Autonomous System for Legged Robots: From Calibration and Pose Estimation to CLF Reactive Motion Planning

by

Jiunn-Kai Huang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Robotics)
in the University of Michigan
2022

Doctoral Committee:

Professor Jessy W. Grizzle, Chair
Assistant Professor Maani Ghaffari Jadidi
Professor Benjamin Kuipers
Associate Professor Ram Vasudevan
Dr. Jeffrey Walls, Toyota Research Institute

Jiunn-Kai Huang

bjhuang@umich.edu

ORCID iD: 0000-0002-3589-3027

DEDICATION

To my parents, family, and my partner, Won. Without your support, none of this work would have been possible.

ACKNOWLEDGMENTS

Wow, what a splendid adventure! Over the past four years, I have been so focused on the PhD degree that it is now that I have the chance to reflect on it all.

First of all, I would like to thank my advisor, Professor Jessy Grizzle, for taking me, who has an entirely different background, as his PhD student and for allowing me to follow my passions and to do research in both perception and motion planning. You not only showed me how to do top-tier research but also demonstrated how to be a good leader. I would also like to thank Professor Maani Ghaffari Jadidi, who guided me on the chapter of Lie group, reproducing kernel Hilbert space, and many other mapping and localization techniques. Next, I would like to thank the other members of my committee – Professor Benjamin Kuipers, Professor Ram Vasudevan, and Dr. Jeffrey Walls. Thank you for your helpful comments, time, mentorship, direction, and your advice and conversations regarding my goals. Denise Edmund, Damen Provost, and Dan Newman, thank you for always being so kind, helpful, and patient.

Thank you to all the members of the Biped Robotics Lab: Xingye (Dennis) Da, Omar Harib, Ross Hartley, Yukai Gong, Eva Mungai, Grant Gibson, and Oluwami Dosunmu-Ogunbi. Your support and patience in the lab were invaluable. Thank you Dennis for helping me order impossible GPUs when they had been out of stock for several months and for introducing me to many people at your workspace. Thank you Omar for the countless hours of discussion of coding techniques. I am thankful for Ross Hartley, all the journey of biped lab starting from you taking me to assist your research when I was an MS student. Also, thank you for helping me get on track in my first year of the PhD and for the many hours at the whiteboard. Thank you Eva for pushing me to design the lab logo, jacket, and website, as well as for organizing many of lab gatherings ;-). Thank you Wami; your laughs in the lab certainly make my day, always. Thank you Grant for your support in autonomy experiments and for your fantastic design of the torso stand! Yukai, I have not forgotten you! I do appreciate your awesome development of Cassie's controllers. Your controllers have enabled Cassie to walk on various types of terrain. None of the autonomy work would have been possible without your code. We had been working on autonomy for endless days and nights. I still remember the day that we drove together to Pittsburgh to take a Cassie's battery. Thank you also for many instant control lectures.

To Chih-Kang (Ken) Chang and Lu Gan, thanks for your company to take classes and fight for exams in late nights. To Tzu-Yuan (Justin) Lin and Chien-Erh (Cynthia) Lin, your friendship

made the long journey joyful. To Jiayang (David) Xu, Tingting Liu, and Ziyi Ye, you guys were an amazing group of people; thank you all for being so consistently encouraging and supportive.

To all the 17 MS students (in the order I worked with), Kaustav Chakraborty, Chenxi Feng, Madhav Achar, Isabel Taylor, Shoutian Wang, Hengxu You, Santoshi Kulkarni, Jianyang Tang, Minzhe Li, Peter Wrobel, Yingwen Tan, Jinze Liu, Dongmyeong Lee, Kuan-Ting Lee, Ruohua Li, Rui Chen, and Qi Dai, thank you for assisting my research and for achieving my goal. For the MS students jointing autonomy experiments, research will be outdated, and technology will be updated. However, our memory of working together on the Wave Field and the Ford Robotics Building will remain. Lastly, thank you, Cassie Blue and Digit; without your attendance, completing this dissertation would have been impossible.

Jiunn-Kai (Bruce) Huang

April 30, 2022

TABLE OF CONTENTS

Dedication . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Figures . . . . . **x**

List of Tables . . . . . **xiii**

List of Appendices . . . . . **xiv**

List of Algorithms . . . . . **xv**

List of Acronyms . . . . . **xvi**

Abstract . . . . . **xx**

Chapter

1 Introduction **1**

1.1 Motivation 1
1.2 Objectives 2
1.3 Dissertation Roadmap 6
1.4 Contributions 7
1.5 Open-source Software 10

Part I Autonomous System for Legged Robots and Sensor Calibration 12

2 Design of Perception Suite for Legged Robots **12**

2.1 Introduction and Sensors 12
2.2 Perception Suite 14

3 Improvements to Target-Based 3D LiDAR to Camera Calibration **16**

3.1 Introduction and Related Work 16
3.1.1 Rough Overview of the Most Common Target-based Approaches 17
3.1.2 Our Contributions 19
3.2 Finding the LiDAR Target Vertices 19
3.2.1 Remarks on LiDAR Point Clouds 20

