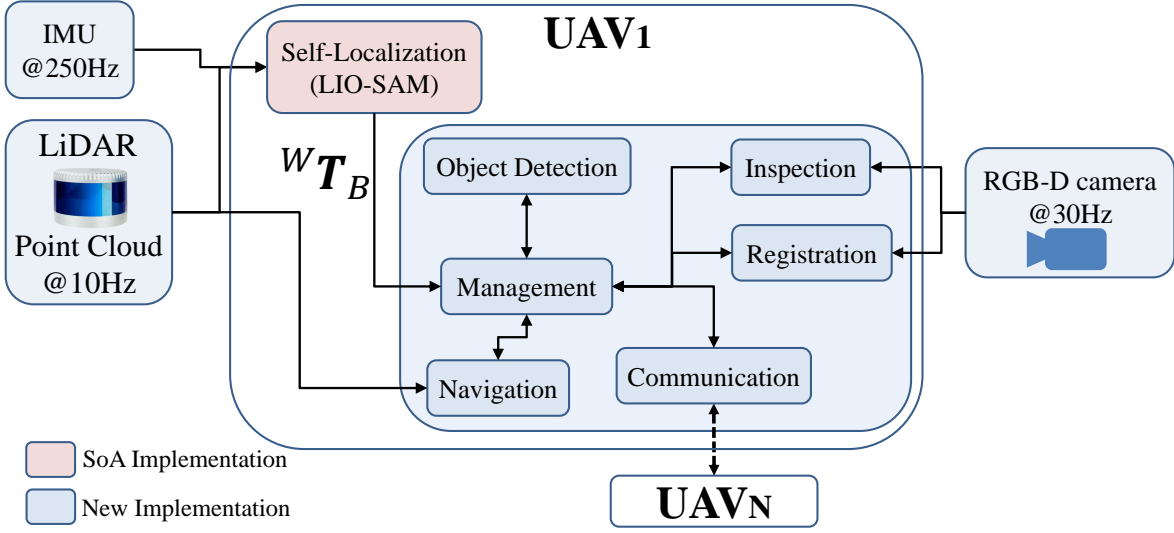


FIGURE 6.1: The proposed system block diagram for autonomous visual inspection of aircraft using multiple UAVs.

is the fixed transformation from the coordinate frame  $C$  to the coordinate frame  $B$ . Fig. 6.2 illustrates the coordinate frames and their corresponding transformations on a simulated Hector quadrotor [150].

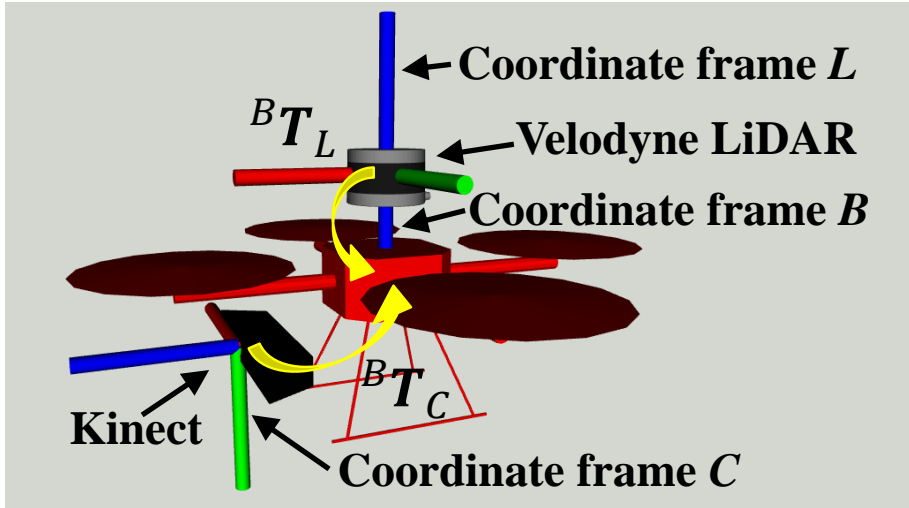


FIGURE 6.2: A Hector quadrotor carries a Kinect RGB-D camera and a Velodyne 3D LiDAR. The three coordinate frames  $B$ ,  $L$ , and  $C$  are shown, where  $X$ ,  $Y$ , and  $Z$  axes are in red, green, and blue, respectively. The transformations  $B T_L$  and  $B T_C$  are annotated with yellow arrows.

The first module of the proposed system is the self-localization module. We employ the state-of-the-art LiDAR-inertial system LIO-SAM [132] as the self-localization module, which receives IMU measurements and point clouds from the LiDAR and estimates the robot poses (pose of the coordinate frame  $B$ ) in the fixed world coordinate



frame  $W$  using tightly-coupled LiDAR-inertial odometry estimation via smoothing and mapping. Assuming  ${}^W\mathbf{T}_B$  is the estimated pose at any given time instance, each UAV runs LIO-SAM independently, and thus, each UAV has its own world coordinates. Refer to [132] for a detailed description of LIO-SAM.

The navigation module is primarily responsible for estimating the path for navigation and precise controlled movement, avoiding all obstacles. The Registration module performs registration between multiple UAVs to execute the inspection task collaboratively. The object detection module identifies multiple objects of interest for inspection, whereas the inspection module carries out the inspection task. The management module controls the task flow and data flow management to ensure synchronization between modules. The communication module is responsible for information flow between the participating UAVs. We explain the task flow of the proposed system below to enhance understanding of its operations.

### 6.3.1 Operational Flow

The primary objective of the present system is to conduct an autonomous visual inspection task, wherein a UAV identifies specific portions of the aircraft body and performs inspections. We divide an aircraft model into multiple landmarks (e.g., nose, left engine, right engine, front landing gear, left wing, right wing, tail, etc.) for a UAV to identify. We assume that a 3D model of the aircraft is available a priori. In cases where the aircraft model is not available, a model is created by manually flying a UAV around the aircraft and generating a map with LIO-SAM. The flow of different tasks is discussed below.

- (i) UAVs are positioned near any of the landmarks of the aircraft, and the RGB-D camera faces towards the aircraft, allowing the UAVs to takeoff freely.
- (ii) After takeoff, the UAV first identifies the nearest landmark and then self-positions to achieve a better view. Subsequently, the UAV registers with a fixed coordinate frame,  $F$ , wherein the templates were captured. All participating UAVs register with the fixed coordinate frame  $F$  from various locations and landmarks. This is one of our major contributions in this work, where all UAVs register with the coordinate frame  $F$  without viewing a common landmark.
- (iii) This novel registration method enables UAVs to distribute landmarks based on their nearest locations, significantly speeding up the inspection task.
- (iv) The UAV identifies multiple objects of interest requiring inspection around the current landmark.

- (v) Subsequently, the UAV estimates poses, allowing for the best possible view of each identified object, and the UAV sequentially self-positions with these poses.
- (vi) The UAV performs the inspection task with captured point clouds from the RGB-D camera.

The management module in Fig. 6.1 is the core of our system, primarily managing synchronization among other modules, while the navigation module takes care of precise restricted navigation. The communication module uses the ROS [101], [111] message passing method for all communication among different modules and across UAVs.

### 6.3.2 Simulated Environment

We validate our approach in the Gazebo simulator [95]. A view of the environment resembling an airport is shown in Fig. 6.3. It consists of static models of an anonymous

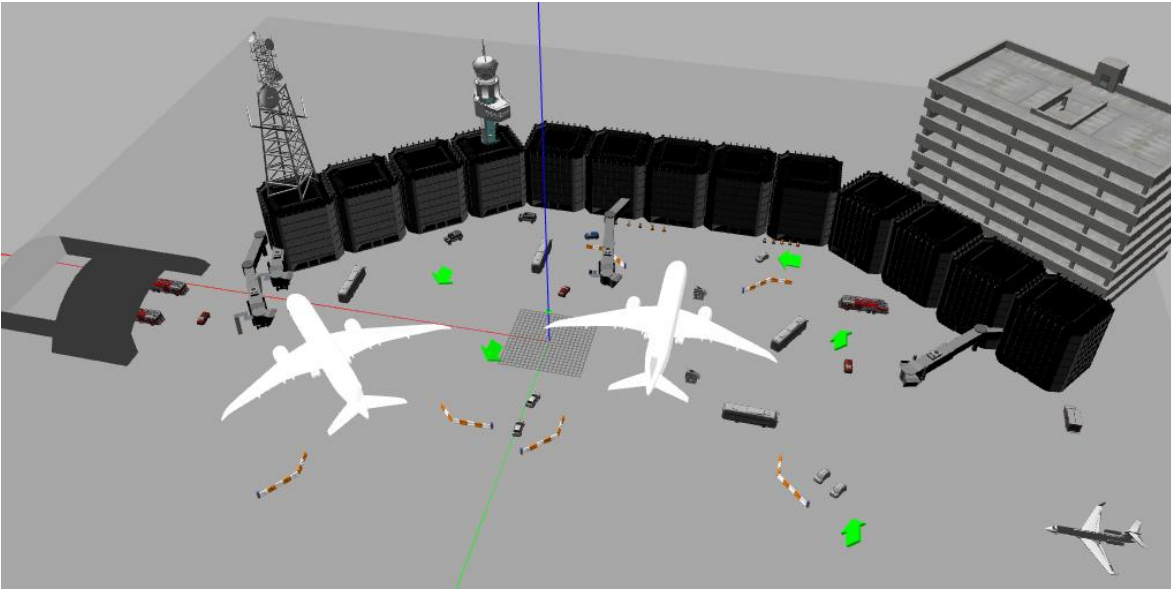


FIGURE 6.3: Gazebo model of an airport environment.

aircraft, approximately 52 meters long, with a 55.5 meters wing span, and a maximum height of 14 meters. Additionally, typical objects found in an airport setting, such as boarding bridges, ladders, ground vehicles, boards and markers are also included. These objects provide necessary geometric features required for better map estimation using LiDAR-Inertial localization around the aircraft. We require to construct a 3D representation of the aircraft geometry using LiDAR-Inertial localization if one is not available a priori. Note that the presence of these external geometric features is not essential for the inspection process once a 3D model of the aircraft is made available.

The inspection task is intended to inspect multiple sensors that are present on the outer surface of the aircraft during the pre-flight maintenance or periodic maintenance phases. The anonymous aircraft model, that we use in the Gazebo simulated environment, does not contain any sensors, e.g., Angle of Attack sensor, TAT sensor, ATC antenna, Airborne Collision Avoidance, ice detector, Air temperature sensor, etc., for inspection. Therefore, we attempt to mimic such sensors using specific geometrically shaped colored patches on the aircraft surfaces with the help of the open-source Blender software [151]. Fig. 6.4 shows multiple patches on the aircraft surface. In the simula-

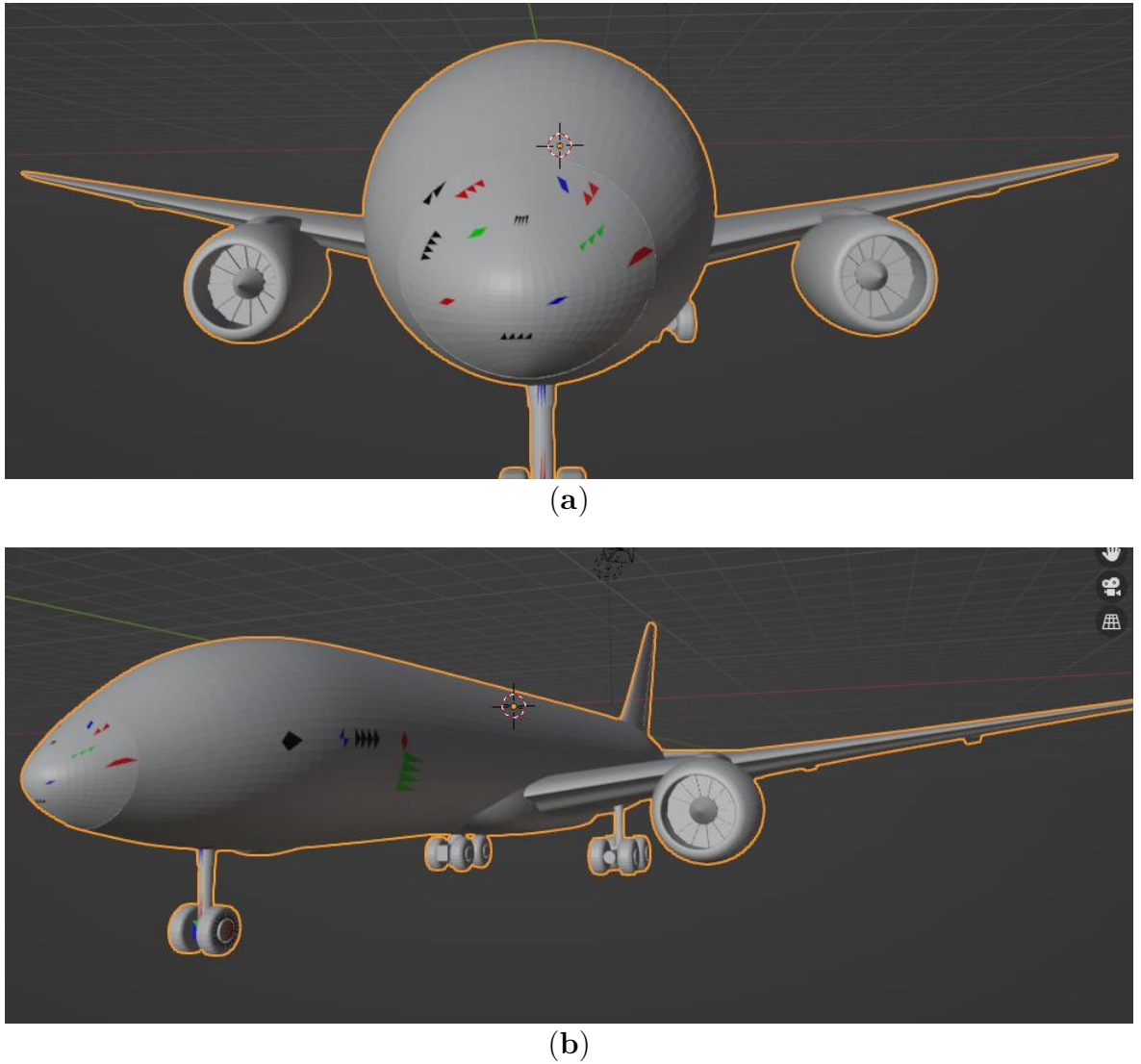


FIGURE 6.4: Multiple geometrical shaped colored patches on the aircraft's outer surface.

tion environment, we use the Hector quadrotor [150] with a Kinect sensor [42] mounted as the forward-looking RGB-D camera and a Velodyne 3D LiDAR on top, as shown in Fig. 6.2.

### 6.3.3 Template Generation

We will first discuss template generation, which involves creating a three dimensional model of the aircraft with the highest possible accuracy. The process is offline with a single execution. In this process, we manually fly a UAV around the aircraft and use LIO-SAM to generate a map of the environment. The speed of the UAV is restricted within a specific range for optimal performance by LIO-SAM. Fig. 6.5 illustrates the outcome of the LIO-SAM execution pictorially. The LIO-SAM generates the map in a

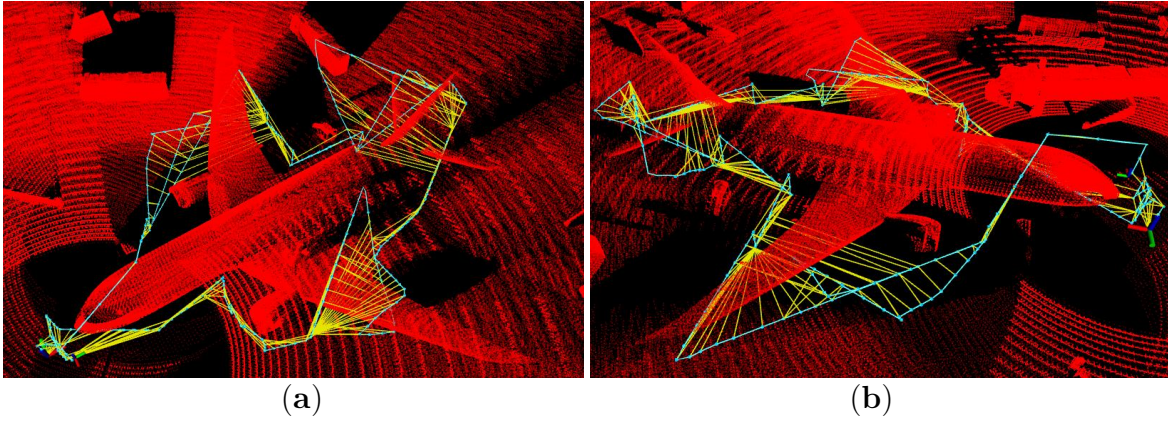


FIGURE 6.5: The map output from LIO-SAM visualized from different view points.

fixed world coordinate frame, denoted by  $F$  as mentioned above. The aircraft model is then extracted manually by deleting map points that do not belong to the aircraft body surface. Fig. 6.6 shows the aircraft model from multiple view points extracted from the LIO-SAM map. The aircraft model does not contain any simulated objects (colored geometric patches) because the point cloud is without any color information. Afterward, the landmark regions are segmented to create templates for each landmark. Fig. 6.7 shows two sample templates created from the aircraft model. All the templates are created in the coordinate frame  $F$ , and the location of the centroid of each template is stored with a fixed transformation  $\mathbf{T}l_F$ .

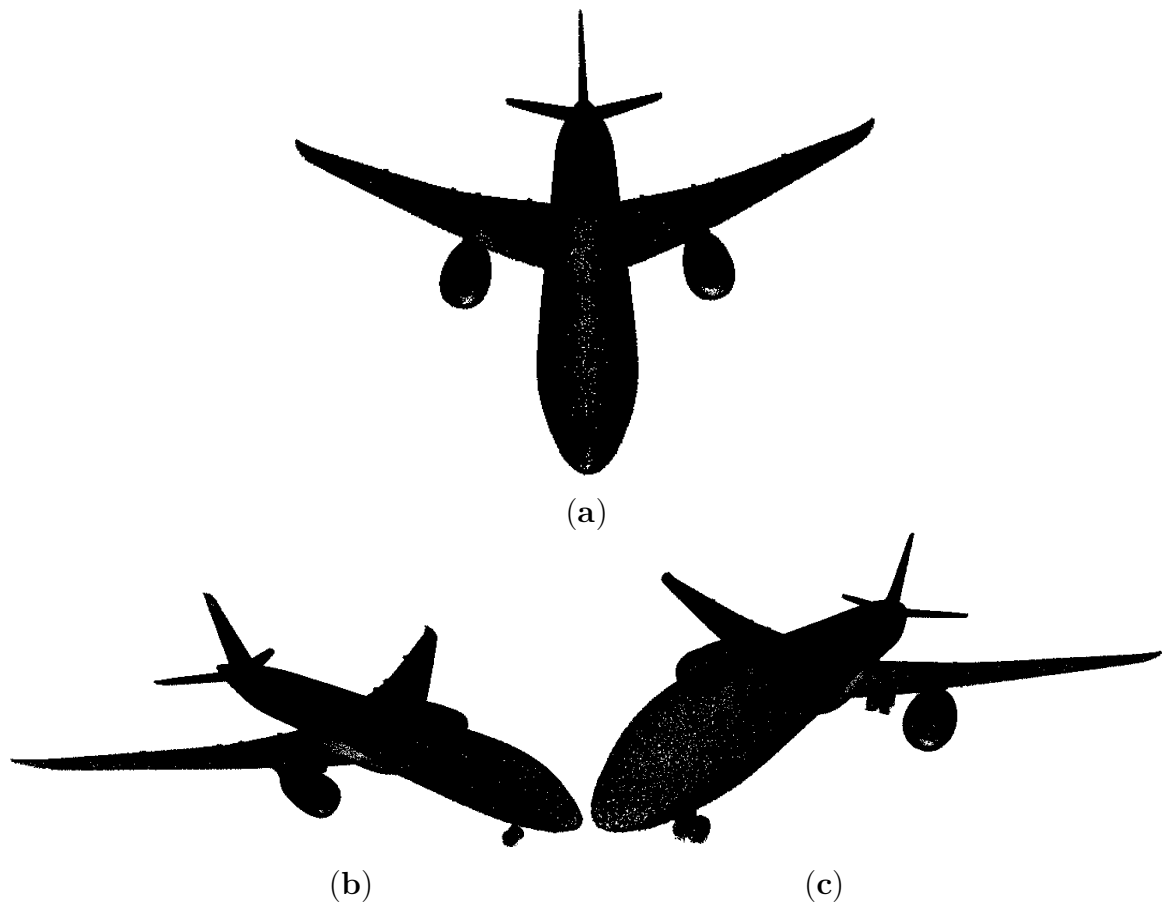


FIGURE 6.6: Aircraft model extracted from the LIO-SAM map and visualized from multiple view points.

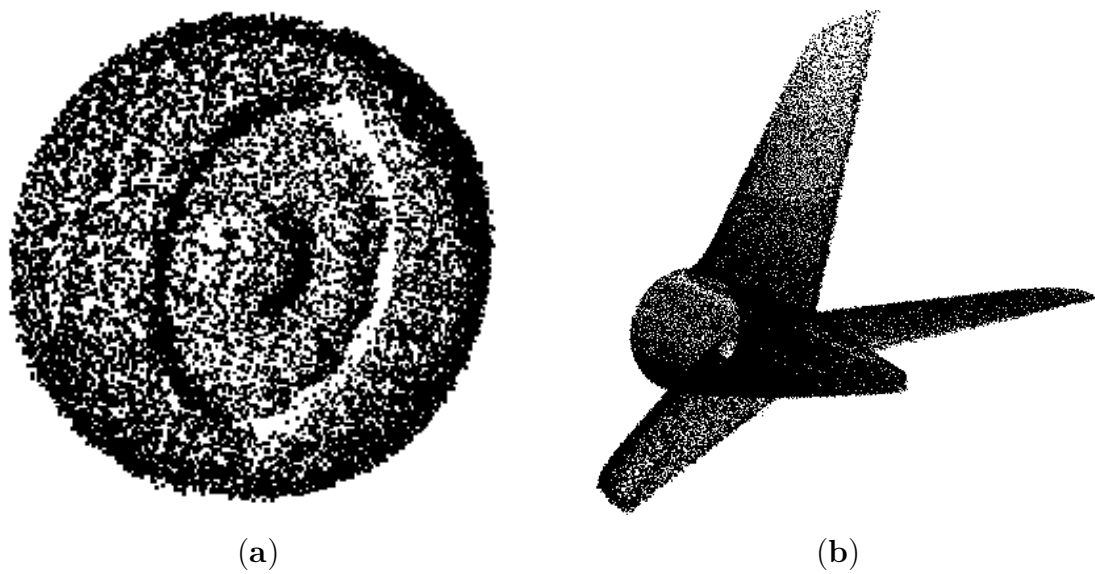


FIGURE 6.7: Templates of different landmarks are created from the aircraft model: (a) Template of the left engine, (b) Template of the tail.



### 6.3.4 Navigation

The navigation module guides the UAV to traverse safely from its current location to the desired goal location with a given goal pose. The navigation module consists of two parts: Global planning and Local planning.

#### 6.3.4.1 Global Planning

We utilize the map generated from LIO-SAM for global planning. The point cloud data are converted to Octomap [134], providing information about the planning space. Fig. 6.8 shows snapshots of the planning space in Octomap representation. We use

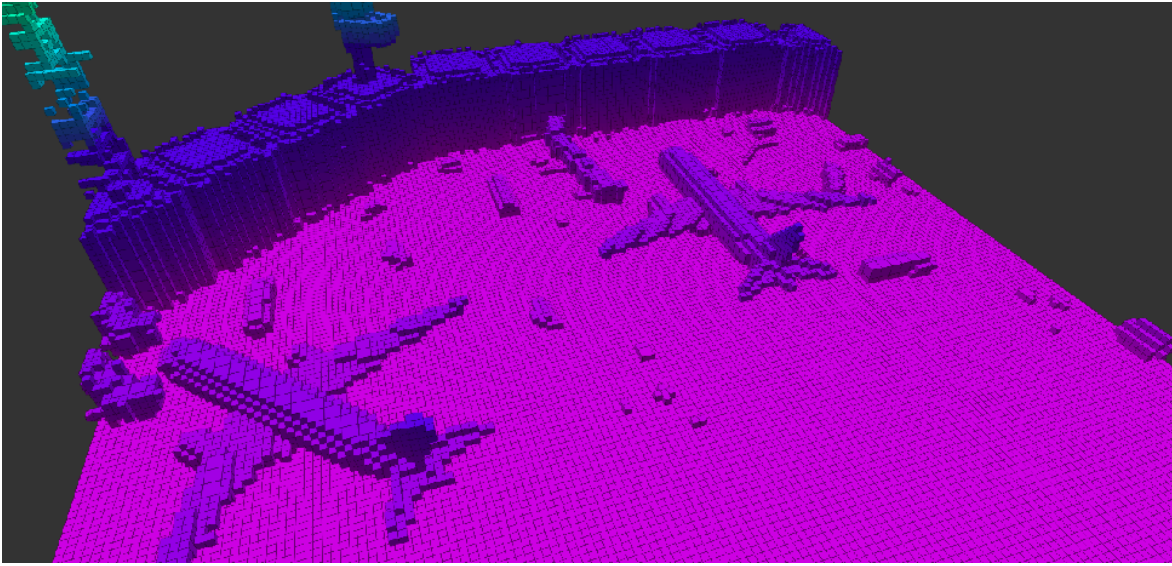


FIGURE 6.8: Octomap representation of the simulated airport created using the LIO-SAM point cloud.

RRTConnect [152] to produce the global path. Being a sampling-based algorithm, RRTConnect also requires a collision checker, which validates the sampled states. We use the Flexible Collision Checking Library (FCL) [153] as the collision checker.

Safe navigation is a major concern when any UAV moves near the aircraft. Therefore, the planner segments the entire planning space into a hierarchical order. Fig. 6.9 shows the planning space with two colored regions (yellow and red) around the aircraft model. The red regions are closest to the aircraft body and, therefore, considered as no-fly zones. The yellow regions are further from the red regions and are considered as restricted buffer zones. Therefore, the global planner always generates paths outside the yellow regions. If any UAV mistakenly enters the no-fly zones, it must execute the following steps sequentially for safe navigation: (i) immediately abort the mission and notify the mission leader UAV (discussed in Section 6.3.5), (ii) move to the nearest yellow zone, and (iii) navigate to the home location and land safely.

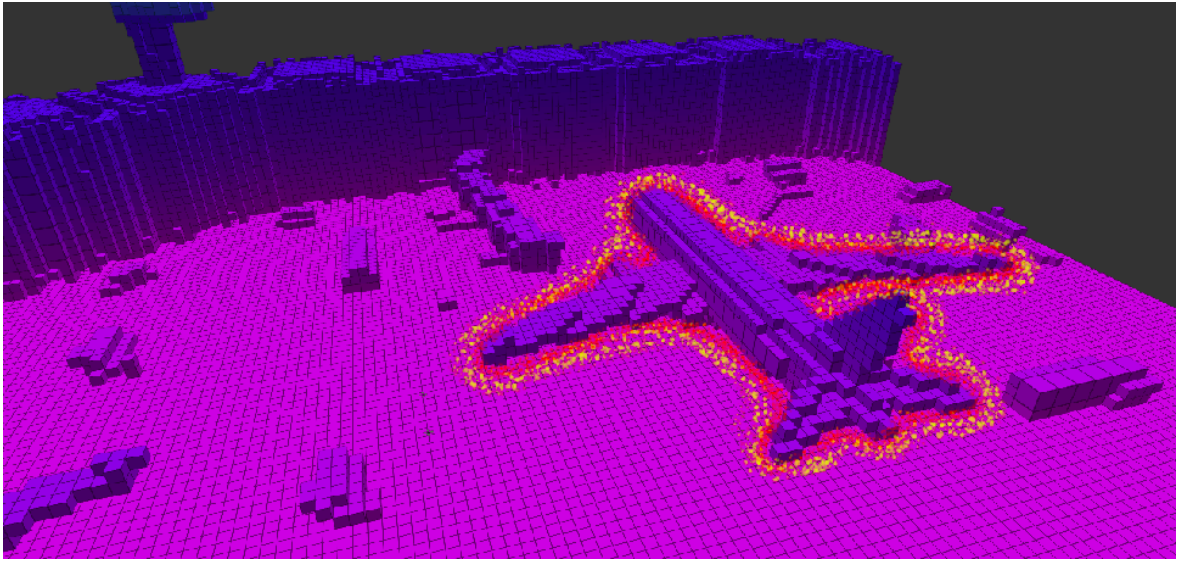


FIGURE 6.9: Bounded regions (red and yellow) for safe UAV operation.

#### 6.3.4.2 Local Planning

Although we have a collision-free global plan, it is possible that any dynamic obstacle may come close to the robot. For example, when the UAV tries to reach the way-points (taken from the global plan) closer to the ground, any unknown dynamic obstacle such as a ground vehicle, humans, etc., may come very close to the robot. Such dynamic obstacles are not considered in global planning; therefore, handling such situations using the global plan is not possible, and we must use a local planner to avoid such dynamic situations. We have defined a FOV of five meters in radius around the UAV to tackle such unknown obstacles. The UAV continuously checks the validity of the next global way-point inside the FOV. If the way-point is found to be invalid, a fresh valid way-point is generated inside the FOV; next, we generate a local plan using Probabilistic Road Map (PRM) [154]. The PRM keeps a careful check on the available free space by using the occupied information from the obstacle estimation module. The details of the obstacle estimation module are presented in Chapter 5. The PRM ensures a safe local plan is generated if any unknown obstacle is found. So, we need both global and local planning algorithms for safe navigation. Algorithm 5: Local Path Planning( ) describes the steps of the local path planning, as described above.

#### 6.3.5 Registration

Each UAV performs self-localization and map creation in a fixed world coordinate frame, where each robot has its own world coordinates. We can only comprehend the relative locations of other UAVs once the localizations of all UAVs are in a single



**Algorithm 5** : Local Path Planning( )**Input:** Octomap from point cloud data of LIO-SAM, GoalPose**Output:** Safe plan for UAV to navigate

---

```

1: Initialization :  $GlobalPlan \leftarrow RRTConnect(GoalPose)$ 
2: for  $waypoint \in GlobalPlan$  do
3:   if (waypoint valid and inside FOV) then
4:      $localpath \leftarrow PRM(waypoint)$ 
5:   else if (waypoint not valid or waypoint not in FOV) then
6:      $new\_waypoint \leftarrow Neighborhood(waypoint)$ 
7:      $localpath \leftarrow PRM(new\_waypoint)$ 
8:   end if
9:   for  $local\_waypoint \in localpath$  do
10:     $uav\_motion \leftarrow UAVControl(local\_waypoint)$ 
11:   end for
12: end for

```

---

coordinate frame. The registration process aims to estimate transformations from all individual world coordinates to another fixed coordinate, allowing us to estimate the relative locations of every UAV in the fixed coordinate frame. Therefore, the registration process is the core of collaborative navigation. Better registration always contributes to a more accurate understanding of the relative positions of all UAVs and subsequently aids in distributing landmarks among the participating UAVs. Fig. 6.10 shows the flow diagram of registering a UAV to the fixed coordinate frame  $F$ . The detailed steps of the registration process are described below.

1. The registration process starts with template matching, where a template point cloud ( $PC_t$ ) is matched with the point cloud captured using the RGB-D camera.
  - (i) The RGB-D point cloud,  $PC_{RGBD}$ , is a dense point cloud, but the templates are not as dense as RGB-D point clouds. Therefore, we first down-sample the RGB-D point cloud using the VoxelGrid process [155], where the entire visible space is segmented into sets of tiny 3D voxels or boxes, and all the points present inside a single voxel will be approximated with its centroid. The down-sampling converts  $PC_{RGBD}$  with a similar point density as the templates, and let us name this down-sampled point cloud as  $PC_{DS}$ , which is in the coordinate frame  $C$ .
  - (ii) The system proceeds by matching all templates to the point cloud  $PC_{DS}$  and identifies the best match. The matching process relies on Fast Point Feature Histograms (FPFH) [156], a streamlined version of the original Point Feature Histograms (PFH) [157]. PFH is a 3D point cloud processing technique

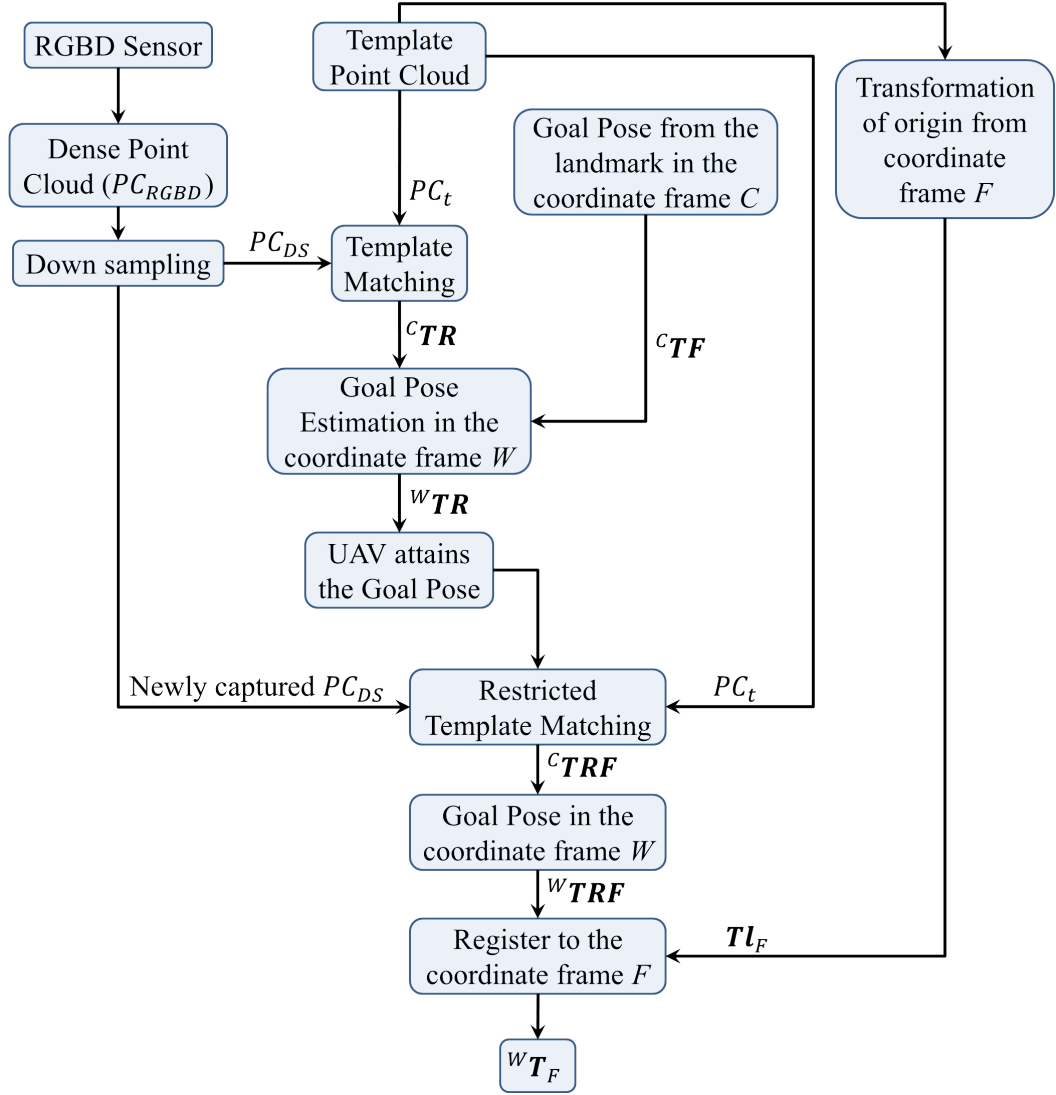


FIGURE 6.10: Process flow diagram for registering the world coordinate frame  $W$  of a UAV to the fixed coordinate frame  $F$ .

designed to capture detailed geometric relationships between neighboring points, offering a rich description of local surface features. PFH encodes three main types of geometric information for every pair of points in the neighborhood:

- **Angular features** such as the angle between surface normals at each point, providing insight into local curvature.
- **Distance metrics** between points, which help quantify spatial relationships.
- **Azimuthal and elevation angles** between points in the neighborhood, representing the directionality of surface changes.

This rich encoding of local geometry results in a high-dimensional histogram that accurately captures the surface's shape. However, PFHs comprehensive pairwise feature calculations make it computationally expensive. FPFH simplifies the original PFH approach by reducing the complexity of these pairwise calculations. Instead of evaluating every pair of points in the local neighborhood, FPFH divides the computation into two stages: first, it computes a simplified PFH for each point, using only the point and its direct neighbors, and then aggregates the information across these reduced calculations. This results in significantly faster computation while still preserving sufficient geometric detail for tasks like matching and alignment.

The FPFH technique uses a Sample Consensus method to match the template point cloud ( $PC_t$ ) with the target point cloud ( $PC_{DS}$ ), computing a rigid transformation ( ${}^C\mathbf{TR}$ ) that aligns the template to the target. A match is considered successful when the deviation between the two point clouds, measured as the average Euclidean distance between corresponding 3D points, falls below a set threshold  $Th_1$ .

2. The next step is to place the UAV in a way from where the landmark would be visible in the best possible way. The transformation  ${}^C\mathbf{TR}$  provides a measure of the landmark location in the coordinate frame  $C$ . We consider a normal position of 3 meters away from the landmark as the position to achieve the best possible visibility of the landmark. The UAV can be commanded to a goal pose expressed only in the world coordinate frame  $W$ . Therefore, we generate a corresponding goal pose in the coordinate frame  $W$ . Equ. (6.1) shows the relation to generate a goal pose in the coordinate frame  $W$ .

$${}^W\mathbf{TR} = {}^W\mathbf{T}_B ({}^B\mathbf{T}_C ({}^C\mathbf{TR} ({}^C\mathbf{TF}))) \quad (6.1)$$

where,  ${}^C\mathbf{TF}$  represents the transformation of three meters away from the landmark in the coordinate frame  $C$ . We provide the goal pose  ${}^W\mathbf{TR}$  to the navigation module to attain the pose by the UAV. Fig. 6.11 shows the current pose and the goal pose in the coordinate frame  $W$  on Rviz visualizer [99]. We can consider the UAV is aligned with the landmark once the UAV attains the exact pose, but the UAV navigation is not always accurate. The navigation module assumes that a UAV has reached the goal pose when the UAV attains a pose which is very close to the goal pose, i.e., the Euclidean distance between the current UAV pose and

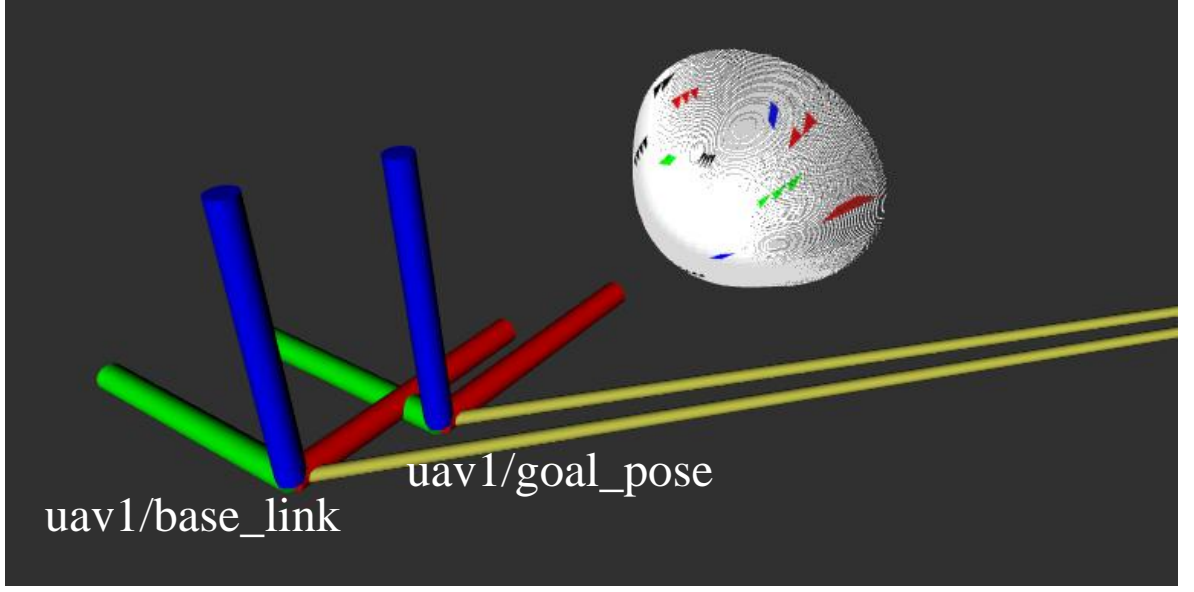


FIGURE 6.11: An Rviz snapshot is explaining the current pose ('uav1/base\_link') of the UAV and the goal pose ('uav1/goal\_pose') in the coordinate system  $W$  for reaching the corresponding landmark (Nose).

the goal pose is less than a threshold  $Th_2$ , and the yaw angle between the poses is less than a threshold  $Th_3$ .