3.2.2 New Method for Determining Target Vertices . . . 21
3.3 Image Plane Corners and Correspondences with the LiDAR Vertices . . . 22
3.4 Extrinsic Transformation Optimization . . . 23
3.4.1 Euclidean distance . . . 23
3.4.2 IoU optimization . . . 23
3.5 Experimental Results . . . 25
3.5.1 Data Collection . . . 25
3.5.2 Baseline Implementation for LiDAR Vertices . . . 25
3.5.3 Camera Corners and Associations . . . 26
3.5.4 Extrinsic Calibration . . . 26
3.5.5 Computation Performance . . . 28
3.5.6 Quantitative Results and Round-robin Analysis . . . 28
3.6 Qualitative Results and Discussion . . . 29
3.7 Conclusions . . . 30

4 Global Unifying Intrinsic Calibration on Lie Group for Spinning LiDAR and Solid-state LiDAR . . . 32

4.1 Introduction . . . 32
4.1.1 Our Contributions . . . 33
4.2 Related Work . . . 34
4.2.1 Spinning LiDAR . . . 35
4.2.2 Solid-State LiDAR . . . 35
4.3 Proposed LiDAR Intrinsic Model and Analysis . . . 36
4.3.1 Point-to-Plane Distance . . . 36
4.3.2 Overfitting, Underfitting, and Model Mismatch . . . 37
4.3.3 Target Placement Guideline . . . 38
4.3.4 Parsing Points on a Target to Collections of Points . . . 41
4.4 Baseline Intrinsic Calibration Methods . . . 42
4.4.1 3-Parameter Model . . . 42
4.4.2 6-Parameter Model . . . 43
4.4.3 Can the models be expressed by elements of $\mathrm{Sim}(3)$? . . . 43
4.4.4 Cost function for baseline models . . . 44
4.5 Global Optimization: from Min to Min-Min . . . 44
4.5.1 Problem Statement and Compact Sets . . . 45
4.5.2 From Min to Min-Min . . . 45
4.5.3 Determining the Calibration Parameters . . . 46
4.6 Simulation and Experimental Results . . . 47
4.6.1 Simulated LiDAR Environment . . . 47
4.6.2 Intrinsic Calibration of Spinning LiDARs . . . 49
4.6.3 Intrinsic Calibration of Solid-State LiDARs . . . 51
4.7 Conclusions . . . 53

5 Automatic Calibration Pipeline . . . 54

5.1 Automatic Calibration System . . . 54
5.1.1 Front-End . . . 55

5.1.2 Back-End . . . 55
5.1.3 Extrinsic and Intrinsic Calibration . . . 55

Part II Perception for Pose Estimation with Global Solution 57

6 LiDARTag: A Real-Time Fiducial Tag System for Point Clouds . . . 57

6.1 Introduction . . . 57
6.2 Related Work . . . 59
6.3 Tag Design and LiDAR Characteristics . . . 60
6.3.1 LiDAR Point Clouds vs Camera Images . . . 60
6.3.2 Tag Placement and Design . . . 61
6.4 Tag Detection . . . 64
6.4.1 Feature Detection . . . 65
6.4.2 Feature Clustering . . . 65
6.4.3 Cluster Validation . . . 66
6.5 Pose Estimation and Initialization . . . 69
6.5.1 LiDARTag Pose Estimation . . . 69
6.5.2 Optimization Initialization . . . 70
6.6 Function Construction and Tag Decoding . . . 72
6.7 Experimental Results . . . 74
6.7.1 Pose Evaluation and Decoding Accuracy . . . 74
6.7.2 LiDARTag System and Speed Analysis . . . 76
6.7.3 False Positive Analysis . . . 78
6.8 Conclusion and Future Work . . . 79

7 Optimal Shape Design for LiDAR Pose Estimation and Global Solver . . . 81

7.1 Introduction . . . 81
7.1.1 Contributions . . . 82
7.2 Related Work . . . 83
7.2.1 Fiducial Markers . . . 83
7.2.2 Target-Based LiDAR-Camera Calibration . . . 84
7.3 Optimal Shape For Sparse LiDAR Point Clouds . . . 85
7.3.1 Convex Shape Generation . . . 85
7.3.2 Edge Points Determination . . . 88
7.3.3 Shape Sensitivity . . . 88
7.3.4 Initial Cost Function . . . 90
7.4 Robust Shape for Real LiDAR Sensors . . . 91
7.4.1 Partial Illumination of Target . . . 91
7.4.2 Optimization for the Optimal Shape . . . 91
7.5 Global Pose and Feature Estimation . . . 92
7.6 Simulation Results . . . 93
7.7 Experimental Results . . . 96
7.7.1 Quantitative Experimental Results in M-Air . . . 96
7.7.2 Qualitative Experimental Results and Target Partial Illumination . . . 97

7.8 Conclusion . . . 97

Part III Motion Planning System for Legged Robots 101

8 Informable Multi-Objective and Multi-Directional RRT* System for Robot Path Planning . . . 101

8.1 Introduction and Contributions . . . 101
8.2 Related Work . . . 104
8.2.1 Common Path Planners . . . 105
8.2.2 Car-Pooling and Ride-Sharing . . . 105
8.2.3 Multi-objective Path Planners and Traveling Salesman Problem . . . 106
8.3 Informable Multi-Objecticve and Multi-Directional RRT* . . . 107
8.3.1 Standard RRT* Algorithm . . . 107
8.3.2 Multi-objective and Multi-directional RRT* on Graphs . . . 107
8.3.3 Discussion of Informability . . . 110
8.4 Enhanced Cheapest Insertion and Genetic Algorithm . . . 111
8.4.1 Relaxed Traveling Salesman Problem . . . 111
8.4.2 Graph Definitions and Connectivity . . . 111
8.4.3 Enhanced Cheapest Insertion Algorithm . . . 111
8.4.4 Genetic Algorithm . . . 113
8.4.5 Discussion of Time Complexity . . . 115
8.5 Experimental Results . . . 115
8.6 Conclusion and Future Work . . . 117

9 Efficient Anytime CLF Reactive Planning System for a Bipedal Robot on Undulating Terrain . . . 119

9.1 Introduction . . . 119
9.2 Related Work and Contributions . . . 120
9.2.1 Sampling-Based Motion Planning . . . 121
9.2.2 Optimization-based Planning and DARPA Subterranean Challenge . . . 121
9.2.3 Reactive Planning . . . 122
9.2.4 Contributions . . . 123
9.3 Construction of a Control Lyapunov Function . . . 124
9.3.1 Lyapunov and Control-Lyapunov Functions . . . 125
9.3.2 Redesign of CLF Proposed in the Literature . . . 126
9.3.3 State Representation . . . 127
9.3.4 Construction of Control Lyapunov Function . . . 128
9.3.5 Closed-form Solution . . . 130
9.3.6 Qualitative Analysis of the Closed-loop Trajectories . . . 131
9.4 Omnidirectional CLF-RRT* . . . 132
9.4.1 Standard RRT* Algorithm . . . 132
9.4.2 Omnidirectional CLF-RRT* Algorithm . . . 133
9.5 Reactive Planning System . . . 137
9.5.1 Elements of the Overall Planning System . . . 138

9.5.2 Planning Thread . . . 138
9.5.3 Reactive Thread . . . 142
9.6 Simulation and Experimental Results . . . 143
9.6.1 Angular Linear Inverted Pendulum (ALIP) Robot with Simulated Challenging Outdoor Terrains and Indoor Cluttered Scenes . . . 144
9.6.2 Validation of Control Command Feasibility via a Whole-body Cassie Simulator . . . 145
9.6.3 Perception Suite Design and Hardware System Integration . . . 146
9.6.4 Software System Integration for Real-time Deployment . . . 146
9.6.5 Full Autonomy Experiments with Cassie Blue . . . 147
9.6.6 Experiment Discussion . . . 149
9.7 Conclusion and Future Work . . . 149

Part IV Conclusion and Future Directions 151

10 Conclusion and Contributions . . . 151

10.1 Summary of Dissertation and Contributions . . . 151

11 Future Work . . . 153

11.1 Extension of Autonomy System . . . 153
11.2 Global Solver for Extrinsic Calibration of LiDAR and Camera . . . 153
11.2.1 LiDAR and Camera Simulators . . . 154
11.2.2 3D Points Triangulation from 2D Image Features with Geometry Constraints 155
11.2.3 Globally Optimal Solution of Extrinsic Parameters via Point-to-Point Correspondences . . . 157
11.2.4 Simulation Results . . . 159
11.3 Light-Weight Topological Map and Localization for Robotics Navigation . . . 160
11.3.1 Construction of Topological Map via Voronoi Graph . . . 160
11.3.2 Localization into Topological Map . . . 161

Appendices . . . 163

A.1 Proof of Uniqueness of Similarity Transformation . . . 163
A.2 Lagrangian Duality Relaxation for P2P Distance on $\mathrm{SE}(3)$ . . . 169
A.3 Observations on Global Convergence and Convexity . . . 171
B.1 ECI-Gen Traveling Salesman Problem (TSP) Solver Benchmarking . . . 174

Bibliography . . . 180

LIST OF FIGURES

FIGURE

1.1 Cassie Blue and Digit robot. 2
1.2 Proposed autonomy stack for Cassie Blue. 3
1.3 Roadmap of dissertation. 6

2.1 Computer-Aided design (CAD) of sensor suite. 13
2.2 Adjustable mounts for cameras and Light Detection and Ranging (LiDAR) sensors. . . 14
2.3 Material for sensor suite. 14
2.4 Sensor suite with Cassie Blue. 15

3.1 Building a semantic map for autonomous navigation. 17
3.2 Irregularity of LiDAR point clouds. 18
3.3 Comparison of un-calibrated and calibrated systems. 20
3.4 Proposed method to estimate LiDAR vertices. 21
3.5 Intersection over Union (IoU) of image corners and projected LiDAR vertices. 24
3.6 Refinement of camera corners. 26
3.7 Visual depiction of validation data. 29
3.8 Qualitative validation results. 29