3. The error in placing the UAV with respect to the exact goal pose ( ${}^W\mathbf{TR}$ ) may introduce a significant error to the registration process. Therefore, we reiterate the template matching process between the previously matched template  $PC_t$  and the current point cloud captured by the RGB-D camera, as shown in Fig. 6.10. The template matching procedure follows the same way as explained above in the first point but in a more restricted way. The restricted way is defined as setting a smaller value to the maximum distance between point correspondences and setting a bigger value to the number of iterations in FPFH matching, where these values produce more accurate matching with a refined transformation matrix  ${}^C\mathbf{TRF}$ . The transformation  ${}^C\mathbf{TRF}$  provides the landmark location from the current position of the UAV in the coordinate frame  $C$ , and we calculate the location of the landmark ( ${}^W\mathbf{TRF}$ ) in the coordinate frame  $W$  using Equ. (6.2).

$${}^W\mathbf{TRF} = {}^W\mathbf{T}_B ({}^B\mathbf{T}_C ({}^C\mathbf{TRF})) \quad (6.2)$$

4. Our next target is to establish a relation between the coordinate frames  $F$  and  $W$ .

The template is in the coordinate frame  $F$ , and the template is aligned with the landmark with a transformation  ${}^W\mathbf{TRF}$  in the coordinate frame  $W$ . Therefore, we can estimate a transformation between the coordinate frames  $W$  and  $F$ , and this transformation remains fixed as both the coordinate frames are fixed frames. The fixed transformation  ${}^W\mathbf{T}_F$  from the coordinate frame  $W$  to the coordinate frame  $F$  is formulated in Equ. (6.3).

$${}^W\mathbf{T}_F = {}^W\mathbf{TRF}(\mathbf{T}l_F) \quad (6.3)$$

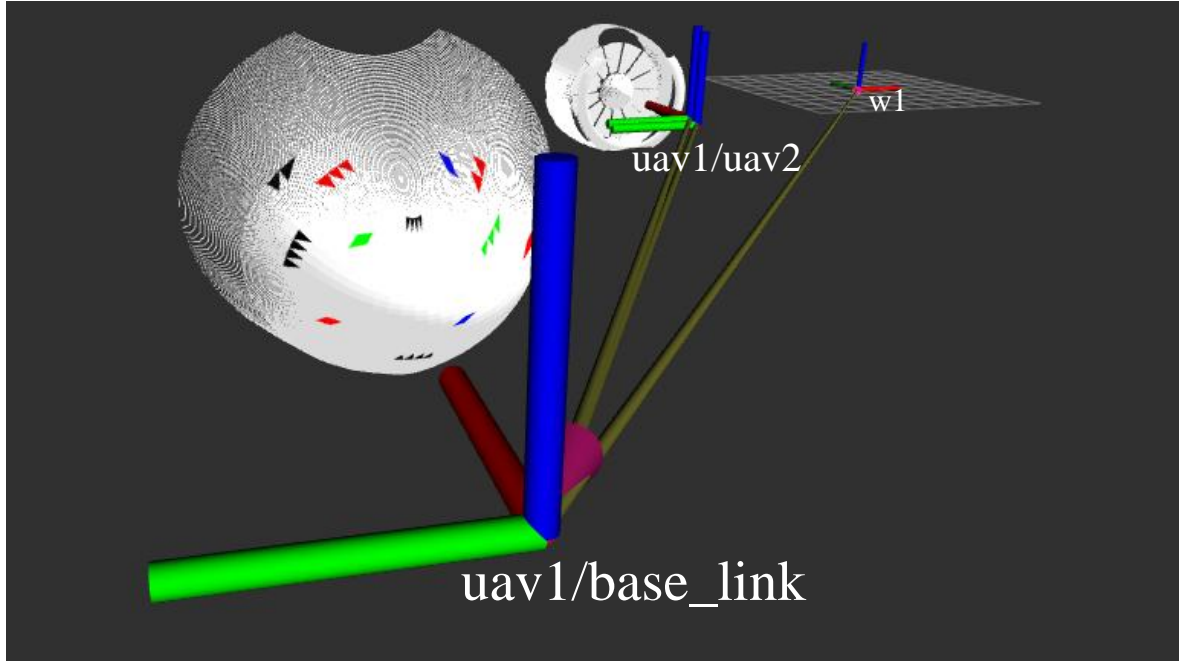
where,  $\mathbf{T}l_F$  is the transformation of the origin of the point cloud  $PC_t$  in the coordinate frame  $F$ . The fixed transformation  ${}^W\mathbf{T}_F$  allows the UAV to register with the coordinate frame  $F$ . Once a UAV registers with the coordinate frame  $F$ , the UAV gets the locations of all other landmarks in the coordinate frame  $F$  and also its own world coordinate frame  $W$ . All participating UAVs perform this process and register with the coordinate frame  $F$ .

5. The present system considers the first UAV, i.e., with id 1, as the mission leader, which means the  $UAV_1$  receives the corresponding fixed transformations  ${}^W\mathbf{T}_F$  from all other UAVs and subsequently all other UAVs start sending their current poses to the  $UAV_1$ . The  $UAV_1$  calculates the poses of all the UAVs with respect to its own base\_link frame. Equ. (6.4) shows the formulation to calculate the pose ( ${}^{uav1}\mathbf{T}_2$ ) for  $UAV_2$  from the base\_link frame of  $UAV_1$ .

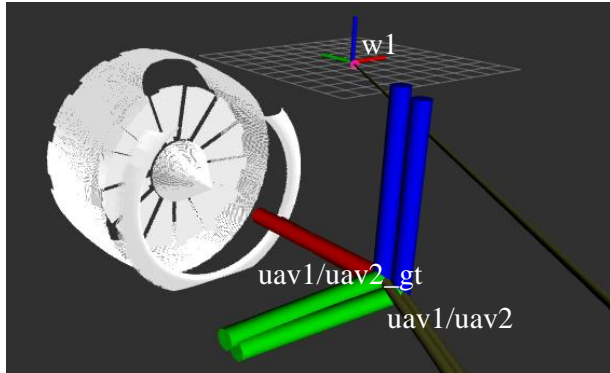
$$\begin{aligned} {}^{W2}\mathbf{T}_1 &= {}^{W2}\mathbf{T}_{2F} ({}^{W1}\mathbf{T}_{1F}^{-1} ({}^{W1}\mathbf{T}_{1B})) \\ {}^{uav1}\mathbf{T}_2 &= {}^{W2}\mathbf{T}_1^{-1} ({}^{W2}\mathbf{T}_{2B}) \end{aligned} \quad (6.4)$$

where,  $W1$  and  $W2$  represent the world coordinate frame of  $UAV_1$  and  $UAV_2$ , respectively. Therefore,  ${}^{W1}\mathbf{T}_{1B}$  and  ${}^{W2}\mathbf{T}_{2B}$  represent the current pose of  $UAV_1$  and  $UAV_2$ , respectively,  ${}^{W1}\mathbf{T}_{1F}$  and  ${}^{W2}\mathbf{T}_{2F}$  represent the fixed registration transformation for  $UAV_1$  and  $UAV_2$ , respectively.  $UAV_1$  distributes the landmarks among the participating UAVs based on the neighboring landmarks of every UAV. We adopt this simplistic design in the present system.

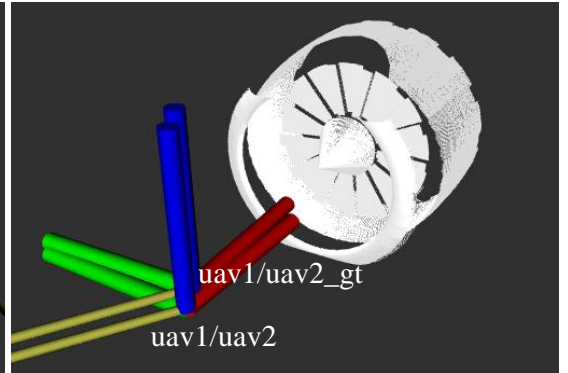
Fig. 6.12 shows a sample output of the registration process. Two UAVs are used in this process, where  $UAV_1$  is registered with the landmark ‘Nose’, and  $UAV_2$  is registered with the landmark ‘Left Engine’. The estimated pose of  $UAV_2$  from  $UAV_1$  is



(a)



(b)



(c)

FIGURE 6.12: The output of the registration with the coordinate frame  $F$  with two UAVs on Rviz visualizer: (a) The pose of  $UAV_2$  relative to  $UAV_1$  is indicated with an arrow from the base\_link of  $UAV_2$  ('uav1/uav2') to the base\_link of  $UAV_1$  ('uav1/base\_link'), (b)–(c) Closer views of  $UAV_2$ , where two poses indicate the estimated relative pose ('uav1/uav2') and the ground truth pose ('uav1/uav2\_gt') from  $UAV_1$ .

shown with a link between the two `base_link` frames of both the UAVs in Fig. 6.12(a), whereas Fig. 6.12(b) and (c) show the closer views of  $UAV_2$  with the ground truth pose ( $uav1/uav2\_gt$ ) and the estimated pose ( $uav1/uav2$ ). The closeness of both the poses indicates the estimation accuracy. A UAV starts the inspection task after receiving the list of landmarks from  $UAV_1$ . The inspection process starts with object identification, which is discussed in the next section.

### 6.3.6 Object Detection

The inspection task is based on inspecting different sensors or objects, e.g., Angle of Attack sensor, TAT sensor, ATC antenna, Traffic alert and Collision Avoidance, Airborne Collision Avoidance, ice detector, Air temperature sensor, etc., which can be visually checked from the outside, and each object has different inspection parameters. The inspection can only start after identifying a specific object. These types of objects, which have distinct geometrical shapes and create undulations on the surface of the aircraft, are missing in the simulated model of the aircraft. Therefore, we simulate these objects as small colorful geometrical shaped patches on the surface of the aircraft, as discussed in Section 6.3.2. We first use an object detection module to detect objects from the captured RGB images that are present around a landmark. The object detection is based on a learning-based detection method, and we use a well-established deep learning-based object classification tool, YOLO v3 [158], for this purpose. We choose YOLO v3 for its acceptable accuracy for such simplistic objects and fast execution to support real-time systems.

We use three patches around the ‘Nose’ landmark as the intended objects out of all the patches put on the aircraft surface and a very small set of training data (567 images with three classes) to train the model quickly and establish the capability of the object detection method. One sample output of the object detection module is presented in Fig. 6.13. Once the objects are detected, the next task is to estimate poses for the UAV to observe the objects in the best possible way for the inspection task. The steps to estimate and place the UAV to observe an object for the inspection task are discussed below.

1. The object detection module produces the coordinates of the bounding boxes (as in Fig. 6.13) of the detected objects on the RGB image plane. Our first task is to estimate the corresponding 3D bounding boxes. Let us assume a 2D pixel with coordinates  $(p_y, p_x)$  on the image plane; then, we calculate the corresponding 3D coordinates of the pixel in the coordinate frame  $C$  by 2D-3D relations as deduced in Equ. (A.11), and the current relation is presented in Equ. (6.5).



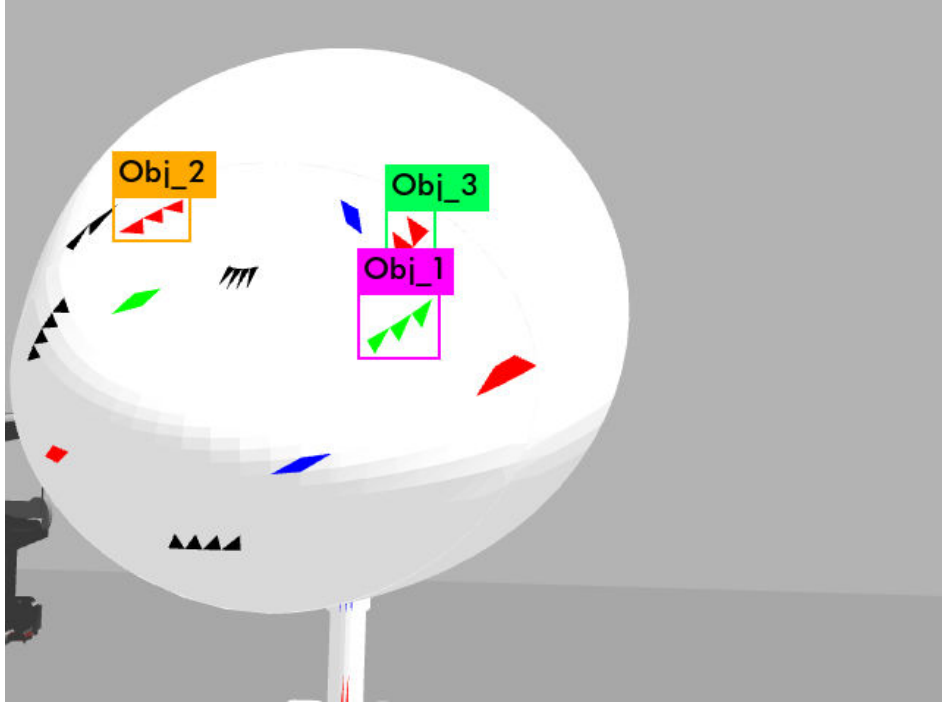


FIGURE 6.13: A sample output of the object detection module, where the detected three classes are annotated with bounding boxes and object IDs.

$$\begin{aligned}
 P_z &= D \\
 P_x &= (p_x - c_x) \times P_z / f_x \\
 P_y &= (p_y - c_y) \times P_z / f_y
 \end{aligned} \tag{6.5}$$

where,  $D$  is the depth value of the pixel location  $(p_y, p_x)$  taken from the depth image,  $(P_x, P_y, P_z)$  is the corresponding 3D point,  $(c_y, c_x)$  is the principal point of the camera, and  $f_x$  and  $f_y$  are the focal lengths in horizontal and vertical directions, respectively. Thus, we can get corresponding 3D bounding rectangles for each of the 2D bounding boxes.

2. Afterwards, we extract all the RGB-D points that fall within the 3D bounding parallelepiped and calculate the centroids. Now we create goal poses for each centroid, where the poses are at the same height as the centroid and looking towards the corresponding centroids from a user-defined distance. These poses are considered as the best view poses to capture the data for the corresponding objects for inspection. The poses are first calculated in the coordinate frame  $C$  and subsequently converted to the coordinate frame  $W$  using the similar transformation as

shown in Equ. (6.1). The navigation module uses these poses sequentially to position the UAVs for the inspection task. Fig. 6.14 shows snapshots of Gazebo [95] and the corresponding Rviz visualizer [99], where a UAV is positioned to get the best view of Obj\_1.

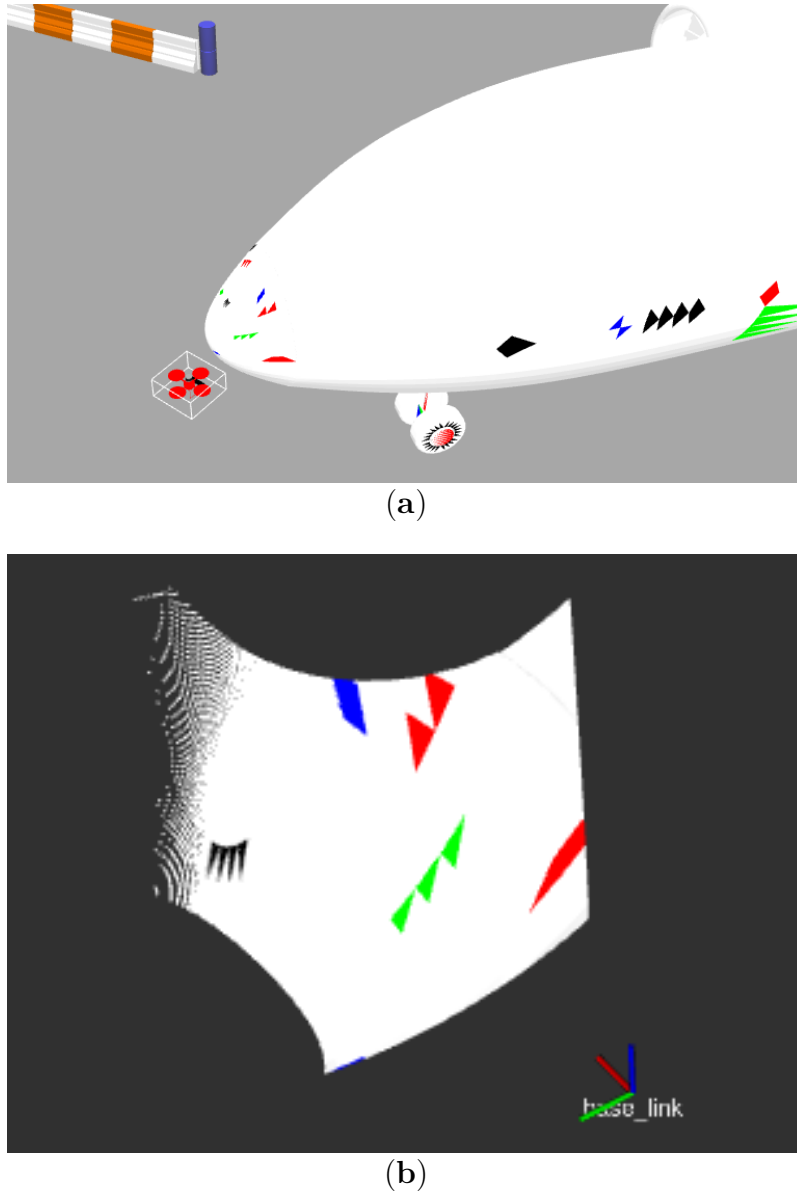


FIGURE 6.14: (a) A Gazebo snapshot illustrates a UAV positioned at the optimal viewing pose for Obj\_1, (b) The corresponding Rviz visualization displays the base\_link frame facing Obj\_1 (three consecutive green triangles) and the captured RGB-D point cloud, where Obj\_1 is visible.

### 6.3.7 Object Inspection

Once a UAV is positioned to the given goal pose to obtain the best possible view of the object, the inspection module starts. The inspection of the simulated objects is carried out in the form of area measurement and maximum distance measurement in the present system. The simulated object measurement is quite different from real objects on the aircraft surface. Therefore, a domain-specific learning-based inspection is required in real scenarios, but we present the measurement of simulated objects to demonstrate the capability of the system.

The inspection module uses RGB-D point clouds captured after reaching the best possible view goal pose, where the object is completely visible in the best possible way. We first use a color-based region-growing algorithm [159] to segment the points that form the colored objects. Afterward, these segmented points are used to measure the area and maximum distance on a metric scale. We measure the area of an object by measuring the areas of the individual triangles, where the area of a triangle is measured using the lengths of the three sides. The maximum distance is calculated by calculating the Euclidean distance between the two furthest corner points. The measured area and the maximum distance are shown pictorially in Fig. 6.15. Inspecting in a real

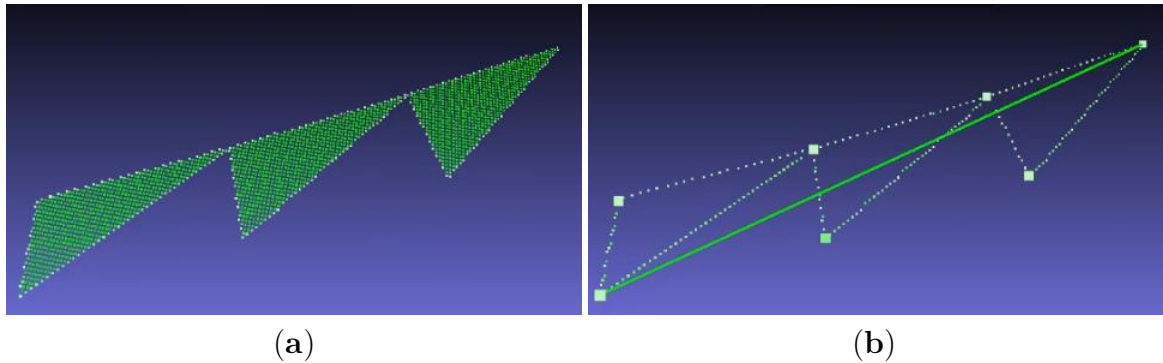


FIGURE 6.15: (a) The segmented point cloud for Obj\_1, representing the area of Obj\_1, (b) The calculated maximum distance using the two furthest ending corner points.

scenario, the UAVs are able to localize any specific anomaly on the aircraft surface in the coordinate frame  $F$  and also from the nearest landmark, which helps the service person to identify any anomaly quickly.

### 6.3.8 Communication Module

The communication module is responsible for the information sharing among the participating UAVs. All communications are implemented using the ROS message passing protocol, where the sizes of all types of messages are of nominal size and therefore do

not have many bandwidth requirements. Information sharing is only required at the time of the registration process in the proposed system; therefore, the system requires moderate network speed at the time of the registration process. The rest of the mission time requires negligible bandwidth for information sharing because information sharing happens only when any UAV is aborting the mission.

## 6.4 Experimental Results

We evaluate the proposed system only in a simulated airport-like environment, as described in Section 6.3.2. The inspection task is performed with two UAVs. The proposed system is ROS [101], [111] compatible and implemented with C++ and Python-based ROS nodes. The entire system runs on a standard workstation, having an Intel Core i7-9700k with 8 cores @ 3.60 GHz, 32 GB of RAM, and 8 GB of Nvidia RTX2070 Graphics. We present accuracy measurements for registration and object measurement processes with the simulated data. Table 6.1 shows the values of the parameters that are set experimentally in the implementation of the present system for all of our experiments.

TABLE 6.1: Parameter Values in The Proposed System

$f_x$	$f_y$	$c_x$	$c_y$	$Th_1$ (meter)	$Th_2$ (meter)	$Th_3$
554.255	554.255	320.5	240.5	0.004	0.03	1°

### 6.4.1 Navigation and Path Following

We have used the default position controller that comes with the Hector quadrotor [150] package for controlling the UAV to follow the planned path. Fig. 6.16 shows the planned path (in green) for a mission that consists of eight inspection spots and the actual path (in yellow) that is traversed by the UAV. The closeness between the green and yellow paths shows the navigation accuracy. Table 6.2 shows the effective distance calculated from the planned path and the actual distance traveled by the UAV for different runs, as shown in Fig. 6.16. The actual traveled distance is almost 13.75% extra on average from the planned distance. The main reasons for the extra traveled distances are the erroneous control estimation by the controller and inaccurate maneuvering to the given intermediate poses.

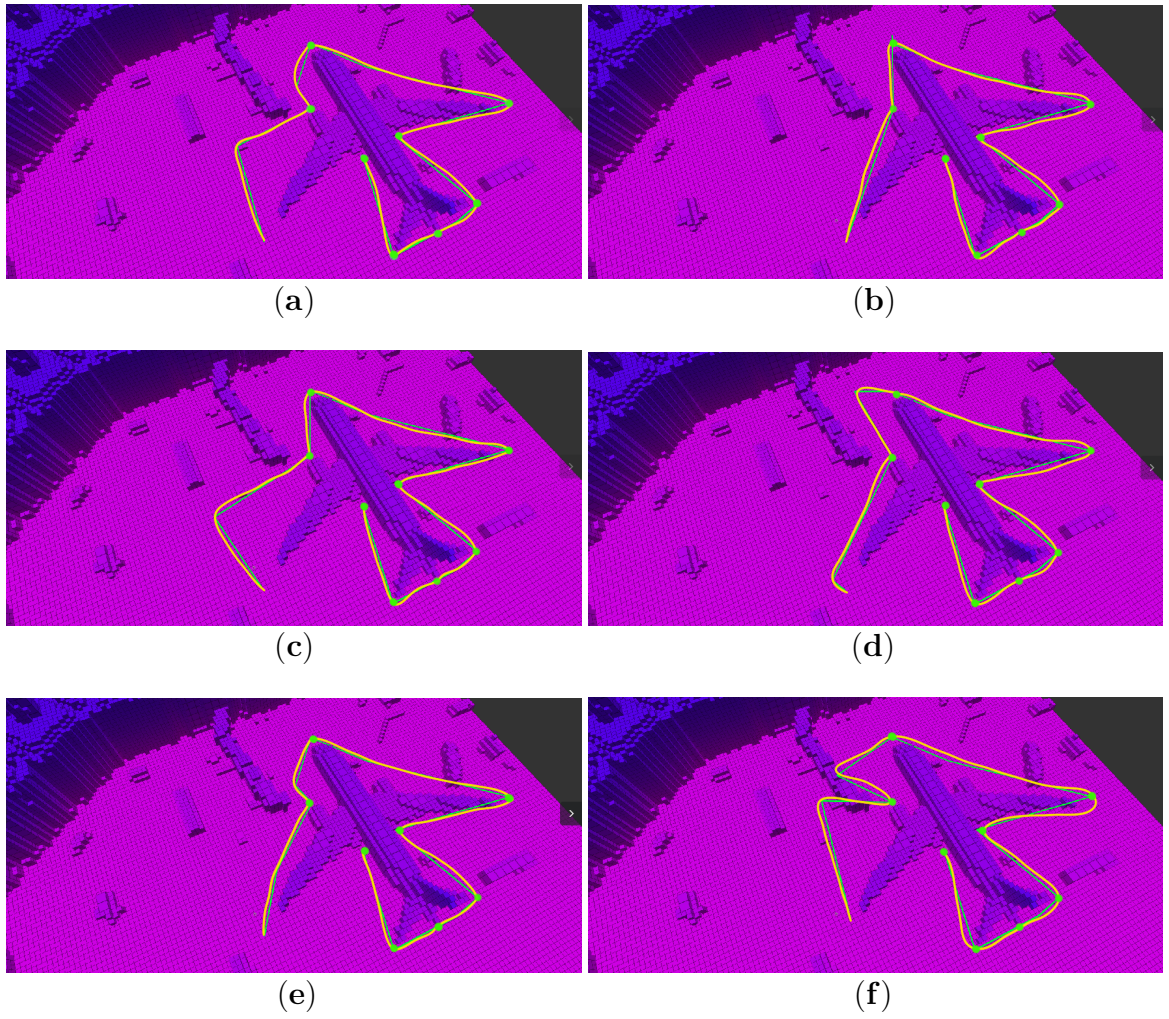


FIGURE 6.16: Obtain planned path (green) and actual path traced by the UAV (yellow) for inspecting various points in different runs.

TABLE 6.2: Effective and Actual Travelled Distances

Run	Effective Distance (m)	Actual Travelled Distance (m)
(a)	210.76	239.35
(b)	201.73	233.64
(c)	211.53	237.08
(d)	208.88	241.29
(e)	202.03	225.85
(f)	227.06	258.19

### 6.4.2 Registration Accuracy

The registration with the coordinate frame  $W$  depends on the accuracy of the template matching procedure for landmark detection. Therefore, a little error in template

matching affects the registration, but the registration error does not have any impact on the inspection process. The registration process helps in initial task distribution, and thereafter, all the UAVs navigate and run the inspection process independently. We evaluate the accuracy of the proposed registration process against the ground truth, where the ground truth is collected from Gazebo. Fig. 6.17 presents a comparison between the relative pose estimation of  $UAV_2$  from  $UAV_1$  in repeated executions, where the maximum error is always below 0.5 meters in all the three axes. The error in

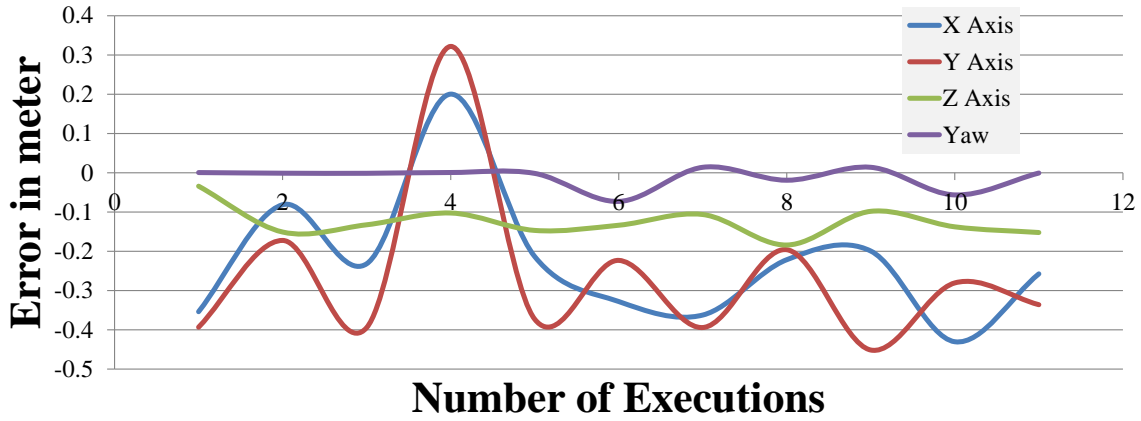


FIGURE 6.17: Error in the relative pose of  $UAV_2$  from  $UAV_1$ . Any value close to zero represents a more accurate estimation.

the Z-axis is the minimum among all the three axes. The orientation of the UAVs is controlled only by the Yaw angle of the Hector quadrotor. Therefore, we provide the comparison only against the yaw parameter, and it is the most accurate parameter in our estimations, as shown in Fig. 6.17.

### 6.4.3 Measurement Validation

We evaluate the measurement accuracy after calculating the object area and the maximum distance as discussed in Section 6.3.7. We performed a repeated execution to measure the area and maximum distances for all three objects and validate them against the ground truth. Fig. 6.18 shows the error plot in the area estimation of an object for multiple executions. The average and maximum errors of area measurement are 0.0004257 square meters and 0.0007349 square meters, respectively. Fig. 6.19 shows the error plot of maximum distance estimation of the object for multiple executions. The average and maximum errors of the maximum distance measurement are 0.001969 meters and 0.004043 meters, respectively. These error values confirm that the measurement accuracy is in the millimeter range.



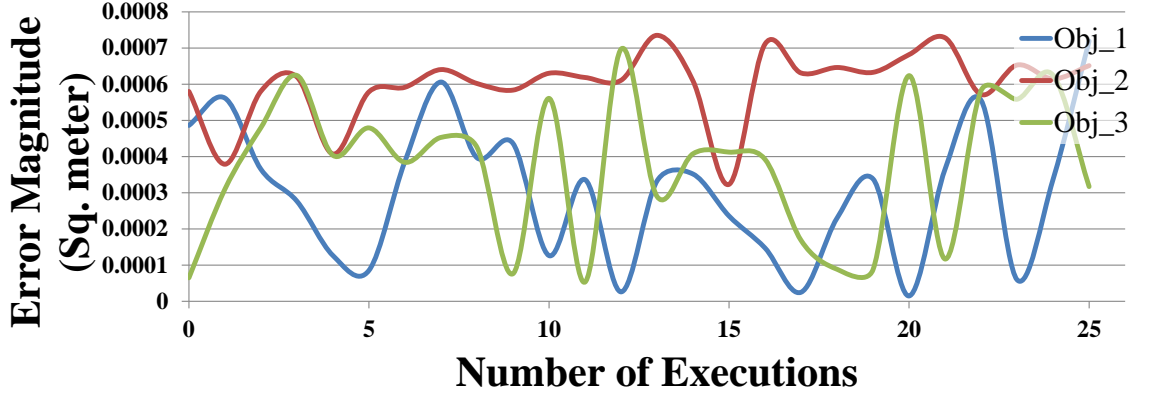


FIGURE 6.18: Error plot for area measurements of all three objects.

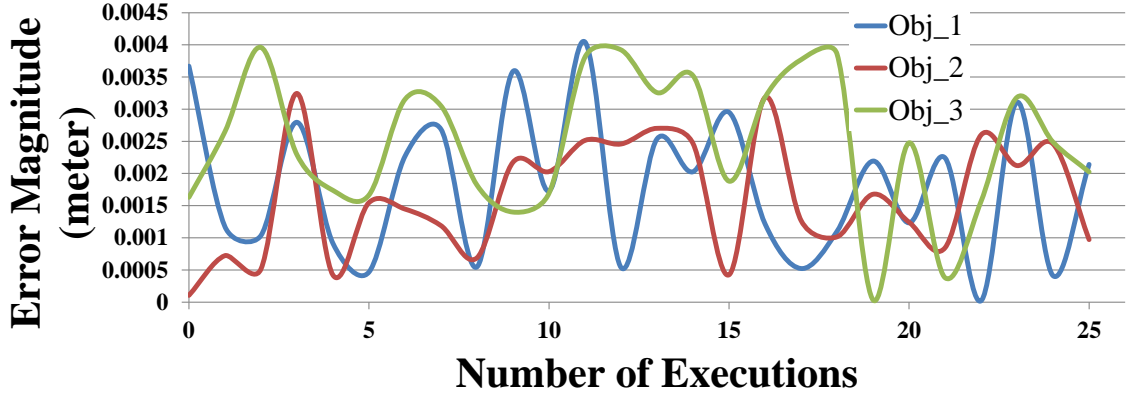


FIGURE 6.19: Error plot for the maximum distances of all three objects.

#### 6.4.4 Execution Time

We compute execution times for each task that is performed from the UAV takeoff to the inspection. Table 6.3 shows the timing details, where the maximum time denotes the maximum execution time of each task that we recorded in our multiple experiments and the average time denotes the simple average of each task in 20 runs. In this experimental setup, two UAVs have participated and there are a total of 8 landmarks to be inspected, and we consider each UAV covers four landmarks and each landmark has four objects on average to be inspected. Therefore, each UAV inspects 16 objects on average. The first three tasks (top three rows) in the Table 6.3 are executed by the UAVs in parallel, and these are one-time executions for the entire mission. The fourth task (fourth row) in the Table 6.3 requires one-time execution for each landmark, which means each UAV spends approximately  $6.12 \times 4 = 24.48$  seconds on average for object detection for four landmarks. The fifth and sixth tasks (fifth and sixth rows) of the Table 6.3 are executed for each object, which means each UAV spends approximately  $5.85 \times 16 = 93.6$  seconds on average for object best view alignment



TABLE 6.3: Execution Timing Details

Sub Tasks	Maximum Time (Seconds)	Average Time (Seconds)
Takeoff	9.10	8.21
Initial Template matching	39.20	33.32
Registration with $F$	1.50	1.08
Object detection (for each landmark)	9.00	6.12
Object best view alignment (for each objects)	8.30	5.85
Object inspection (for each objects)	20.00	11.24
UAV travel time (Covering 140 meters)	132.30	130.00

and approximately  $11.24 \times 16 = 179.84$  seconds on average for object inspection. A UAV covers about 140 meters on average to complete the inspection of 4 landmarks and 16 objects, and it takes approximately 130 seconds on average to execute the desired paths. Therefore, two UAVs take approximately 7 minutes and 51 seconds to complete the entire inspection from the beginning. In our execution, the entire mission takes approximately 8 minutes and 10 seconds, considering message synchronization, network delays, and re-transmission.

## 6.5 Conclusions

We present a novel system for automatic visual inspection of aircraft surfaces using autonomous collaborative UAVs. In this scenario, the UAVs take off from any nearby location of the aircraft and face the RGB-D camera towards the aircraft. All the UAVs find the nearest landmarks using a template matching procedure and register with a single coordinate system. Unlike the existing visual registration method for SLAM, the proposed system registers UAVs with a single coordinate frame, where the UAVs do not watch any common landmark. The registration process helps in task distribution among the participating UAVs. The UAVs navigate using a proposed multilayered zones for safe navigation around the aircraft by avoiding obstacles. The system identifies the objects of interest using a deep-learning-based object detection tool and then performs the inspection. We implement a simple measuring algorithm for simulated objects of interest, where we achieve millimeter range accuracy. The proposed system completes the entire mission within 8.16 minutes using two UAVs.

## Chapter 7

# Conclusions and Future Works

### 7.1 Contributions

This thesis tackles the formidable task of improving environment perception within autonomous mobile robot navigation systems. It delves into the complexities of enabling robots to understand and interact with their surroundings effectively. With a concentrated effort, the thesis aims to delve deeper into dynamic obstacle estimation, focusing on real-time prediction for the avoidance of moving obstacles, while also exploring collaborative localization techniques leveraging visual sensors to precisely determine the robot's position relative to its environment. Through meticulous research and experimentation, this work seeks to advance the capabilities of autonomous robots, ultimately paving the way for safer and more efficient navigation in dynamic environments.

The thesis introduces an obstacle estimation system for dynamic environments using depth images in Chapter 3, facilitating autonomous robotic navigation. The system employs a u-depth map and introduces a restricted v-depth map for accurate estimation of obstacles in terms of position and dimensions. Dynamic binary thresholding is applied to the u-depth map in the proposed system to enhance accuracy of obstacle detection and dimension estimation. An efficient algorithm is utilized for obstacle tracking. The system is designed to identify and track dynamic obstacles that change their sizes and shapes dynamically. Performance evaluation involves testing the proposed system with multiple self-captured and open data sequences. The dynamic obstacle detection and tracking system runs on board at 60 Hz, achieving an average computation time of 0.6 ms per obstacle detection and tracking. Successful tracking is demonstrated at a maximum speed of 5 m/s. However, the system is limited by the speed of obstacles and cannot detect very fast-moving obstacles. The maximum operating depth range of RGB-D sensors is approximately 3 meters, often insufficient for tracking fast-moving obstacles.

The thesis proceeds to introduce the collaborative VI-SLAM framework, CORB2I-SLAM, in Chapter 4, focusing on collaborative localization. The framework enables

each participating robot to carry a visual sensor, such as a monocular, stereo, or RGB-D camera, and an inertial sensor like IMU. Following a centralized architecture, all participating robots are treated as client robots with limited computation capabilities, while a central server possesses higher computation capabilities. The framework empowers client robots to independently run either VIO or VO without relying on the server. The odometry module on client robots is designed to detect noisy inertial sensors and exclude them in pose estimation if necessary. Additionally, the odometry module is crafted to estimate the accuracy of poses and reinitialize with a new sub-map in case of inaccurate tracking. A novel map-merging procedure is proposed between a non-metric scale map and a metric scale map, yielding superior accuracy compared to state-of-the-art techniques. The proposed framework undergoes extensive evaluation on open datasets and in real flight scenarios, demonstrating improved accuracy.

The thesis additionally introduces an enhanced version of the dynamic obstacle estimation system using LiDAR in Chapter 5. This system leverages u-depth and the previously proposed restricted v-depth representations derived from LiDAR point clouds for detecting and estimating obstacles. The utilization of multiple depth representations enables the circumvention of computation-intensive modules, resulting in a faster method compared to state-of-the-art techniques. The proposed system effectively addresses the challenge of estimating long-range obstacles and demonstrates notable improvements across various scenarios, including indoor and outdoor environments, multiple static and moving obstacles, as well as obstacles of diverse shapes and sizes. Evaluation of the proposed system encompasses multiple open datasets, self-captured simulated datasets, and real-time executions with data captured using robotic platforms. The system exhibits a significant enhancement in detecting very thin obstacles, such as slender poles on the road. However, the performance degrades when the self-localization module produces erroneous poses.

The thesis concludes by presenting an end-to-end visual inspection system for pre-flight inspections of aircraft using autonomous collaborative UAVs in Chapter 6. The participating UAVs employ LiDAR-inertial SLAM for self-localization and integrate the proposed obstacle estimation method using LiDAR, as outlined in Chapter 5, to ensure safe navigation by avoiding both static and dynamic obstacles. We introduce a novel registration method to align the participating UAVs with a fixed coordinate frame for collaboration. The proposed registration procedure employs a template matching technique for registration, eliminating the need for UAVs to observe any common landmarks. This registration process facilitates task distribution among the participating UAVs. Ensuring the safety of the multi-million-dollar aircraft involves implementing proposed multilayered zones around the aircraft to guide the UAVs at

restricted speeds. The system identifies objects of interest using a deep-learning-based object detection tool and performs inspections with millimeter-range accuracy. The proposed system successfully completes the entire mission within 8.16 minutes using two UAVs.

## 7.2 Future Research Directions

The CORB2I-SLAM architecture is built upon a client-server model, where all communication data flows through the server system, resulting in a high dependency on the server. In the future, we aim to redesign the framework, aiming for reduced dependency on the server or exploring a distributed framework with minimized data transmission. Recognizing the advantages of LiDAR sensors over cameras, our focus will also include extending the CORB2I-SLAM framework to integrate LiDAR-inertial SLAM on the client.

The inspection system outlined in Chapter 6 assumes that each participating UAV carries a single forward-looking RGB-D camera. However, there are requirements to inspect from different perspectives, such as looking upwards or downwards, for tasks like inspecting the wing from below and from the top, which is not feasible with the current configuration. Therefore, we plan to incorporate upward and downward-looking cameras for a more versatile inspection setup. The current system designates  $UAV_1$  as the mission leader, introducing a dependency on  $UAV_1$  for the entire system. In the future, we aim to explore the development of a sophisticated distributed framework to enhance task efficiency.