4.1 Unifying intrinsic LiDAR calibration. 33
4.2 Illustration of the basis' of a ring plane. 38
4.3 Oriented tetrahedron and ring plane. 39
4.4 Illustration of a degenerate case. 40
4.5 Illustration of collections of LiDAR returns of the target from a spinning LiDAR. Different colors stand for different collections. 42
4.6 Point-to-Plane (P2P) cost improvement insensitive to condition number of basis. . . . 47
4.7 Four target placement as a tetrahedron. 48
4.8 Improvement of Velodyne LiDAR. 49
4.9 Improvement of solid-state LiDAR. 50

5.1 System diagram for automatic intrinsic and extrinsic calibration. 54

6.1 Visualization of LiDARTags of two different sizes in full point cloud scan. 58
6.2 Unstructured nature of LiDAR point cloud. 61
6.3 Design of LiDARTag. 62

6.4 Diagram of LiDARTag system. . . . 63
6.5 Intermediate steps of LiDARTag system 64
6.6 Initial state of cluster. . . . 67
6.7 Coordinate system of LiDARTag. . . . 68
6.8 Proposed method to estimate LiDARTag's pose. . . . 69
6.9 Four possible rotations of LiDARTag. . . . 72
6.10 Pose estimation of tag at 2 and 16 meters. . . . 75
6.11 Detection in cluttered laboratory and spacious outdoor environment. . . . 78

7.1 Illustration of vertex and pose estimation using proposed optimal target shape at 30 meters. . . . 82
7.2 Optimization process for determining optimal target shape. . . . 85
7.3 Equation (7.5) versus convexity. . . . 87
7.4 Illustration of math symbols. . . . 88
7.5 Shape sensitivity under rotation. . . . 90
7.6 Partial illumination and rotation of candidate shape. . . . 92
7.7 Resulting optimal shape from (7.16) in arbitrary units. . . . 93
7.8 Pose definition and illustration of template fitting. . . . 94
7.9 Simulation results of pose and vertex estimation at various distances. . . . 94
7.10 Simulation results with target placed at various distances. . . . 95
7.11 Perception sensors for experiments. . . . 97
7.12 Experimental setup for optimal target shape. . . . 98
7.13 Experimental results of optimal shape at various distances. . . . 98
7.14 Experimental results of partially illuminated target at various distances. . . . 99

8.1 Illustration of informable multi-objective and multi-directional RRT* (IMOMD-RRT*) system evaluated on OpenStreetMap of Chicago. . . . 102
8.2 System of proposed IMOMD-RRT* . . . 103
8.3 Illustration of acyclic graph and cyclic graphs with intriangularity inequality. . . . 106
8.4 Illustration of tree expansion and connection nodes of tree. . . . 108
8.5 Illustration of better connection nodes resulting in better path. . . . 109
8.6 Illustration of rewired path of pseudo-destinations. . . . 110
8.7 Illustration of insertion cost. . . . 112
8.8 Illustration of mutation and crossover in Genetic Algorithm. . . . 114
8.9 Quantitative and qualitative results for OSM of Seattle. . . . 117
8.10 Providing prior knowledge to the proposed IMOMD-RRT* system to avoid bug traps. 117

9.1 Cassie Blue autonomously traversing the Wave Field (at night) via the proposed reactive planning system. . . . 120
9.2 Deficiency of existing CLFs. . . . 124
9.3 Illustration of robot pose-centric polar representation. . . . 125
9.4 Explanation of signs in (9.3). . . . 126
9.5 Distance to target and penalty on yaw motion in (9.9) both affect closed-loop behavior. 131
9.6 Closed-loop trajectories generated by Control Lyapunov Function (CLF) in (9.5) vary as function as initial relative heading to target. . . . 132

9.7 Summary of proposed reactive planning system. 137
9.8 Elevation map built online while Cassie autonomously traverses Wave Field. 138
9.9 Simulated scenes and results obtained with proposed reactive planning system. 139
9.10 Simulation of C++-implementation of reactive planner on full-dynamic model of Cassie. 140
9.11 Computer-Aided design (CAD) of sensor suite. 141
9.12 Sensor suite with Cassie Blue. 142
9.13 Illustration of how various processes in overall autonomy system are distributed and their computation frequencies. 143
9.14 Autonomy experimental results on Wave Field. 144
9.15 Control commands sent to Cassie Blue. 145
9.16 Trajectories of autonomous experiments on first floor of Ford Robotics Building. . . . 146
9.17 Autonomy experimental results on second floor of Ford Robotics Building. 147

11.1 Illustration of LiDAR Simulator. 154
11.2 Camera Simulator. 155
11.3 Illustration of Parameters in Grunert Algorithm. 156
11.4 Illustration of Distance between Rays and Estimated Vertices. 157
11.5 Qualitative Results of Grunert Algorithm and Proposed Method. 158
11.6 Quantitative Results of Grunert Algorithm and Proposed Method. 159
11.7 Voronoi graph of second floor of FRB. 161
11.8 Topological maps of second floor of FRB. 162