## Appendix A

# 3D Reconstruction from 2D Images

### A.1 Pinhole Camera Model

The process of mapping three dimensional (3D) objects onto a two dimensional (2D) plane is commonly referred to as projection. A camera model offers a mapping procedure or function utilized for projecting the 3D world onto a 2D plane, referred to as the image plane. The objective is to design a function that accurately mimics the real world when captured by a physical camera. Various camera models, each with different complexities, are available to capture the perspective of the real world. The perspective effect, where an object appears smaller as it moves farther from the viewer or camera, is a characteristic property inherent in human vision, as the human eye naturally produces such a perspective projection. The simplest form of perspective projection utilizes a pinhole camera model [13]. This projection can be implemented through matrix multiplication when represented in a homogeneous coordinate system. In a homogeneous coordinate system, a vector in  $\mathbb{R}^n$  is represented with  $n + 1$  elements, as opposed to  $n$  elements in standard Euclidean coordinates. The homogeneous representation is scale-invariant. In this appendix, square brackets and parentheses are used to represent vectors in homogeneous coordinates and Euclidean coordinates, respectively.

To comprehend the pinhole camera model, consider placing a camera at a certain distance from a 3D point. The camera comprises a camera center ( $C$ ) and an image plane (Fig. A.1). The 3D camera coordinate frame is denoted by its camera center  $C$  and is aligned with the world coordinate frame. Therefore, the coordinates of  $C$  are  $(0,0,0)^T$ . The other 3D point, toward which the camera is oriented in this coordinate frame  $C$ , is  $M_C = (x,y,z)^T$ . We project the point  $M_C$  onto the image plane by translating it along a straight line toward the camera center, as illustrated in Fig. A.1.

We make the assumption that the image plane is situated parallel to the  $xy$ -plane at  $z = f$ , and the center of the image plane is located at  $(0,0,f)^T$ . Therefore, the  $x$

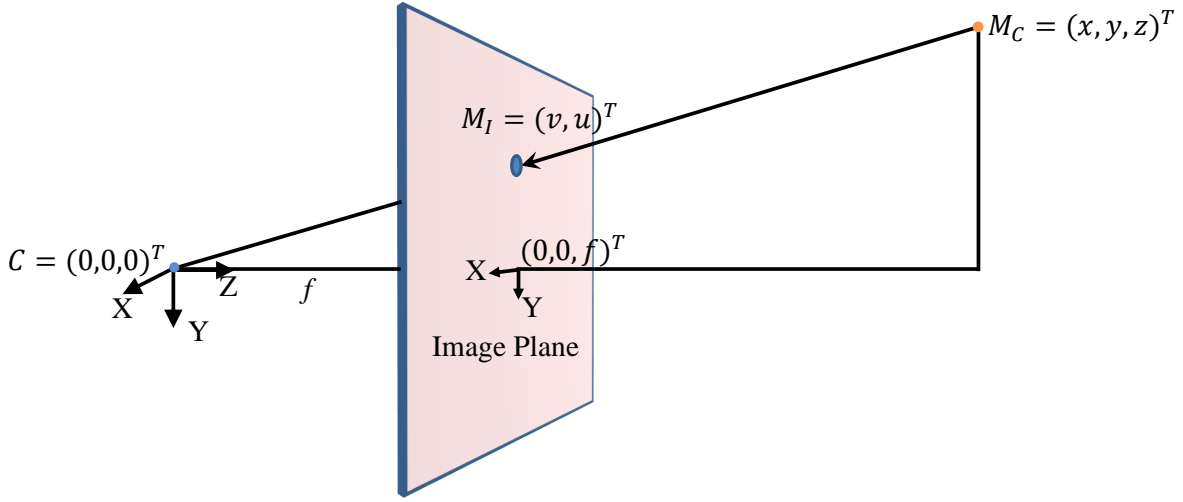


FIGURE A.1: The perspective projection of a 3D point  $M_C$  onto an image plane  $M_I$  using the pinhole camera model.

and y axes of the 2D image coordinate are parallel to the x and y axes of the 3D camera coordinate frame. The projection of the 3D point  $M_C$  onto the image plane results in a 2D image point  $M_I = (v, u)^T$  (Fig. A.1). Using similar triangles, we can calculate the coordinates of the 2D point  $M_I$ , as demonstrated in Equ. (A.1).

$$\begin{aligned} u &= \frac{f \times x}{z} \\ v &= \frac{f \times y}{z} \end{aligned} \tag{A.1}$$

This perspective projection can be defined by expression (A.2), where the sign ‘ $\rightarrow$ ’ denotes this operation.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow f \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \\ 1 \end{pmatrix} \tag{A.2}$$

This perspective projection can further be re-written with matrix multiplication as in expression (A.3).



$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (\text{A.3})$$

In homogeneous coordinates the expression (A.3) can be presented as shown in expression (A.4).

$$\begin{bmatrix} x \\ y \\ z \\ \alpha \end{bmatrix} \rightarrow \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ \alpha \end{bmatrix} \quad (\text{A.4})$$

where,  $\alpha$  represents any non-zero constant used for the transformation from Euclidean to homogeneous coordinates. In the given configuration, our assumption was that the 3D camera coordinate frame is aligned with the world coordinate frame. However, in reality, the camera can be positioned at different locations while looking in various directions. This implies that we can have multiple cameras with arbitrary poses involving rotations and translations. Each camera possesses its own 3D camera coordinate frame and 2D image coordinate frame, while a single world coordinate frame  $W$  is shared among them. The scenario is depicted in Fig. A.2, where the world and camera coordinate frames are represented with origins  $O$  (axes are  $X_W, Y_W, Z_W$ ) and  $C$  (axes are  $X_C, Y_C, Z_C$ ), respectively. Therefore, if the homogeneous form of a 3D point in the world coordinate frame and camera coordinate frame is denoted as  $M_W$  and  $M_C$ , respectively, then the relationship between them is illustrated in Equ. (A.5).

$$M_C = \left( \begin{array}{ccc|c} \mathbf{R} & & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{array} \right) M_W \quad (\text{A.5})$$

where,  $\mathbf{R}_{3 \times 3}$  represents the camera rotation matrix, and  $\mathbf{t}_{3 \times 1}$  denotes the camera translation vector. The projection function, which maps the projection of a 3D point in the world coordinate frame to the 2D image coordinate frame, can be expressed as shown in expression (A.6). Here, we consider that the 3D point  $(x, y, z)^T$  is now represented in the world coordinate frame  $W$ .

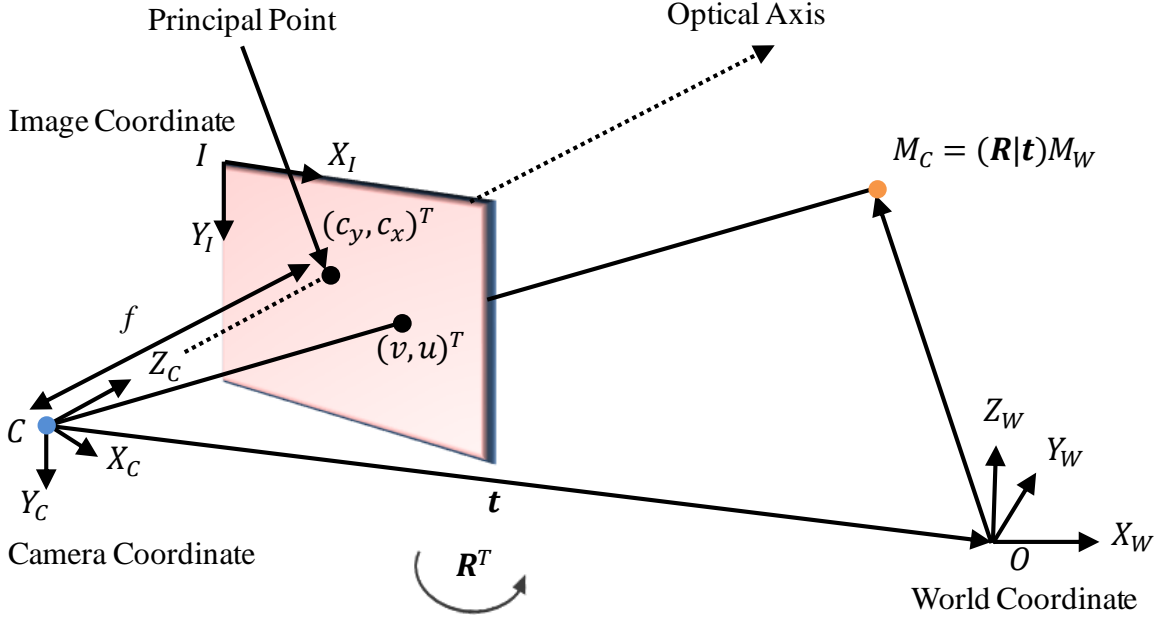


FIGURE A.2: Perspective projection incorporating the world, camera and image coordinates using pinhole camera model.

$$\begin{bmatrix} x \\ y \\ z \\ \alpha \end{bmatrix} \rightarrow \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \left( \begin{array}{ccc|c} \mathbf{R} & & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{array} \right) \begin{bmatrix} x \\ y \\ z \\ \alpha \end{bmatrix} \quad (\text{A.6})$$

The projection component of expression (A.6) can be further streamlined, as depicted in Equ. (A.7), by eliminating extra rows and columns. This consolidated projection represents the projection matrix  $\mathbf{P}$ .

$$\mathbf{P} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} (\mathbf{R}|\mathbf{t}) \quad (\text{A.7})$$

The projection operation can also be expressed as a functional operation, where  $\varphi(\cdot)$  is regarded as the projection function acting on 3D world points, as illustrated in Equ. (A.8).

$$\varphi(M_W) = \mathbf{P}M_W \quad (\text{A.8})$$

In our above formulation, we made some simplified assumptions that can be refined to better reflect reality. Initially, we assumed the two axes of the 2D image plane were identical, but in actuality, digital cameras have non-square pixel elements. This discrepancy can be addressed in the pinhole camera model by employing different focal lengths, denoted as  $f_x$  and  $f_y$  for the x and y axes, respectively. Additionally, we might consider positioning the origin of the image plane elsewhere rather than at the intersection of the Z-axis and the image plane. Fig. A.2 depicts this scenario, where the origin of the 2D image coordinate is at the top-left corner of the image plane and denoted by  $I$  (axes are  $X_I, Y_I$ ). Furthermore, various types of lens distortions exist, but for simplicity, we exclude these distortions from the scope and refer to [13] for more details. To accommodate the first two conditions, a simple translation of the image plane and the utilization of  $f_x$  and  $f_y$  in the projection matrix, as illustrated in Equ. (A.9), are sufficient.

$$\begin{aligned}
 \mathbf{P} &= \begin{pmatrix} 1 & 0 & c_x \\ 0 & 1 & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix} (\mathbf{R}|\mathbf{t}) \\
 &= \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} (\mathbf{R}|\mathbf{t}) \\
 &= \mathbf{K} (\mathbf{R}|\mathbf{t})
 \end{aligned} \tag{A.9}$$

where,  $(c_y, c_x)^T$  is the translation applied to the image plane after the perspective projection. This point  $(c_y, c_x)$  is commonly referred to as the principal point, as illustrated in Fig. A.2. The internal or intrinsic calibration matrix is denoted as  $\mathbf{K}$ . The projection matrix in Equ. (A.9) can be further utilized through matrix multiplication for perspective projection, as formulated in Equ. (A.10).

$$\begin{aligned}
 M_I &= \mathbf{K} M_C \\
 M_I &= \mathbf{K} [\mathbf{R}|\mathbf{t}] M_W
 \end{aligned} \tag{A.10}$$

where,  $M_W$  represents the 3D point in the world coordinate frame, and  $M_I$  is the corresponding 2D image point in homogeneous form. When employing the current form of the intrinsic camera matrix  $\mathbf{K}$ , Equ. (A.1) can be revised to Equ. (A.11) to project a 3D point in the camera coordinate frame onto the image plane.

$$\begin{aligned} u &= \frac{f_x \times x}{z} + c_x \\ v &= \frac{f_y \times y}{z} + c_y \end{aligned} \tag{A.11}$$

We recommend referring to [13] for an in-depth exploration of diverse camera models and the projections derived from them.

## A.2 Structure From Motion

Structure from motion (SfM) [6] is a well-known concept in the computer vision domain, addressing the challenge of reconstructing the 3D structure of a scene along with corresponding camera poses from an unordered set of images (calibrated or non-calibrated). True to its name, SfM constructs a rigid 3D structure from a collection of images taken from different viewpoints. When viewed as a pipeline, SfM comprises a sequence of algorithms that operate sequentially in a defined manner. These algorithms include:

- Feature Estimation and Matching
- Epipolar Geometry and Essential Matrix Estimation
- Fundamental Matrix Estimation
- Camera Pose Estimation
- Optimization

The estimated 3D structure and camera poses are typically refined periodically using an offline optimization process called bundle adjustment (BA) [110], where the computation time scales with the number of input images. For a comprehensive survey on SfM, we refer to [160].

### A.2.1 Feature Estimation and Matching

Image features are defined as specific pixels within an image that possess distinctive characteristics, enabling easy identification across different images. Researchers employ various algorithms such as Scale Invariant Feature Transform (SIFT) [161], Speeded Up Robust Features (SURF) [162], Oriented FAST and Rotated BRIEF (ORB) [149], etc., to detect these features and subsequently match them across multiple images utilizing

their corresponding feature descriptors. In Fig. A.3, matched ORB features between two images captured from different viewpoints are illustrated. The connected green lines represent the point correspondences of these matched feature points, with only four sampled point correspondences displayed for enhanced visibility.

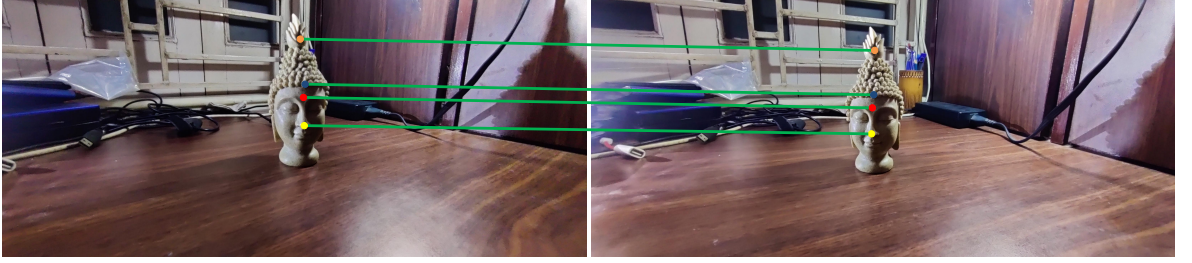


FIGURE A.3: An example of matched ORB feature points between two images captured from two different view points.

### A.2.2 Epipolar Geometry and Essential Matrix Estimation

The fundamental matrix serves as a representation of the relationships among all point correspondences between two images captured from distinct viewpoints [13]. This matrix, denoted as  $\mathbf{F}_{3 \times 3}$ , is of rank two, constituting an algebraic depiction of the connection between the two images. In this section, we briefly delve into the epipolar constraint, followed by the subsequent estimation of the fundamental matrix in the next section. Consider Fig. A.4, where a point  $M_W$  exists in 3D space and is visible in two different camera views. The left view captures  $M_W$  as  $M_l$ , while the right view captures the same point as  $M_r$ . Let  $C_l$  and  $C_r$  represent the left and right camera centers, respectively, forming a baseline between the two camera views. Evidently, the points  $M_W$ ,  $M_l$ ,  $C_l$ ,  $M_r$ ,  $C_r$  lie on a planar surface referred to as the epipolar plane, denoted by  $\pi$ . This coplanarity can be expressed using three coplanar vectors, as shown in Equ. (A.12).

$$\overrightarrow{C_l M_l} \cdot (\overrightarrow{C_l C_r} \times \overrightarrow{C_r M_r}) = 0 \quad (\text{A.12})$$

Now, let us consider Fig. A.5 and assume that the location of  $M_W$  is unknown. The feature detector provides  $M_l$  as a feature point on the left image, and our goal is to determine its corresponding point,  $M_r$ , in the right image. Knowing that  $M_l$  lies on the epipolar plane  $\pi$ , it is governed by the baseline vector  $\overrightarrow{C_l C_r}$  and the vector  $\overrightarrow{C_r M_r}$  to establish the corresponding point in the right image. Consequently,  $M_r$  must lie on the line of intersection between the two planes: the epipolar plane  $\pi$  and the second

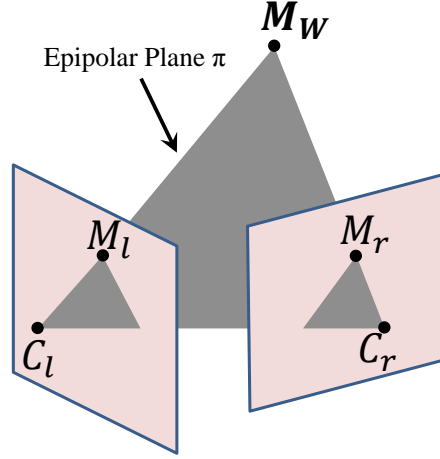


FIGURE A.4: Perspective projection of a 3D point  $M_W$  in two different views as  $M_l$  and  $M_r$ .

image plane. This intersection line is denoted as  $l_r$ . This line,  $l_r$ , is the projection of the ray formed with the points  $M_l$  and  $C_l$  onto the right image plane and is referred to as the epipolar line corresponding to the point  $M_l$ . Each feature point on the left image has its individual epipolar line on the right image. The advantage lies in the fact that the corresponding point of  $M_l$  must lie on the corresponding epipolar line  $l_r$  on the right image, allowing us to focus solely on the line  $l_r$  instead of the entire image plane. Consequently, we can examine only the feature points from the right image that lie on the line  $l_r$  and select the best match as  $M_r$ . The projection of the left camera center  $C_l$  on the right image plane is termed the epipole and denoted as  $e_r$ , as depicted in Fig. A.5. Similarly, we can obtain the epipole  $e_l$  and epipolar line  $l_l$  on the left image for the point  $M_r$ . All epipolar lines formed on the right image plane for each feature point on the left image must pass through the epipole  $e_r$ , and vice versa.

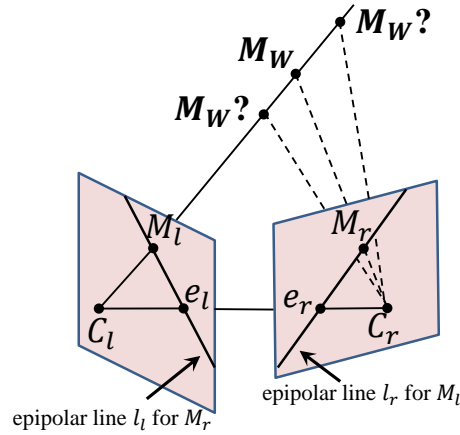


FIGURE A.5: Epipolar geometry.

Now, let us assume that the cameras for both the left and right images are calibrated, meaning we already know the internal calibration matrices for both cameras. Let  $\mathbf{K}_l$  and  $\mathbf{K}_r$  be the internal calibration matrices for the left and right cameras, respectively. The corresponding normalized homogeneous coordinates of  $M_l$  and  $M_r$  are denoted as  $\bar{M}_l$  and  $\bar{M}_r$ , and their relations are illustrated in Equ. (A.13) [13].

$$\begin{aligned}\bar{M}_l &= \mathbf{K}_l^{-1} M_l \\ \bar{M}_r &= \mathbf{K}_r^{-1} M_r\end{aligned}\tag{A.13}$$

Now, if we express the coplanarity relation of three vectors, as presented in Equ. (A.12), in these normalized coordinates, the constraints can be formulated as shown in Equ. (A.14).

$$(\bar{M}_r^T) \cdot [\mathbf{t} \times (\mathbf{R} \bar{M}_l)] = 0\tag{A.14}$$

where,  $\mathbf{R}$  and  $\mathbf{t}$  represent the rotation matrix and translation vector, respectively, transforming the left camera coordinate frame to the right camera coordinate frame. The multiplication by  $\mathbf{R}$  rotates the point  $\bar{M}_l$  into the right camera coordinate frame. Let us define  $[\mathbf{t}_x]$  as a matrix such that  $[\mathbf{t}_x] \mathbf{v} = \mathbf{t} \times \mathbf{v}$  for any vector  $\mathbf{v}$ . Then, Equ. (A.14) can be reexpressed as shown in Equ. (A.15).

$$\begin{aligned}(\bar{M}_r^T) ([\mathbf{t}_x] \mathbf{R} \bar{M}_l) &= 0 \\ \Rightarrow (\bar{M}_r^T) \mathbf{E} \bar{M}_l &= 0\end{aligned}\tag{A.15}$$

where,  $\mathbf{E}_{3 \times 3} = [\mathbf{t}_x] \mathbf{R}$  is called the Essential matrix [13].

### A.2.3 Fundamental Matrix Estimation

Our earlier assumption was that both cameras were calibrated. However, it is possible that the cameras are uncalibrated, meaning the internal calibrations  $\mathbf{K}_l$  and  $\mathbf{K}_r$  are not known a priori. In such cases, we must operate with the homogeneous image coordinate system rather than the normalized homogeneous coordinate system. In this scenario, we utilize Equ. (A.13) and Equ. (A.15) to establish a relation in the homogeneous image coordinate system, as demonstrated in Equ. (A.16).



$$\begin{aligned}
(\mathbf{K}_r^{-1}M_r)^T([\mathbf{t}_x]\mathbf{R}(\mathbf{K}_l^{-1}M_l)) &= 0 \\
\Rightarrow M_r^T\mathbf{K}_r^{-T}([\mathbf{t}_x]\mathbf{R}\mathbf{K}_l^{-1}M_l) &= 0 \\
\Rightarrow M_r^T\mathbf{K}_r^{-T}([\mathbf{t}_x]\mathbf{R})\mathbf{K}_l^{-1}M_l &= 0 \\
\Rightarrow M_r^T(\mathbf{K}_r^{-T}\mathbf{E}\mathbf{K}_l^{-1})M_l &= 0 \\
\Rightarrow M_r^T\mathbf{F}M_l &= 0
\end{aligned} \tag{A.16}$$

where,  $\mathbf{F}_{3 \times 3} = \mathbf{K}_r^{-T}\mathbf{E}\mathbf{K}_l^{-1}$  represents the Fundamental matrix. Consequently, both the Essential and Fundamental matrices are employed to elucidate the geometric relationship among corresponding points in a pair of cameras. The fundamental disparity between these matrices lies in the fact that the Essential matrix operates within a normalized homogeneous coordinate frame, specifically designed for calibrated cameras. On the other hand, the Fundamental matrix is applicable in a homogeneous image coordinate frame, catering to uncalibrated cameras. The Essential matrix possesses five degrees of freedom, while the Fundamental matrix has seven degrees of freedom. It's noteworthy that both matrices maintain a rank of two [13].

Now, typically, we acquire a set of point correspondences between two images, denoted by  $n$ . All  $n$  correspondences must adhere to the constraint of the Fundamental matrix, namely  $M_r^T\mathbf{F}M_l = 0$ . We can construct a homogeneous linear system incorporating all these point correspondences, as illustrated in Equ. (A.17).

$$\begin{bmatrix} M_{r_{xi}} & M_{r_{yi}} & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} M_{l_{xi}} \\ M_{l_{yi}} \\ 1 \end{bmatrix} = 0 \tag{A.17}$$

where,  $i = 1, 2, \dots, n$  and this can be formed as in Equ. (A.18) and further simplify as in Equ. (A.19).

$$\begin{aligned}
M_{l_{xi}}M_{r_{xi}}f_{11} + M_{l_{xi}}M_{r_{yi}}f_{21} + M_{l_{xi}}f_{31} + M_{l_{yi}}M_{r_{xi}}f_{12} + M_{l_{yi}}M_{r_{yi}}f_{22} \\
+ M_{l_{yi}}f_{32} + M_{r_{xi}}f_{13} + M_{r_{yi}}f_{23} + f_{33} = 0
\end{aligned} \tag{A.18}$$

$$\begin{bmatrix}
M_{l_{x1}}M_{r_{x1}} & M_{l_{x1}}M_{r_{y1}} & M_{l_{x1}} & M_{l_{y1}}M_{r_{x1}} & M_{l_{y1}}M_{r_{y1}} & M_{l_{y1}} & M_{r_{x1}} & M_{r_{y1}} & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
M_{l_{xn}}M_{r_{xn}} & M_{l_{xn}}M_{r_{yn}} & M_{l_{xn}} & M_{l_{yn}}M_{r_{xn}} & M_{l_{yn}}M_{r_{yn}} & M_{l_{yn}} & M_{r_{xn}} & M_{r_{yn}} & 1
\end{bmatrix}
\begin{bmatrix}
f_{11} \\
f_{21} \\
f_{31} \\
f_{12} \\
f_{22} \\
f_{32} \\
f_{13} \\
f_{23} \\
f_{33}
\end{bmatrix} = 0 \tag{A.19}$$

We can consider Equ. (A.19) as the standard linear system of equations in the form  $Ax = 0$ , and it can be solved using Singular Value Decomposition (SVD) [163]. Typically, feature correspondences contain noise and numerous outliers. Hence, we employ the Random Sample Consensus (RANSAC) [164] method to achieve a more accurate estimate of the fundamental matrix by discarding noisy point correspondences. The internal calibration matrix is usually estimated offline through a standard camera calibration procedure [13] for a given camera. In the case of an unknown camera, we initialize the values and refine the matrix subsequently using optimization techniques described in Section A.2.5. Following this, we utilize the camera calibration matrices and the Fundamental matrix to obtain the Essential matrix, as derived in Equ. (A.20).

$$\begin{aligned}
\mathbf{F} &= \mathbf{K}_r^{-T} \mathbf{E} \mathbf{K}_l^{-1} \\
\Rightarrow \mathbf{K}_r^T \mathbf{F} \mathbf{K}_l &= \mathbf{K}_r^T \mathbf{K}_r^{-T} \mathbf{E} \mathbf{K}_l^{-1} \mathbf{K}_l \\
\Rightarrow \mathbf{K}_r^T \mathbf{F} \mathbf{K}_l &= (\mathbf{K}_r^T \mathbf{K}_r^{-T}) \mathbf{E} (\mathbf{K}_l^{-1} \mathbf{K}_l) \\
\Rightarrow \mathbf{K}_r^T \mathbf{F} \mathbf{K}_l &= \mathbf{E} \\
\Rightarrow \mathbf{E} &= \mathbf{K}_r^T \mathbf{F} \mathbf{K}_l
\end{aligned} \tag{A.20}$$

Hence, the Essential matrix  $\mathbf{E}$  is a homogeneous matrix and is solely constrained by  $(\bar{\mathbf{M}}_r^T) \mathbf{E} \bar{\mathbf{M}}_l = 0$ , as demonstrated in Equ. (A.15). However, as we have previously observed, we can also construct the matrix in a nonhomogeneous form using the relation  $\mathbf{E} = [\mathbf{t}_x] \mathbf{R}$ .

### A.2.4 Camera Pose Estimation

H. C. Longuet-Higgins [165] demonstrated that the relative pose, i.e.,  $\mathbf{R}$  and  $\mathbf{t}$ , can be recovered from the Essential matrix up to the scale of  $\mathbf{t}$ . This is because the projection rule, as depicted in Equ. (A.10), remains valid when the coordinates of the 3D world point increase by any factor, such as  $\lambda$ , while the translation vector is simultaneously reduced by the same factor  $\lambda$ , as illustrated in Equ. (A.21).

$$M_I = \mathbf{K}[\mathbf{R} | (\frac{1}{\lambda}\mathbf{t})](\lambda M_W) \quad (\text{A.21})$$

Mathematically, four poses can be calculated from a given Essential matrix. However, the correct pose is the one that produces scene points in the frontal side of both cameras, adhering to the cheirality constraint [13]. Therefore, the four theoretical solutions are obtained by decomposing the Essential matrix  $\mathbf{E}$  using SVD and subsequently verifying the cheirality constraint after creating 3D points through triangulation [166], [167]. The 3D points corresponding to the correct camera pose establish the 3D scene structure.

If we assume that the left camera is aligned with the world coordinate frame, then the rotation and translation of the left camera become  $\mathbf{R} = [\mathbf{I}]$  and  $\mathbf{t} = 0$ , respectively, while the rotation and translation of the right camera become  $\mathbf{R}$  and  $\mathbf{t}$ , respectively. Consequently, we obtain two fully calibrated cameras (internal calibration matrices and poses) and their corresponding 3D scene structure. Any new camera must have a 3D-2D relation with the reconstructed 3D scene structure. Therefore, the pose of the new images can be estimated using linear least squares [112]. This problem is known as the Perspective- $n$ -Point (PnP) problem [168], which can also be solved using the RANSAC [164] method.

### A.2.5 Optimization Using Bundle Adjustment

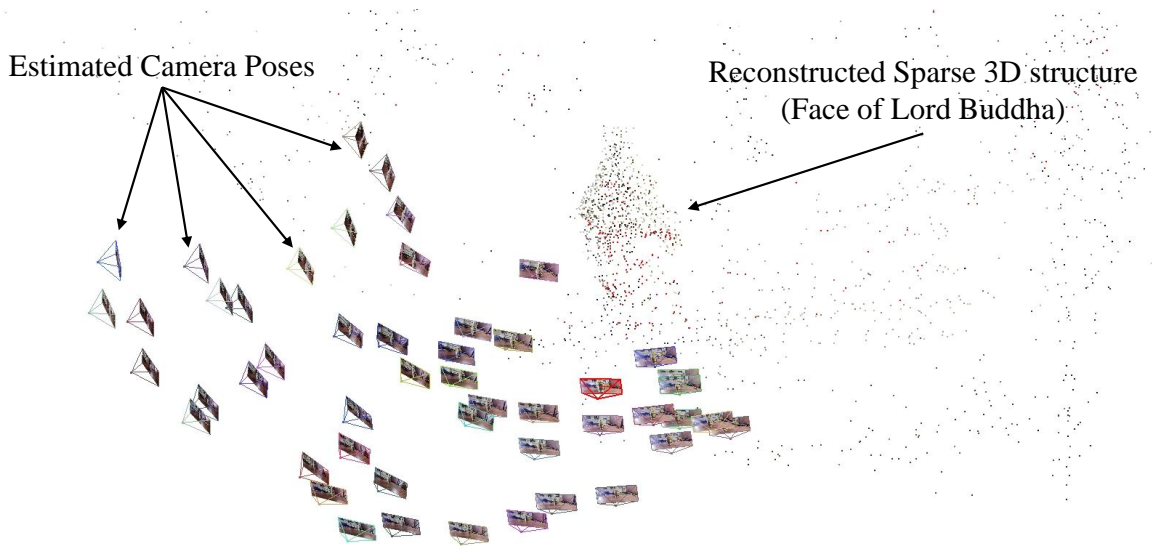
The estimated camera poses and the reconstructed 3D structure are typically noisy due to the noise in the point correspondences. Consequently, all the camera poses and the 3D structure undergo further global refinement through an optimization process. The optimization algorithm formulates the cost as the summation of the reprojection errors [13] of every 3D point in its possible visibility on the corresponding image planes, a technique known as the Levenberg-Marquardt algorithm [169]. This optimization process is commonly referred to as Bundle Adjustment (BA) [110], and the cost function is formulated in Equ. (A.22).

$$\underset{\{M_{W_i}\}, \{\varphi_j\}}{\text{minimize}} \sum_{i=1}^n \sum_{j=1}^m V_{ij} D(M_{I_{ij}}, \varphi_j(M_{W_i}))^2 \quad (\text{A.22})$$

where,  $n$  denotes the number of points, and  $m$  denotes the number of cameras.  $\varphi_j(M_{W_i})$  implies the predicted projection of the  $i^{\text{th}}$  3D point,  $M_{W_i}$ , onto the  $j^{\text{th}}$  camera, and  $M_{I_{ij}} \in \mathbb{R}^2$  is the corresponding true projection.  $D(p, q)$  denotes the Euclidean distance between the 2D image points  $p$  and  $q$ , and  $V_{ij}$  is a binary variable that attains a value of 1 when point  $i$  is visible in camera  $j$ , and 0 otherwise.



(a)



(b)

FIGURE A.6: An example of sparse 3D reconstruction using SfM pipeline: (a) Sample images that are used for the reconstruction, (b) Sparse reconstructed structure and estimated camera poses.

Consider Fig. A.6(a), where multiple images are captured from different viewpoints of a rigid structure (specifically, a statue of the face of the Lord Buddha). The sparse

3D structure is reconstructed from these 2D images using a widely used SfM pipeline, as proposed in [170], with BA implemented in a distributed manner across multiple cores [171]. The optimized reconstructed sparse 3D structure is depicted in Fig. A.6(b).

We can infer from Fig. A.6(b) that the sparse reconstruction may not convey the intended structural meaning. Therefore, a dense reconstruction may provide a clearer structural interpretation, as illustrated in Fig. A.7. This denser reconstruction is achieved using a method proposed in [172], [173].

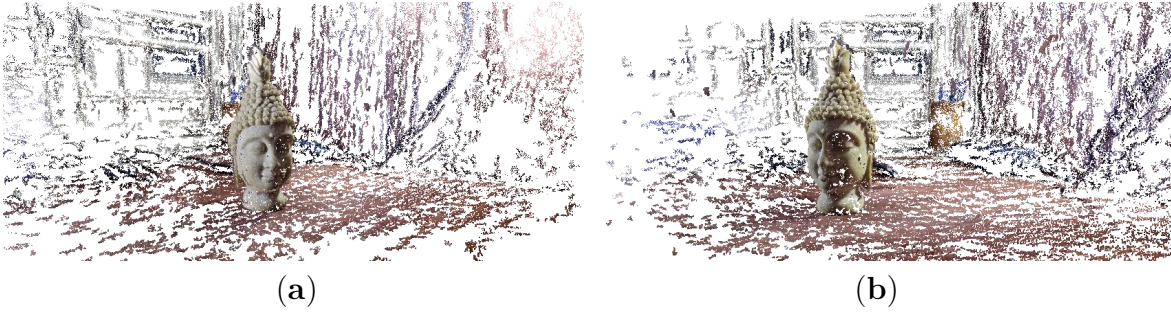


FIGURE A.7: A dense reconstruction using PMVS2/CMVS pipeline [172], [173]: (a)–(b) Multiple views.

# Bibliography

- [1] P. Wang, T. Gu, B. Sun, D. Huang, and K. Sun, “Research on 3D Point Cloud Data Preprocessing and Clustering Algorithm of Obstacles for Intelligent Vehicle”, *World Electric Vehicle Journal*, vol. 13, no. 7, 2022, ISSN: 2032-6653. DOI: [10.3390/wevj13070130](https://doi.org/10.3390/wevj13070130).
- [2] A. Saha, B. C. Dhara, S. Umer, K. Yuri, J. M. Alanazi, and A. A. AlZubi, “Efficient Obstacle Detection and Tracking Using RGB-D Sensor Data in Dynamic Environments for Robotic Applications”, *Sensors*, vol. 22, no. 17, 2022, ISSN: 1424-8220. DOI: [10.3390/s22176537](https://doi.org/10.3390/s22176537). [Online]. Available: <https://www.mdpi.com/1424-8220/22/17/6537>.
- [3] C.-C. Lin, W.-L. Mao, T. W. Chang, C.-Y. Chang, and S. S. S. Abdullah, “Fast Obstacle Detection Using 3D-to-2D LiDAR Point Cloud Segmentation for Collision-free Path Planning”, *Sensors and Materials*, vol. 32, no. 7, pp. 2365–2374, 2020. DOI: [10.18494/SAM.2020.2810](https://doi.org/10.18494/SAM.2020.2810). [Online]. Available: <https://doi.org/10.18494/SAM.2020.2810>.
- [4] J. Lin, H. Zhu, and J. Alonso-Mora, “Robust Vision-based Obstacle Avoidance for Micro Aerial Vehicles in Dynamic Environments”, in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 2682–2688. DOI: [10.1109/ICRA40945.2020.9197481](https://doi.org/10.1109/ICRA40945.2020.9197481).
- [5] P. Turaga, R. Chellappa, and A. Srivastava, “Chapter 7 - Statistical Methods on Special Manifolds for Image and Video Understanding”, in *Handbook of Statistics*, ser. Handbook of Statistics, C. Rao and V. Govindaraju, Eds., vol. 31, Elsevier, 2013, pp. 178–201. DOI: <https://doi.org/10.1016/B978-0-444-53859-8.00007-2>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780444538598000072>.
- [6] S. Ullman, “The Interpretation of Structure from Motion”, in *Proceedings of the Royal Society of London*, 1153. 1979, vol. 203, pp. 405–426. DOI: <https://doi.org/10.1098/rspb.1979.0006>.
- [7] S. Song and J. Xiao, “Tracking Revisited Using RGBD Camera: Unified Benchmark and Baselines”, in *IEEE International Conference on Computer Vision*, 2013, pp. 233–240. DOI: [10.1109/ICCV.2013.36](https://doi.org/10.1109/ICCV.2013.36).



- [8] A. K. Sadhu, S. Shukla, T. Bera, and R. Dasgupta, “Safe and Fast Path Planning in Cluttered Environment using Contiguous Free-space Partitioning”, in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6972–6978. DOI: [10.1109/ICRA.2019.8793921](https://doi.org/10.1109/ICRA.2019.8793921).
- [9] A. K. Sadhu, M. Ludhiyani, D. Chakraborty, T. Bera, A. Sinha, and R. Dasgupta, “Autonomous Robot Navigation in Cluttered Environment”, *IET Conference Proceedings*, 28–34(6), Oct. 2020. [Online]. Available: <https://digital-library.theiet.org/content/conferences/10.1049/icp.2021.1140>.
- [10] D. Scaramuzza and F. Fraundorfer, “Visual Odometry [Tutorial]”, *IEEE Robotics and Automation Magazine*, vol. 18, no. 4, pp. 80–92, Dec. 2011. DOI: [10.1109/MRA.2011.943233](https://doi.org/10.1109/MRA.2011.943233).
- [11] F. Fraundorfer and D. Scaramuzza, “Visual Odometry: Part II - Matching, Robustness, and Applications”, *IEEE Robotics and Automation Magazine*, vol. 19, pp. 78–90, Jun. 2012. DOI: [10.1109/MRA.2012.2182810](https://doi.org/10.1109/MRA.2012.2182810).
- [12] Y. Alkendi, L. Seneviratne, and Y. Zweiri, “State of the Art in Vision-Based Localization Techniques for Autonomous Navigation Systems”, *IEEE Access*, vol. 9, pp. 76 847–76 874, 2021. DOI: [10.1109/ACCESS.2021.3082778](https://doi.org/10.1109/ACCESS.2021.3082778).
- [13] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Second. Cambridge University Press, ISBN: 0521540518, 2004.
- [14] R. Szeliski, *Computer vision algorithms and applications*. London; New York, 2011. [Online]. Available: <http://link.springer.com/book/10.1007/978-1-84882-935-0>.
- [15] R. Mur-Artal and J. Tardós, “ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”, *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [16] S. Maity, A. Saha, and B. Bhowmick, “Edge SLAM: Edge Points Based Monocular Visual SLAM”, in *IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 2408–2417. DOI: [10.1109/ICCVW.2017.284](https://doi.org/10.1109/ICCVW.2017.284).
- [17] S. Yang and S. Scherer, “Direct monocular odometry using points and lines”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3871–3877. DOI: [10.1109/ICRA.2017.7989446](https://doi.org/10.1109/ICRA.2017.7989446).



- [18] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard, “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”, *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.
- [19] W. Chen, G. Shang, A. Ji, C. Zhou, X. Wang, C. Xu, Z. Li, and K. Hu, “An Overview on Visual SLAM: From Tradition to Semantic”, *Remote Sensing*, vol. 14, no. 13, 2022, ISSN: 2072-4292. DOI: [10.3390/rs14133010](https://doi.org/10.3390/rs14133010). [Online]. Available: <https://www.mdpi.com/2072-4292/14/13/3010>.
- [20] A. Macario Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, “A Comprehensive Survey of Visual SLAM Algorithms”, *Robotics*, vol. 11, no. 1, 2022, ISSN: 2218-6581. DOI: [10.3390/robotics11010024](https://doi.org/10.3390/robotics11010024). [Online]. Available: <https://www.mdpi.com/2218-6581/11/1/24>.
- [21] P. Schmuck and M. Chli, “CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams”, *Journal of Field Robotics*, vol. 36, no. 4, pp. 763–781, 2018.
- [22] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, “Keyframe-based visual-inertial SLAM using nonlinear optimization”, *International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.
- [23] Y. Yang, P. Geneva, K. Eickenhoff, and G. Huang, “Visual-Inertial Odometry with Point and Line Features”, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 2447–2454. DOI: [10.1109/IROS40897.2019.8967905](https://doi.org/10.1109/IROS40897.2019.8967905).
- [24] D. Scaramuzza and Z. Zhang, “Aerial Robots, Visual-Inertial Odometry of”, in *Encyclopedia of Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–9, ISBN: 978-3-642-41610-1. DOI: [10.1007/978-3-642-41610-1\\_71-1](https://doi.org/10.1007/978-3-642-41610-1_71-1). [Online]. Available: [https://doi.org/10.1007/978-3-642-41610-1\\_71-1](https://doi.org/10.1007/978-3-642-41610-1_71-1).
- [25] H. Li and J. Stueckler, “Visual-Inertial Odometry With Online Calibration of Velocity-Control Based Kinematic Motion Models”, *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6415–6422, 2022. DOI: [10.1109/LRA.2022.3169837](https://doi.org/10.1109/LRA.2022.3169837).
- [26] M. Karrer, P. Schmuck, and M. Chli, “CVI-SLAM—Collaborative Visual-Inertial SLAM”, *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2762–2769, 2018. DOI: [10.1109/LRA.2018.2837226](https://doi.org/10.1109/LRA.2018.2837226).