A.1 Figure taken from [1] to illustrate experimental setup. 171
A.2 Proposed method for 3D registration problems. 172
A.3 f vs s for calibration parameters. 173

B.1 Validation of ECI-Gen TSP solver on small complete graphs with triangularity constraints. 176
B.2 Validation of ECI-Gen TSP solver on large complete graphs with triangularity constraints. 176
B.3 Validation of ECI-Gen TSP solver on small complete graphs without triangularity constraints. 177
B.4 Validation of ECI-Gen TSP solver on large complete graphs without triangularity constraints. 177
B.5 Validation of ECI-Gen TSP solver on small incomplete graphs with triangularity constraints. 178
B.6 Validation of ECI-Gen TSP solver on large incomplete graphs with triangularity constraints. 178
B.7 Validation of ECI-Gen TSP solver on small incomplete graphs without triangularity constraints. 179
B.8 Validation of ECI-Gen TSP solver on large incomplete graphs without triangularity constraints. 179

LIST OF TABLES

TABLE

3.1 Fitting and validation results for proposed calibration algorithm. 27
3.2 Summary of validation data. 28

4.1 Validation data for various calibration methods on Velodyne LiDAR. 51
4.2 Consistency of proposed methods. 52
4.3 Effect of number of cluster K on spinning LiDAR. 52
4.4 Experimental results of a solid-state LiDAR. 53

6.1 Decoding accuracy of Reproducing Kernel Hilbert Space (RKHS) method and pose accuracy. 74
6.2 Computation time for each step for indoors and outdoors. 76
6.3 Numbers of rejected clusters in each step in different scenes. 77
6.4 Comparison of different methods to compute double sum in (6.18). 77
6.5 Numbers of false positive rejection. 79

7.1 Pose and vertex accuracy of simulation results at various distances. 99
7.2 Pose and vertex accuracy of experimental results at various distances. 100

8.1 Quantitative results of the proposed IMOMD-RRT* system on large maps. 116

9.1 The default values of parameters. 130

LIST OF APPENDICES

A Global Unifying Intrinsic Calibration on Lie Group for Spinning LiDAR and Solid-state LiDAR **163**

B Informable Multi-Objective and Multi-Directional RRT* System for Robot Path Planning **174**

LIST OF ALGORITHMS

ALGORITHM

1 Proposed Global Optimizer for $\mathrm{Sim}(3)$ 45

2 $\mathcal{T} = (V, E) \leftarrow$ Omnidirectional CLF RRT* 134
3 $n_{\text{parent}} \leftarrow$ ChooseParent($\mathcal{N}_T, n_{\text{nearest}}, n_{\text{new}}$) 136
4 $\mathcal{T} \leftarrow$ ReWire($\mathcal{T}, \mathcal{N}_F, n_{\text{min}}, n_{\text{new}}$) 136

5 Voronoi Graph Construction 161

LIST OF ACRONYMS

IMU Inertial Measurement Unit

LiDAR Light Detection and Ranging

GPS Global Positioning System

GPU Graphics Processing Unit

LiPo Lithium Polymer

CAD Computer-Aided design

TBB Threading Building Blocks library

ROS Robot Operating System

UDP User Datagram Protocol

AR Augmented Reality

DoF Degrees of Freedom

ToF Time-of-Flight

FoV Field of View

CoM Center of Mass

SOTA State-of-the-Art

SLAM Simultaneous Localization and Mapping

IoU Intersection over Union

PnP Perspective-n-Point

P2P Point-to-Plane

PIP Point-In-Polygon

RANSAC Random Sample Consensus

SVD Singular Value Decomposition

RMS Root-Mean-Square

RKHS Reproducing Kernel Hilbert Space

MMA Method of Moving Asymptotes

RMSE Root-Mean-Square-Error

LQR Linear Quadratic Regulator

SoS Sum of Squares

PCA Principal Components Analysis

SVM Support Vector Machine

QCQP Quadratically Constrained Quadratic Program

QP Quadratic Program

SDP Semidefinite Programming

RRT Rapidly Exploring Random Tree

RRT* Rapidly Exploring Random Tree Star

CLF Control Lyapunov Function

CBF Control Barrier Function

OPA Optical Phased Array

MEMS Micro-Electro-Mechanical System

FSM Finite-State Machine

ALIP Angular Linear Inverted Pendulum

InEKF Invariant Extended Kalman Filter

MLM Multi-Layer Map

ICRA IEEE International Conference on Robotics and Automation

DARPA SubT DARPA Subterranean Challenge

MPC Model Predictive Control

FRB Ford Robotics Building

TSP Traveling Salesman Problem

R-TSP Relaxed Traveling Salesman Problem

HD High-Definition

DP Dynamic Programming

OSM OpenStreetMap

ABSTRACT

The term "mobile robot" implies a machine or system that is capable of moving within and performing purposeful behaviors on the real world. Legged robots are types of mobile robots with articulated limbs to provide locomotion, and they have the potential to access unstructured environments, which will eventually enable legged robots to aid in package delivery, terrain exploration, search and rescue, disaster relief, and one day even become assistants in our homes. Two-legged robots (i.e., bipedal robots) that are tall and slim can easily adapt to structures built for humans (narrow staircases or passages). In addition to assisting the elderly or people with physical disabilities, bipedal robots and their two-legged cousins, exoskeletons, are improving productivity. Moreover, knowledge acquired about bipedal robots, exoskeletons, prosthetics, and human biomechanics is transferable and becomes positive reinforcement. It is these characteristics that have motivated me to work on autonomy of bipedal robots.

The capabilities of bipedal robots have yet to be unleashed in the service of society due to a number of challenges. One key challenge involves problems in sensor fusion, pose estimation, and smooth motion planning, as these aspects are critical for a bipedal robot to autonomously walk toward a distant destination and to smoothly avoid dynamic obstacles detected through various sensing modalities, while maintaining its stability.

This dissertation seeks to develop a full autonomy system that allows bipedal robots to 1) acquire multi-modal data from a calibrated perception suite; 2) estimate their poses in textureless environments; 3) detect and avoid dynamic obstacles; 4) traverse unexplored, unstructured environments and undulating terrains; and 5) perform point-to-point topometric navigation. All the research presented in this dissertation focuses on advancing the state of the art in mobile robot autonomy. To ensure the practicality of our work, we have evaluated all of our algorithms on Cassie Blue.

To allow Cassie to perceive its surroundings, we built a perception suite that includes Light Detection and Ranging (LiDAR) sensors, cameras, and Inertial Measurement Units (IMUs). To ensure the obtained measurements are valid and meaningful, sensor fusion and calibration are required for most modern perception and navigation systems deployed on autonomous robots. Accordingly, we develop an automatic pipeline for both LiDAR-camera extrinsic calibration and intrinsic calibration for LiDARs. The resulting calibrated system achieves pixel-level error when projecting a LiDAR point cloud on the camera's image plane. Additionally, we propose a unifying

intrinsic calibration technique for both spinning and solid-state LiDARs. Our method reduces by 48.7% the error in our factory-calibrated LiDAR. The calibrated sensors mounted on the perception suite enable us to open up an important chapter in Cassie's autonomy.

Later, we develop the very first fiducial marker system for LiDAR point clouds — LiDARTag — to estimate a mobile robot's pose. Because LiDAR sensors are not affected by rapidly changing ambient lighting, LiDARTags can be used where cameras cannot, and the proposed fiducial marker can even operate in a completely dark environment. The LiDARTag system achieves millimeter translation error and a few degrees of rotation error, reaches 99.7% accuracy on decoding, and runs faster than 100 Hz. However, due to the quantization uncertainty and sparsity of LiDAR returns, the performance of the pose estimation when using a 32-beam Velodyne LiDAR is degraded when the target is farther than 12 meters. We therefore propose the concept of target shape optimization for enhancing pose and vertex estimation from LiDAR point clouds. Specifically, we design a target so that the edge points induced by LiDAR rings are "highly" sensitive to translation and rotation. This attenuates the deleterious effects of quantization uncertainty and sparsity of a target's LiDAR image. The resulting target shape is naturally asymmetric to reduce pose ambiguity. Moreover, we present a means that leverages target geometry to estimate the target vertices while globally estimating the pose. We formulate the pose estimation problem as a semidefinite program and modify a global convex solver to efficiently compute the target pose. With the enhanced target shape and solver, we achieve centimeter error in translation and a few degrees of error in rotation even when an occluded target of width 0.96 meter is placed 30 meters away.

Next, multi-objective or multi-destination path planning is crucial for mobile robotics applications such as mobility as a service, robotics inspection, and electric vehicle charging for long trips. We propose an anytime iterative system to concurrently solve the multi-objective path planning problem and determine the visiting order of destinations. The system is comprised of an anytime informable multi-objective and multi-directional RRT* algorithm to form a simple connected graph, and a proposed solver that consists of an enhanced cheapest insertion algorithm and a genetic algorithm to solve the relaxed traveling salesman problem in polynomial time.

Finally, we propose and experimentally demonstrate a reactive planning system for bipedal robots traversing unexplored, challenging terrains. The system consists of a low-frequency planning thread (5 Hz) to find an asymptotically optimal path and a high-frequency reactive thread (300 Hz) to accommodate robot deviation. The planning thread includes: a multi-layer local map to compute traversability for the robot on the terrain; an anytime omnidirectional Control Lyapunov Function (CLF) for use with a Rapidly Exploring Random Tree Star (RRT*) that generates a vector field for specifying motion between nodes; a sub-goal finder when the final goal is outside of the current map; and a finite-state machine to handle high-level mission decisions. The system also includes a reactive thread to obviate the non-smooth motions that arise with traditional RRT* algorithms

when performing path following. The reactive thread copes with robot deviation while eliminating non-smooth motions via a vector field (defined by a closed-loop feedback policy) that provides real-time control commands to the robot's gait controller as a function of instantaneous robot pose.