- [27] H. Zhang, X. Chen, H. Lu, and J. Xiao, “Distributed and collaborative monocular simultaneous localization and mapping for multi-robot systems in large-scale environments”, *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, p. 1 729 881 418 780 178, 2018. DOI: [10.1177/1729881418780178](https://doi.org/10.1177/1729881418780178). [Online]. Available: <https://doi.org/10.1177/1729881418780178>.
- [28] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli, “COVINS: Visual-Inertial SLAM for Centralized Collaboration”, in *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2021, pp. 171–176. DOI: [10.1109/ISMAR-Adjunct54149.2021.00043](https://doi.ieeecomputersociety.org/10.1109/ISMAR-Adjunct54149.2021.00043). [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/ISMAR-Adjunct54149.2021.00043>.
- [29] R. Egodagamage and M. Tuceryan, “Distributed monocular slam for indoor map building”, *Journal of Sensors*, 2017.
- [30] M. Giamou, K. Khosoussi, and J. How, “Talk resource-efficiently to me: Optimal communication planning for distributed loop closure detection”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1–9.
- [31] T. Cieslewski, S. Choudhary, and D. Scaramuzza, “Data-Efficient Decentralized Visual SLAM”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2466–2473.
- [32] G. Gibbs, H. Jia, and I. Madani, “Obstacle Detection with Ultrasonic Sensors and Signal Analysis Metrics”, *Transportation Research Procedia*, vol. 28, pp. 173–182, 2017, ISSN: 2352-1465. DOI: <https://doi.org/10.1016/j.trpro.2017.12.183>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2352146517311031>.
- [33] D. Beltran and L. Basañez, “A Comparison between Active and Passive 3D Vision Sensors: BumblebeeXB3 and Microsoft Kinect”, in *ROBOT2013: First Iberian Robotics Conference: Advances in Intelligent Systems and Computing*. Springer International Publishing, 2014, pp. 725–734, ISBN: 978-3-319-03413-3. DOI: [10.1007/978-3-319-03413-3\\_54](https://doi.org/10.1007/978-3-319-03413-3_54). [Online]. Available: [https://doi.org/10.1007/978-3-319-03413-3\\_54](https://doi.org/10.1007/978-3-319-03413-3_54).
- [34] R. Labayrade, D. Aubert, and J. Tarel, “Real Time Obstacle Detection in Stereovision on Non Flat Road Geometry Through “V-disparity” Representation”, *IEEE Intelligent Vehicles Symposium*, vol. 2, pp. 646–651, 2002. DOI: [10.1109/IVS.2002.1188024](https://doi.org/10.1109/IVS.2002.1188024).

- [35] R. Labayrade and D. Aubert, “In-vehicle obstacles detection and characterization by stereovision”, *1st International Workshop on in-Vehicle Cognitive*, Jan. 2003.
- [36] H. Oleynikova, D. Honegger, and M. Pollefeys, “Reactive avoidance using embedded stereo vision for MAV flight”, in *IEEE International Conference on Robotics and Automation, ICRA*, IEEE, May 2015, pp. 50–56. DOI: [10.1109/ICRA.2015.7138979](https://doi.org/10.1109/ICRA.2015.7138979). [Online]. Available: <https://doi.org/10.1109/ICRA.2015.7138979>.
- [37] M. Bertozzi, A. Broggi, A. Fascioli, and S. Nichele, “Stereo vision-based vehicle detection”, *IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, pp. 39–44, 2000. DOI: [10.1109/IVS.2000.898315](https://doi.org/10.1109/IVS.2000.898315).
- [38] A. Burlacu, S. Bostaca, I. Hector, P. Hergehelegiu, G. Ivanica, A. Moldoveanu, and S. Caraiman, “Obstacle detection in stereo sequences using multiple representations of the disparity map”, in *International Conference on System Theory, Control and Computing (ICSTCC)*, Oct. 2016, pp. 854–859. DOI: [10.1109/ICSTCC.2016.7790775](https://doi.org/10.1109/ICSTCC.2016.7790775).
- [39] Y. Song, J. Yao, Y. Ju, Y. Jiang, and K. Du, “Automatic Detection and Classification of Road, Car, and Pedestrian Using Binocular Cameras in Traffic Scenes with a Common Framework”, *Complexity*, vol. 2020, pp. 1–17, May 2020, ISSN: 1076-2787. DOI: [10.1155/2020/2435793](https://doi.org/10.1155/2020/2435793). [Online]. Available: <https://doi.org/10.1155/2020/2435793>.
- [40] J. M. S. Martinez and F. E. Ruiz, “Stereo-based aerial obstacle detection for the visually impaired”, in *Workshop on Computer Vision Applications for the Visually Impaired*, 2008, pp. 1–14.
- [41] H.-C. Huang, C.-T. Hsieh, and C.-H. Yeh, “An Indoor Obstacle Detection System Using Depth Information and Region Growth”, *Sensors*, vol. 15, no. 10, pp. 27 116–27 141, 2015, ISSN: 1424-8220. DOI: [10.3390/s151027116](https://doi.org/10.3390/s151027116). [Online]. Available: <https://www.mdpi.com/1424-8220/15/10/27116>.
- [42] Z. Zhang, “Microsoft kinect sensor and its effect”, *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012.
- [43] L. Keselman, J. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, “Intel(R) RealSense(TM) Stereoscopic Depth Cameras”, in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 1267–1276.

- [44] X. Zhu, X. Wu, T. Xu, Z. Feng, and J. Kittler, “Complementary Discriminative Correlation Filters Based on Collaborative Representation for Visual Object Tracking”, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 2, pp. 557–568, 2020. DOI: [10.1109/TCSVT.2020.2979480](https://doi.org/10.1109/TCSVT.2020.2979480).
- [45] X.-F. Zhu, X.-J. Wu, T. Xu, Z.-H. Feng, and J. Kittler, “Robust Visual Object Tracking Via Adaptive Attribute-Aware Discriminative Correlation Filters”, *IEEE Transactions on Multimedia*, vol. 24, pp. 301–312, 2021. DOI: [10.1109/TMM.2021.3050073](https://doi.org/10.1109/TMM.2021.3050073).
- [46] T. Xu, Z. Feng, X. Wu, and J. Kittler, “Adaptive Channel Selection for Robust Visual Object Tracking with Discriminative Correlation Filters”, *International Journal of Computer Vision*, vol. 129, no. 5, pp. 1359–1375, 2021. DOI: [10.1007/s11263-021-01435-1](https://doi.org/10.1007/s11263-021-01435-1).
- [47] S. Hannuna, M. Camplani, J. Hall, M. Mirmehdi, D. Damen, T. Burghardt, A. Paiement, and L. Tao, “DS-KCF: A Real-Time Tracker for RGB-D Data”, *Journal of Real-Time Image Processing*, vol. 16, no. 5, pp. 1439–1458, 2019. DOI: [10.1007/s11554-016-0654-3](https://doi.org/10.1007/s11554-016-0654-3).
- [48] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “High-Speed Tracking with Kernelized Correlation Filters”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015. DOI: [10.1109/TPAMI.2014.2345390](https://doi.org/10.1109/TPAMI.2014.2345390).
- [49] U. Kart, J.-K. Kamarainen, and J. Matas, “How to Make an RGBD Tracker?”, in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, Sep. 2018, pp. 01–15.
- [50] Y. Liu, X. Jing, J. Nie, H. Gao, J. Liu, and G. Jiang, “Context-Aware Three-Dimensional Mean-Shift With Occlusion Handling for Robust Object Tracking in RGB-D Videos”, *IEEE Transactions on Multimedia*, vol. 21, no. 3, pp. 664–677, 2018. DOI: [10.1109/TMM.2018.2863604](https://doi.org/10.1109/TMM.2018.2863604).
- [51] Y. Qian, S. Yan, A. Lukezic, M. Kristan, J. Kamarainen, and J. Matas, “DAL: A Deep Depth-Aware Long-term Tracker”, in *25th International Conference on Pattern Recognition (ICPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jan. 2021, pp. 7825–7832. DOI: [10.1109/ICPR48806.2021.9412984](https://doi.org/10.1109/ICPR48806.2021.9412984). [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/ICPR48806.2021.9412984>.

- [52] S. Yan, J. Yang, J. Kapyla, F. Zheng, A. Leonardis, and J. Kamarainen, “Depth-Track: Unveiling the Power of RGBD Tracking”, in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2021, pp. 10 705–10 713. DOI: [10.1109/ICCV48922.2021.01055](https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.01055). [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/ICCV48922.2021.01055>.
- [53] M. Danelljan, G. Bhat, F. Khan, and M. Felsberg, “ATOM: Accurate Tracking by Overlap Maximization”, in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4660–4669. DOI: [10.1109/CVPR.2019.00479](https://doi.org/10.1109/CVPR.2019.00479).
- [54] G. Bhat, M. Danelljan, L. Van Gool, and R. Timofte, “Learning Discriminative Model Prediction for Tracking”, in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6181–6190.
- [55] G. Yang, F. Chen, C. Wen, M. Fang, Y. Liu, and L. Li, “A new algorithm for obstacle segmentation in dynamic environments using a RGB-D sensor”, in *IEEE International Conference on Real-time Computing and Robotics*, Jun. 2016, pp. 374–378. DOI: [10.1109/RCAR.2016.7784057](https://doi.org/10.1109/RCAR.2016.7784057).
- [56] M. Odelga, P. Stegagno, and H. Bühlhoff, “Obstacle detection, tracking and avoidance for a teleoperated UAV”, in *IEEE international conference on robotics and automation (ICRA)*, 2016, pp. 2984–2990. DOI: [10.1109/ICRA.2016.7487464](https://doi.org/10.1109/ICRA.2016.7487464).
- [57] J. Luiten, T. Fischer, and B. Leibe, “Track to reconstruct and reconstruct to track”, *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1803–1810, 2020. DOI: [10.1109/LRA.2020.2969183](https://doi.org/10.1109/LRA.2020.2969183).
- [58] X. Shi, D. Li, P. Zhao, Q. Tian, Y. Tian, Q. Long, C. Zhu, J. Song, F. Qiao, L. Song, Y. Guo, Z. Wang, Y. Zhang, B. Qin, W. Yang, F. Wang, R. Chan, and S. Q, “Are We Ready for Service Robots? The OpenLORIS-Scene Datasets for Lifelong SLAM”, in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3139–3145.
- [59] G. Klein and D. Murray, “Parallel Tracking and Mapping for Small AR Workspaces”, in *6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234. DOI: [10.1109/ISMAR.2007.4538852](https://doi.org/10.1109/ISMAR.2007.4538852).
- [60] E. Rosten and T. Drummond, “Machine Learning for High-Speed Corner Detection”, in *Computer Vision – ECCV 2006*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443, ISBN: 978-3-540-33833-8.

- [61] G. Klein and D. Murray, “Parallel Tracking and Mapping on a Camera Phone”, in *Eighth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR’09)*, Orlando, Oct. 2009.
- [62] T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera, and J. Jacobo Berlles, “S-PTAM: Stereo Parallel Tracking and Mapping”, *Robotics and Autonomous Systems (RAS)*, vol. 93, pp. 27–42, 2017, ISSN: 0921-8890. DOI: [10.1016/j.robot.2017.03.019](https://doi.org/10.1016/j.robot.2017.03.019).
- [63] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. Jacobo berlles, “Stereo Parallel Tracking and Mapping for robot localization”, in *Proc. of the International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 1373–1378. DOI: [10.1109/IROS.2015.7353546](https://doi.org/10.1109/IROS.2015.7353546).
- [64] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”, *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. DOI: [10.1109/TR0.2015.2463671](https://doi.org/10.1109/TR0.2015.2463671).
- [65] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer, “PL-SLAM: Real-Time Monocular Visual SLAM with Points and Lines”, in *International Conference in Robotics and Automation*, 2017, pp. 4503–4508. DOI: [10.1109/ICRA.2017.7989522](https://doi.org/10.1109/ICRA.2017.7989522).
- [66] A. Pumarola, A. Vakhitov, A. Agudo, F. Moreno-Noguer, and A. Sanfeliu, “Relative Localization for Aerial Manipulation with PL-SLAM”, in *Aerial Robotic Manipulation*. Springer, 2019, pp. 239–248, ISBN: 978-3-030-12945-3. DOI: [10.1007/978-3-030-12945-3\\_17](https://doi.org/10.1007/978-3-030-12945-3_17). [Online]. Available: [https://doi.org/10.1007/978-3-030-12945-3\\_17](https://doi.org/10.1007/978-3-030-12945-3_17).
- [67] P. Kim, B. Coltin, and H. J. Kim, “Low-Drift Visual Odometry in Structured Environments by Decoupling Rotational and Translational Motion”, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7247–7253. DOI: [10.1109/ICRA.2018.8463207](https://doi.org/10.1109/ICRA.2018.8463207).
- [68] S. Yang and S. Scherer, “Cubeslam: Monocular 3-d object slam”, *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 925–938, 2019.
- [69] R. Elvira, J. D. Tardós, and J. Montiel, “ORB-SLAM-Atlas: a robust and accurate multi-map system”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6253–6259. DOI: [10.1109/IROS40897.2019.8967572](https://doi.org/10.1109/IROS40897.2019.8967572).



- [70] G. Silveira, E. Malis, and P. Rives, “An Efficient Direct Approach to Visual SLAM”, *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 969–979, 2008. DOI: [10.1109/TR0.2008.2004829](https://doi.org/10.1109/TR0.2008.2004829).
- [71] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, “DTAM: Dense tracking and mapping in real-time”, in *International Conference on Computer Vision*, 2011, pp. 2320–2327. DOI: [10.1109/ICCV.2011.6126513](https://doi.org/10.1109/ICCV.2011.6126513).
- [72] J. Engel, T. Schöps, and D. Cremers, “LSD-SLAM: Large-Scale Direct Monocular SLAM”, in *Computer Vision – ECCV 2014*, Springer International Publishing, 2014, pp. 834–849, ISBN: 978-3-319-10605-2.
- [73] J. Civera, A. J. Davison, and J. M. M. Montiel, “Inverse Depth Parametrization for Monocular SLAM”, *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 932–945, 2008. DOI: [10.1109/TR0.2008.2003276](https://doi.org/10.1109/TR0.2008.2003276).
- [74] C. Forster, M. Pizzoli, and D. Scaramuzza, “SVO: Fast Semi-Direct Monocular Visual Odometry”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.
- [75] R. Gomez-Ojeda, J. Briales, and J. Gonzalez-Jimenez, “PL-SVO: Semi-direct monocular visual odometry by combining points and line segments”, in *International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2016, pp. 4211–4216.
- [76] J. Engel, V. Koltun, and D. Cremers, “Direct Sparse Odometry”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Mar. 2018.
- [77] X. Gao, R. Wang, N. Demmel, and D. Cremers, “LDSO: Direct Sparse Odometry with Loop Closure”, in *International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2018.
- [78] K. Konolige, M. Agrawal, and J. Solà, “Large-Scale Visual Odometry for Rough Terrain”, in *Robotics Research*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 201–212, ISBN: 978-3-642-14743-2.
- [79] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, “On-Manifold Preintegration for Real-Time Visual-Inertial Odometry”, *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, Feb. 2017, ISSN: 1552-3098. DOI: [10.1109/TR0.2016.2597321](https://doi.org/10.1109/TR0.2016.2597321). [Online]. Available: <https://doi.org/10.1109/TR0.2016.2597321>.
- [80] T. Qin, P. Li, and S. Shen, “Vins-mono: A robust and versatile monocular visual-inertial state estimator”, *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018. DOI: [10.1109/TR0.2018.2853729](https://doi.org/10.1109/TR0.2018.2853729).



- [81] Y. He, J. Zhao, Y. Guo, W. He, and K. Yuan, “PL-VIO: Tightly-Coupled Monocular Visual-Inertial Odometry Using Point and Line Features”, *Sensors*, vol. 18, no. 4, 2018, issn: 1424-8220. DOI: [10.3390/s18041159](https://doi.org/10.3390/s18041159). [Online]. Available: <https://www.mdpi.com/1424-8220/18/4/1159>.
- [82] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, “Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1689–1696. DOI: [10.1109/ICRA40945.2020.9196885](https://doi.org/10.1109/ICRA40945.2020.9196885).
- [83] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone, “3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans”, in *Robotics: Science and Systems (RSS)*, 2020.
- [84] C. Campos, R. Elvira, J. Rodriguez, J. Montiel, and J. Tardos, “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multimap SLAM”, *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021.
- [85] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, “Towards a swarm of agile micro quadrotors”, *Autonomous Robot*, vol. 35, no. 4, pp. 287–300, 2013. DOI: [10.1007/s10514-013-9349-9](https://doi.org/10.1007/s10514-013-9349-9).
- [86] G. Loianno, Y. Mulgaonkar, C. Brunner, D. Ahuja, A. Ramanandan, M. Chari, S. Diaz, and V. Kumar, “A swarm of flying smartphones”, in *International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 1681–1688.
- [87] M. Schwager, P. Dames, D. Rus, and V. Kumar, “A multi-robot control policy for information gathering in the presence of unknown hazards”, in *Robotics research*, 2017, pp. 455–472.
- [88] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, “Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models”, *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1286–1311, 2017. DOI: [10.1177/0278364917732640](https://doi.org/10.1177/0278364917732640). [Online]. Available: <https://doi.org/10.1177/0278364917732640>.
- [89] D. Zou and P. Tan, “CoSLAM: Collaborative Visual SLAM in Dynamic Environments”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 354–366, 2013. DOI: [10.1109/TPAMI.2012.104](https://doi.org/10.1109/TPAMI.2012.104).
- [90] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, “Collaborative monocular SLAM with multiple Micro Aerial Vehicles”, in *International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 3962–3970. DOI: [10.1109/IROS.2013.6696923](https://doi.org/10.1109/IROS.2013.6696923).

- [91] L. Riazuelo, J. Civera, and J. Montiel, “C<sup>2</sup>TAM: A Cloud framework for co-operative tracking and mapping”, *Robotics and Autonomous Systems*, vol. 62, no. 4, pp. 401–413, 2014, ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2013.11.007>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0921889013002248>.
- [92] I. Deutsch, M. Liu, and R. Siegwart, “A framework for multi-robot pose graph SLAM”, in *IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2016, pp. 567–572. DOI: [10.1109/RCAR.2016.7784092](https://doi.org/10.1109/RCAR.2016.7784092).
- [93] M. Ouyang, X. Shi, Y. Wang, Y. Tian, Y. Shen, D. Wang, P. Wang, and Z. Cao, “A Collaborative Visual SLAM Framework for Service Robots”, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 8679–8685. DOI: [10.1109/IROS51168.2021.9636798](https://doi.org/10.1109/IROS51168.2021.9636798).
- [94] J. Liu, R. Liu, K. Chen, J. Zhang, and D. Guo, “Collaborative Visual Inertial SLAM for Multiple Smart Phones”, *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11 553–11 559, 2021.
- [95] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154. DOI: [10.1109/IROS.2004.1389727](https://doi.org/10.1109/IROS.2004.1389727).
- [96] D. Zhang, “Extended Closing Operation in Morphology and Its Application in Image Processing”, in *International Conference on Information Technology and Computer Science*, vol. 1, 2009, pp. 83–87. DOI: [10.1109/ITCS.2009.268](https://doi.org/10.1109/ITCS.2009.268).
- [97] K. Wu, E. Otoo, and K. Suzuki, “Optimizing two-pass connected-component labeling algorithms”, *Pattern Analysis and Applications*, vol. 12, no. 2, pp. 117–135, 2009. DOI: [10.1007/s10044-008-0109-y](https://doi.org/10.1007/s10044-008-0109-y). [Online]. Available: <https://doi.org/10.1007/s10044-008-0109-y>.
- [98] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart, “Extending kalibr: Calibrating the extrinsics of multiple IMUs and of individual axes”, in *IEEE International Conference on Robotics and Automation, ICRA*, 2016, pp. 4304–4311. DOI: [10.1109/ICRA.2016.7487628](https://doi.org/10.1109/ICRA.2016.7487628). [Online]. Available: <https://doi.org/10.1109/ICRA.2016.7487628>.
- [99] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, “RViz: a toolkit for real domain data visualization”, *Telecommunication Systems*, vol. 60, no. 2, pp. 337–345, Oct. 2015, ISSN: 1018-4864. DOI: [10.1007/s11235-015-0034-5](https://doi.org/10.1007/s11235-015-0034-5). [Online]. Available: <https://doi.org/10.1007/s11235-015-0034-5>.