CHAPTER 1

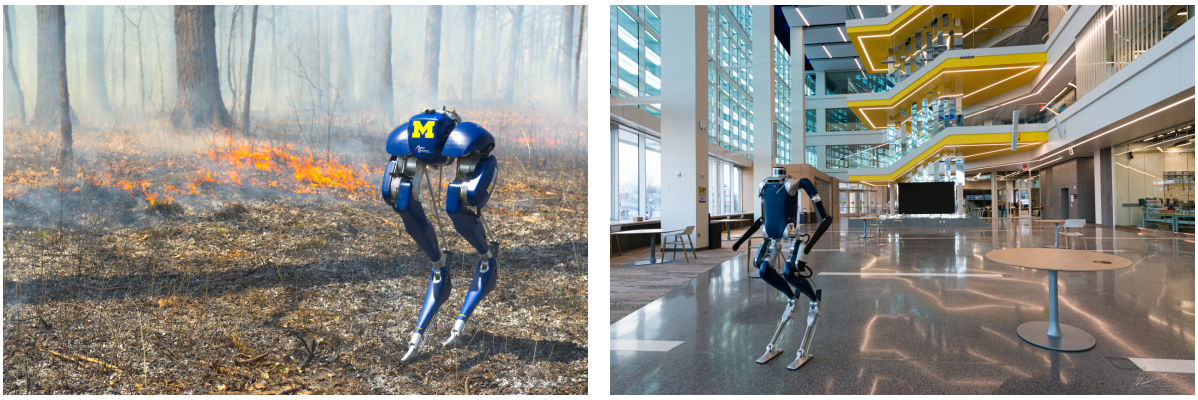
Introduction

1.1 Motivation

The term“mobile robot” implies a machine or system that is capable of moving around and performing purposeful behaviors in the real world. Legged robots are types of mobile robots with articulated limbs to provide locomotion. Having legs has been a huge advantage of legged robots to traverse various types of terrains — not accessible to typical wheeled robots. The importance of legs has been shown in evolutionary history [2]. For example, scorpions, the very first land-dwelling animals [3], opened a new chapter of history as they transitioned onto land with limbs. Over the past 440 million years, many land vertebrates including humans have evolved to keep legs to adapt to undulating and rocky terrains. In addition, such kind of limb morphology has enabled diverse and advanced motion such as jumping, and stepping over obstacles [4]. In other words, legs can step over isolated paths while wheels require a continuous path to travel. These capabilities allow legged robots to potentially assist package delivery, terrain exploration, search-and-rescue, and disaster relief, and one day even, become assistants in our homes.

Humans are unlike typical tetrapods. We walk bipedally upright with free hands for carrying and manipulating tools. Most existing factories, warehouses, and apartments are built for humans and accommodate narrow staircases or passages that are accessible to bipedal robots which are tall and slim but not to other types of legged robots that are short and wide. Furthermore, advances in bipedal robot technology can help human rehabilitation, specifically in regaining the ability to walk, using exoskeletons [5, 6] or powered prosthetics [7, 8]. In addition to assisting the elderly or people with physical disabilities, bipedal robots and their two-legged cousins, exoskeletons, are improving productivity. Moreover, knowledge acquired about bipedal robots, exoskeletons, prosthetics, and human biomechanics is transferable and becomes positive reinforcement. These characteristics have motivated me to work on autonomy of bipedal robots.

Despite the astounding potential of bipeds, full deployment of the capabilities of bipedal robots to autonomously accomplish tasks and to serve society has yet to be unleashed. Most bipeds are confined to controlled environments such as laboratories due to the high DoF, underactuation,



(a) (b)

Figure 1.1: The left shows Cassie Blue, one of Michigan's Cassie-series robots. The robot is shown participating in a controlled burn. It has 20 Degrees of Freedom (DoF), 10 actuators, joint encoders, and an Inertial Measurement Unit (IMU). The robot's serial number is 001. When standing up, Cassie is about one meter tall and its total mass is 31 kg. The right shows another bipedal robot, Digit, in our lab. Digit weighted 45 kg is essentially Cassie with an upper-body and two four-DoF arms. Additionally, Digit is equipped with five cameras and a LiDAR.

and complex dynamics, along with the unstructured environment in which they are instructed to navigate.

This dissertation seeks to develop a full autonomy system that allows bipedal robots to 1) acquire multi-modal data from a calibrated perception suite; 2) estimate their poses in textureless environments; 3) detect and avoid dynamic obstacles; 4) traverse unexplored, unstructured environments and undulating terrains; and 5) perform point-to-point topometric navigation. All the research presented in this dissertation focuses on advancing the state of the art in mobile robot autonomy. To ensure the practicality of our work, we have evaluated all of our algorithms on Cassie Blue (and eventually a Digit robot), as shown in Fig. 1.1.

1.2 Objectives

A number of challenges need to be solved to develop an autonomy system for mobile robots such as Cassie Blue, a bipedal robot with 20 DoF. These challenges include problems in controller design, robot pose estimation, data acquisition, sensor fusion and calibration, mapping, and motion planning, as shown in Fig. 1.2.