- [100] M.-K. Hu, “Visual pattern recognition by moment invariants”, *IRE transactions on information theory*, vol. 8, no. 2, pp. 179–187, 1962. DOI: [10.1109/TIT.1962.1057692](https://doi.org/10.1109/TIT.1962.1057692).
- [101] A. Koubaa, *Robot Operating System (ROS): The Complete Reference (Volume 1)*. Springer Publishing Company, Incorporated, 2016. DOI: [10.1007/978-3-319-26054-9](https://doi.org/10.1007/978-3-319-26054-9).
- [102] R. Gariepy, P. Mukherjee, P. Bovbel, and D Ash. (2019). Husky: Common packages for the Clearpath Husky., [Online]. Available: <https://github.com/husky/husky>. (Accessed: 06.03.2023).
- [103] R. E. Schapire, “Explaining Adaboost”, in *Empirical inference*, Springer, 2013, pp. 37–52.
- [104] Z. Kalal, K. Mikolajczyk, and J. Matas, “Forward-Backward Error: Automatic Detection of Tracking Failures”, in *International Conference on Pattern Recognition*, IEEE Computer Society, 2010, pp. 2756–2759. DOI: [10.1109/ICPR.2010.675](https://doi.org/10.1109/ICPR.2010.675). [Online]. Available: <https://doi.org/10.1109/ICPR.2010.675>.
- [105] B. Babenko, M.-H. Yang, and S. Belongie, “Visual tracking with online Multiple Instance Learning”, in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 983–990.
- [106] D. Bolme, J. Beveridge, B. Draper, and Y. Lui, “Visual object tracking using adaptive correlation filters”, in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2544–2550. DOI: [10.1109/CVPR.2010.5539960](https://doi.org/10.1109/CVPR.2010.5539960).
- [107] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-Learning-Detection”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 07, pp. 1409–1422, 2012, ISSN: 1939-3539. DOI: [10.1109/TPAMI.2011.239](https://doi.org/10.1109/TPAMI.2011.239).
- [108] G. Bradski, “The OpenCV Library”, *Dr. Dobb’s Journal of Software Tools*, 2000.
- [109] A. Saha, B. C. Dhara, S. Umer, A. A. AlZubi, J. M. Alanazi, and K. Yurii, “CORB2I-SLAM: An Adaptive Collaborative Visual-Inertial SLAM for Multiple Robots”, *Electronics*, vol. 11, no. 18, 2022, ISSN: 2079-9292. DOI: [10.3390/electronics11182814](https://doi.org/10.3390/electronics11182814). [Online]. Available: <https://www.mdpi.com/2079-9292/11/18/2814>.

- [110] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, “Bundle Adjustment – A Modern Synthesis”, in *Vision Algorithms: Theory and Practice*, vol. 1883, Springer Berlin Heidelberg, 2000, pp. 298–372. DOI: [10.1007/3-540-44480-7\\_21](https://doi.org/10.1007/3-540-44480-7_21). [Online]. Available: [https://doi.org/10.1007/3-540-44480-7\\_21](https://doi.org/10.1007/3-540-44480-7_21).
- [111] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “Ros: an open-source robot operating system”, in *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, 2009.
- [112] G. S. Watson, “Linear Least Squares Regression”, *The Annals of Mathematical Statistics*, vol. 38, no. 6, pp. 1679–1699, 1967. DOI: [10.1214/aoms/1177698603](https://doi.org/10.1214/aoms/1177698603). [Online]. Available: <https://doi.org/10.1214/aoms/1177698603>.
- [113] T. Chen and Q. Wang, “3D Line Segment Detection for Unorganized Point Clouds from Multi-view Stereo”, in *Computer Vision – ACCV 2010*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 400–411, ISBN: 978-3-642-19309-5. DOI: [10.1007/978-3-642-19309-5\\_31](https://doi.org/10.1007/978-3-642-19309-5_31). [Online]. Available: [https://doi.org/10.1007/978-3-642-19309-5\\_31](https://doi.org/10.1007/978-3-642-19309-5_31).
- [114] Y. Lin, C. Wang, J. Cheng, B. Chen, F. Jia, Z. Chen, and J. Li, “Line segment extraction for large scale unorganized point clouds”, *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 102, Apr. 2015. DOI: [10.1016/j.isprsjprs.2014.12.027](https://doi.org/10.1016/j.isprsjprs.2014.12.027).
- [115] P. Tian, X. Hua, W. Tao, and M. Zhang, “Robust Extraction of 3D Line Segment Features from Unorganized Building Point Clouds”, *Remote Sensing*, vol. 14, no. 14, 2022, ISSN: 2072-4292. DOI: [10.3390/rs14143279](https://doi.org/10.3390/rs14143279). [Online]. Available: <https://www.mdpi.com/2072-4292/14/14/3279>.
- [116] J. Blanco, F. Moreno, and J. Gonzalez-Jimenez, “The malaga urban dataset: High-rate stereo and lidars in a realistic urban scenario”, *International Journal of Robotics Research*, vol. 33, no. 2, pp. 207–214, 2014. DOI: [10.1177/0278364913507326](https://doi.org/10.1177/0278364913507326). [Online]. Available: <https://doi.org/10.1177/0278364913507326>.
- [117] D. Galvez-López and J. D. Tardos, “Bags of Binary Words for Fast Place Recognition in Image Sequences”, *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012. DOI: [10.1109/TR0.2012.2197158](https://doi.org/10.1109/TR0.2012.2197158).
- [118] R. Hartley, J. Trumpf, Y. Dai, and H. li, “Rotation averaging”, *International Journal of Computer Vision*, vol. 103, no. 3, pp. 267–305, 2013. DOI: [10.1007/s11263-012-0601-0](https://doi.org/10.1007/s11263-012-0601-0).

- [119] B. Zhuang, L.-F. Cheong, and G. Lee, “Baseline desensitizing in translation averaging”, in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4539–4547. DOI: [10.1109/CVPR.2018.00477](https://doi.org/10.1109/CVPR.2018.00477).
- [120] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, “The EuRoC micro aerial vehicle datasets”, *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016. DOI: [10.1177/0278364915620033](https://doi.org/10.1177/0278364915620033).
- [121] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A Benchmark for the Evaluation of RGB-D SLAM Systems”, in *International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 573–580. DOI: [10.1109/IROS.2012.6385773](https://doi.org/10.1109/IROS.2012.6385773).
- [122] L. Zheng, P. Zhang, J. Tan, and F. Li, “The Obstacle Detection Method of UAV Based on 2D Lidar”, *IEEE Access*, vol. 7, pp. 163 437–163 448, 2019. DOI: [10.1109/ACCESS.2019.2952173](https://doi.org/10.1109/ACCESS.2019.2952173).
- [123] D. Xie, Y. Xu, and R. Wang, “Obstacle detection and tracking method for autonomous vehicle based on three-dimensional LiDAR”, *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, p. 1 729 881 419 831 587, 2019. DOI: [10.1177/1729881419831587](https://doi.org/10.1177/1729881419831587).
- [124] H. Wang and X. Zhang, “Real-time vehicle detection and tracking using 3D LiDAR”, *Asian Journal of Control*, vol. 24, no. 3, pp. 1459–1469, 2022. DOI: <https://doi.org/10.1002/asjc.2519>.
- [125] M. Dulău and F. Oniga, “Obstacle Detection Using a Facet-Based Representation from 3-D LiDAR Measurements”, *Sensors*, vol. 21, no. 20, 2021. DOI: [10.3390/s21206861](https://doi.org/10.3390/s21206861). [Online]. Available: <https://www.mdpi.com/1424-8220/21/20/6861>.
- [126] A. Asvadi, C. Premevida, P. Peixoto, and U. Nunes, “3D Lidar-Based Static and Moving Obstacle Detection in Driving Environments”, *Robot. Auton. Syst.*, vol. 83, no. C, pp. 299–311, Sep. 2016, ISSN: 0921-8890. DOI: [10.1016/j.robot.2016.06.007](https://doi.org/10.1016/j.robot.2016.06.007).
- [127] A. Kadambi, A. Bhandari, and R. Raskar, “3D Depth Cameras in Vision: Benefits and Limitations of the Hardware”, in *Computer Vision and Machine Learning with RGB-D Sensors*. Springer International Publishing, 2014, pp. 3–26, ISBN: 978-3-319-08651-4. DOI: [10.1007/978-3-319-08651-4\\_1](https://doi.org/10.1007/978-3-319-08651-4_1).

- [128] T.-M. Wang and Z.-C. Shih, “Measurement and Analysis of Depth Resolution Using Active Stereo Cameras”, *IEEE Sensors Journal*, vol. 21, no. 7, pp. 9218–9230, 2021. DOI: [10.1109/JSEN.2021.3054820](https://doi.org/10.1109/JSEN.2021.3054820).
- [129] V. Tadic, A. Toth, Z. Vizvari, M. Klincsik, Z. Sari, P. Sarcevic, J. Sarosi, and I. Biro, “Perspectives of RealSense and ZED Depth Sensors for Robotic Vision Applications”, *Machines*, vol. 10, no. 3, 2022, ISSN: 2075-1702. DOI: [10.3390/machines10030183](https://doi.org/10.3390/machines10030183). [Online]. Available: <https://www.mdpi.com/2075-1702/10/3/183>.
- [130] A. Saha and B. C. Dhara, “3D LiDAR-based Obstacle Detection and Tracking for Autonomous Navigation in Dynamic Environments”, *International Journal of Intelligent Robotics and Applications*, vol. 8, pp. 39–60, 2024. DOI: [10.1007/s41315-023-00302-1](https://doi.org/10.1007/s41315-023-00302-1). [Online]. Available: <https://doi.org/10.1007/s41315-023-00302-1>.
- [131] G. Yang, S. Mentasti, M. Bersani, Y. Wang, F. Braghin, and F. Cheli, “LiDAR point-cloud processing based on projection methods: a comparison”, *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, pp. 1–6, 2020.
- [132] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and R. Daniela, “LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5135–5142.
- [133] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset”, *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013. DOI: [10.1177/0278364913491297](https://doi.org/10.1177/0278364913491297). [Online]. Available: <https://doi.org/10.1177/0278364913491297>.
- [134] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”, *Autonomous Robots*, 2013. DOI: [10.1007/s10514-012-9321-0](https://doi.org/10.1007/s10514-012-9321-0). [Online]. Available: <https://octomap.github.io>.
- [135] *Turtlebot3: Personal Robot Kit*, (last accessed 30th June, 2023). [Online]. Available: <https://www.turtlebot.com>.
- [136] R. Mateus Gago, G. A. S. Pereira, and M. Y. A. Pereira, *Aerial Lidar Dataset of an Indoor Stockpile Warehouse*, 2020. DOI: [10.21227/zyxc-wq04](https://doi.org/10.21227/zyxc-wq04). [Online]. Available: <https://dx.doi.org/10.21227/zyxc-wq04>.



- [137] R. M. Gago, M. Y. A. Pereira, and G. A. S. Pereira, “An aerial robotic system for inventory of stockpile warehouses”, *Engineering Reports*, vol. 3, no. 9, Mar. 2021. DOI: [10.1002/eng2.12396](https://doi.org/10.1002/eng2.12396). [Online]. Available: <https://doi.org/10.1002/eng2.12396>.
- [138] *OpenMANIPULATOR-X: Robot simulation made easy*, (last accessed 30th June, 2023). [Online]. Available: [https://emmanual.robotis.com/docs/en/platform/openmanipulator\\_x/overview/](https://emmanual.robotis.com/docs/en/platform/openmanipulator_x/overview/).
- [139] N. Point, “OptiTrack”, *Natural Point, Inc*, 2011.
- [140] A. Saha, L. Kumar, S. Sortee, and B. C. Dhara, “An Autonomous Aircraft Inspection System using Collaborative Unmanned Aerial Vehicles”, in *IEEE Aerospace Conference (AERO)*, 2023, pp. 1–10. DOI: [10.1109/AERO55745.2023.10115655](https://doi.org/10.1109/AERO55745.2023.10115655).
- [141] P. Silberberg and R. C. Leishman, “Aircraft Inspection by Multirotor UAV Using Coverage Path Planning”, in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2021, pp. 575–581. DOI: [10.1109/ICUAS51884.2021.9476718](https://doi.org/10.1109/ICUAS51884.2021.9476718).
- [142] J. Miranda, S. Larnier, A. Herbulot, and M. Devy, “UAV-based Inspection of Airplane Exterior Screws with Computer Vision”, in *14<sup>th</sup> International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)*, INSTICC, SciTePress, 2019, pp. 421–427, ISBN: 978-989-758-354-4. DOI: [10.5220/0007571304210427](https://doi.org/10.5220/0007571304210427).
- [143] B. He, B. Huang, Y. Lin, and L. Wu, “Intelligent unmanned aerial vehicle (UAV) system for aircraft surface inspection”, in *7<sup>th</sup> International Forum on Electrical Engineering and Automation (IFEEA)*, 2020, pp. 316–321. DOI: [10.1109/IFEEA51475.2020.00073](https://doi.org/10.1109/IFEEA51475.2020.00073).
- [144] D. Cazzato, M. A. Olivares-Mendez, J. L. Sanchez-Lopez, and H. Voos, “Vision-Based Aircraft Pose Estimation for UAVs Autonomous Inspection without Fiducial Markers”, in *45<sup>th</sup> Annual Conference of the IEEE Industrial Electronics Society (IECON)*, vol. 1, 2019, pp. 5642–5648. DOI: [10.1109/IECON.2019.8926667](https://doi.org/10.1109/IECON.2019.8926667).
- [145] K. Malandrakis, A. Savvaris, J. A. G. Domingo, N. Avdelidis, P. Tsilivis, F. Plumacker, L. Z. Fragonara, and A. Tsourdos, “Inspection of Aircraft Wing Panels Using Unmanned Aerial Vehicles”, in *5<sup>th</sup> IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, 2018, pp. 56–61. DOI: [10.1109/MetroAeroSpace.2018.8453598](https://doi.org/10.1109/MetroAeroSpace.2018.8453598).



- [146] K. Maeda, S. Doki, Y. Funabara, and K. Doki, “Flight Path Planning of Multiple UAVs for Robust Localization near Infrastructure Facilities”, in *44<sup>th</sup> Annual Conference of the IEEE Industrial Electronics Society (IECON)*, 2018, pp. 2522–2527. DOI: [10.1109/IECON.2018.8592710](https://doi.org/10.1109/IECON.2018.8592710).
- [147] L. Ruiqian, X. Juan, and Z. Hongfu, “Automated Surface Defects Acquisition System of Civil Aircraft Based on Unmanned Aerial Vehicles”, in *IEEE 2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT)*, 2020, pp. 729–733. DOI: [10.1109/ICCASIT50869.2020.9368540](https://doi.org/10.1109/ICCASIT50869.2020.9368540).
- [148] Z. Cai and N. Vasconcelos, “Cascade R-CNN: High Quality Object Detection and Instance Segmentation”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1483–1498, 2021. DOI: [10.1109/TPAMI.2019.2956516](https://doi.org/10.1109/TPAMI.2019.2956516).
- [149] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF”, in *International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: [10.1109/ICCV.2011.6126544](https://doi.org/10.1109/ICCV.2011.6126544).
- [150] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, “Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo”, in *3rd International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, Springer Berlin Heidelberg, 2012, pp. 400–411, ISBN: 978-3-642-34327-8. DOI: [10.1007/978-3-642-34327-8\\_36](https://doi.org/10.1007/978-3-642-34327-8_36). [Online]. Available: [https://doi.org/10.1007/978-3-642-34327-8\\_36](https://doi.org/10.1007/978-3-642-34327-8_36).
- [151] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: <http://www.blender.org>.
- [152] J. Kuffner and S. LaValle, “RRT-connect: An efficient approach to single-query path planning”, in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, 2000, pp. 995–1001. DOI: [10.1109/ROBOT.2000.844730](https://doi.org/10.1109/ROBOT.2000.844730).
- [153] J. Pan, S. Chitta, and D. Manocha, “FCL: A general purpose library for collision and proximity queries”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 3859–3866. DOI: [10.1109/ICRA.2012.6225337](https://doi.org/10.1109/ICRA.2012.6225337).
- [154] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”, *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996. DOI: [10.1109/70.508439](https://doi.org/10.1109/70.508439).

- [155] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL)”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1–4. DOI: [10.1109/ICRA.2011.5980567](https://doi.org/10.1109/ICRA.2011.5980567).
- [156] R. B. Rusu, N. Blodow, and M. Beetz, “Fast Point Feature Histograms (FPFH) for 3D registration”, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009, pp. 3212–3217. DOI: [10.1109/ROBOT.2009.5152473](https://doi.org/10.1109/ROBOT.2009.5152473).
- [157] R. B. Rusu, N. Blodow, Z. C. Marton, and M. Beetz, “Aligning point cloud views using persistent feature histograms”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 3384–3391. DOI: [10.1109/IRROS.2008.4650967](https://doi.org/10.1109/IRROS.2008.4650967).
- [158] A. Farhadi and J. Redmon, “Yolov3: An incremental improvement”, in *Computer vision and pattern recognition*, Springer Berlin/Heidelberg, Germany, vol. 1804, 2018, pp. 1–6.
- [159] Q. Zhan, Y. Liang, and Y. Xiao, “Color-based segmentation of point clouds”, *ISPRS Laser Scanning Workshop*, vol. 38, Jul. 2009.
- [160] O. Özyesil, V. Voroninski, R. Basri, and A. Singer, “A survey of structure from motion.”, *Acta Numerica*, vol. 26, pp. 305–364, 2017. DOI: [10.1017/S096249291700006X](https://doi.org/10.1017/S096249291700006X).
- [161] D. G. Lowe, “Object recognition from local scale-invariant features”, in *Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157. DOI: [10.1109/ICCV.1999.790410](https://doi.org/10.1109/ICCV.1999.790410).
- [162] H. Bay, T. Tuytelaars, and L. Van Gool, “SURF: Speeded Up Robust Features”, in *European Conference on Computer Vision*, Springer Berlin Heidelberg, 2006, pp. 404–417, ISBN: 978-3-540-33833-8.
- [163] V. Klema and A. Laub, “The singular value decomposition: Its computation and some applications”, *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980. DOI: [10.1109/TAC.1980.1102314](https://doi.org/10.1109/TAC.1980.1102314).
- [164] M. A. Fischler and R. C. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”, *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981, ISSN: 0001-0782. DOI: [10.1145/358669.358692](https://doi.org/10.1145/358669.358692). [Online]. Available: <https://doi.org/10.1145/358669.358692>.
- [165] H. C. Longuet-Higgins, “A computer algorithm for reconstructing a scene from two projections”, *Nature*, vol. 293, pp. 133–135, 1981. DOI: [10.1038/293133a0](https://doi.org/10.1038/293133a0).

- [166] R. I. Hartley and P. Sturm, "Triangulation", *Computer Vision and Image Understanding*, vol. 68, no. 2, pp. 146–157, 1997.
- [167] H. Noble and R. Heale, "Triangulation in research, with examples", *Evidence-Based Nursing*, vol. 22, no. 3, pp. 67–68, 2019, ISSN: 1367-6539. DOI: 10.1136/ebnurs-2019-103145.
- [168] X. X. Lu, "A Review of Solutions for Perspective-n-Point Problem in Camera Pose Estimation", in *Journal of Physics Conference Series*, ser. Journal of Physics Conference Series, vol. 1087, 2018. DOI: 10.1088/1742-6596/1087/5/052009.
- [169] M. I. A. Lourakis and A. A. Argyros, "SBA: A software package for generic sparse bundle adjustment", *ACM Transactions on Mathematical Software*, vol. 36, no. 1, pp. 1–30, 2009. DOI: 10.1145/1486525.1486527. [Online]. Available: <https://doi.org/10.1145/1486525.1486527>.
- [170] C. Wu, "Towards Linear-Time Incremental Structure from Motion", in *2013 International Conference on 3D Vision - 3DV 2013*, 2013, pp. 127–134. DOI: 10.1109/3DV.2013.25.
- [171] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz, "Multicore bundle adjustment", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 3057–3064. DOI: 10.1109/CVPR.2011.5995552.
- [172] Y. Furukawa and J. Ponce, "Accurate, Dense, and Robust Multiview Stereopsis", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1362–1376, 2010. DOI: 10.1109/TPAMI.2009.161.
- [173] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski, "Towards Internet-scale multi-view stereo", in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 1434–1441. DOI: 10.1109/CVPR.2010.5539802.

  
07.03.2024

**PROFESSOR**  
Deptt. of Information Technology  
JADAVPUR UNIVERSITY  
Block -LB, Plot-8, Sector-3  
Salt Lake, Kolkata-700106, India