First, to allow a bipedal robot to perceive surroundings, we design a perception suite to rigidly house different types of sensors such as IMUs, cameras, and LiDARs (see Chapter 2). Sensor calibration, which can be intrinsic or extrinsic, is an essential step in achieving the measurement accuracy required for modern perception and navigation systems deployed on autonomous robots. While determining such a transformation between sensors is not considered glamorous in any sense of the word, it is nonetheless crucial for modern autonomous systems [9–20]. Indeed, an error of

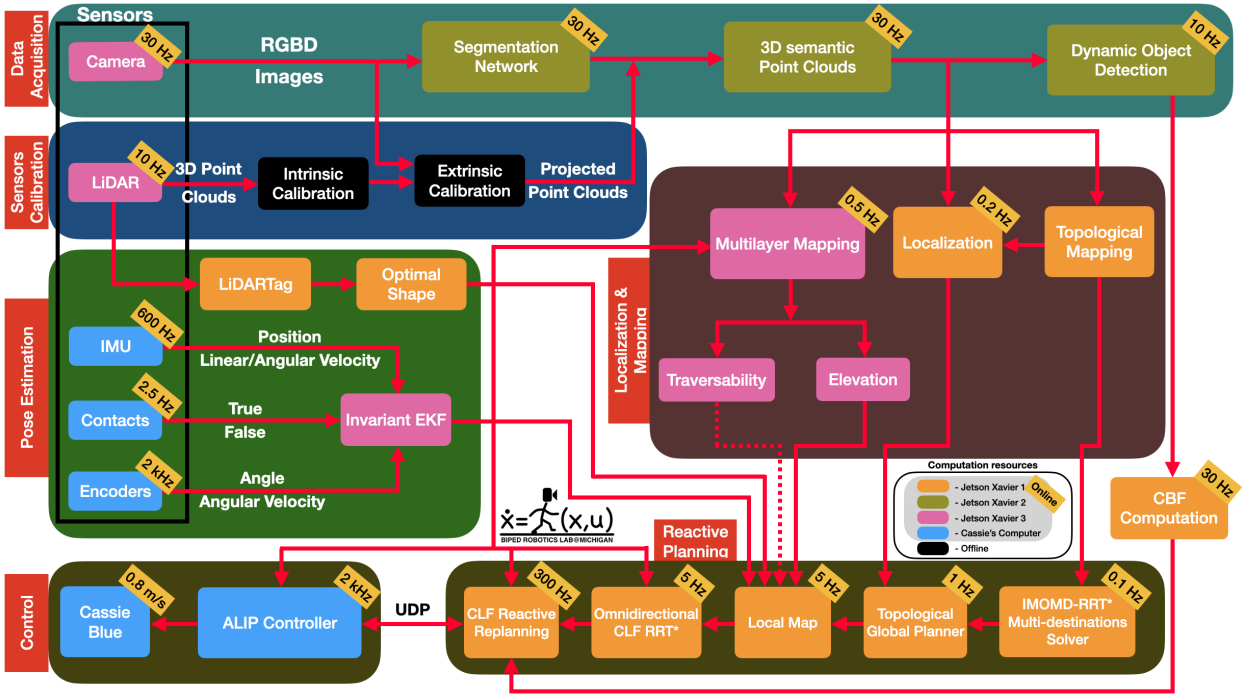


Figure 1.2: The proposed autonomy stack for Cassie Blue. This figure illustrates how the various processes in the overall autonomy system are distributed and their computation frequencies. The larger boxes indicate various modules such as Data Acquisition, Mapping, Planning, and Control. The smaller boxes are colored according to the processor that runs them.

a few degrees in rotation or a few percent in translation can lead to 20 cm reprojection errors at a distance of 5 m when overlaying a LiDAR point cloud on a camera image. The biggest impediments to determining the transformation accurately are the relative sparsity of LiDAR point clouds and systematic errors in their distance measurements. In Chapter 3, we propose (1) the use of targets of known dimension and geometry to ameliorate target pose estimation in face of the quantization and systematic errors inherent in a LiDAR image of a target, (2) a fitting method for the LiDAR to monocular camera transformation that avoids the tedious task of target edge extraction from the point cloud.

During the process of extrinsic calibration, we observe that our LiDAR had a noticeable amount of systematic errors. We then turn our attention to LiDAR intrinsic calibration [21–29]. Intrinsic calibration of a sensor is the process of ensuring that obtained measurements are meaningful and valid. In Chapter 4, we propose a unifying view of calibration for different types of LiDARs (spinning vs. solid-state, for example). Intrinsic calibration models for spinning LiDARs have been based on hypothesized physical mechanisms, resulting in anywhere from three to ten parameters to be estimated from data, while no phenomenological models have yet been proposed for solid-state LiDARs. Instead of going down that road, we propose to abstract away from the physics of a LiDAR type (spinning vs. solid-state, for example) and focus on the point cloud's spatial geometry

generated by the sensor. By modeling the calibration parameters as an element of a matrix Lie Group, we achieve a unifying view of calibration for different types of LiDARs. We further prove mathematically that the proposed model is well-constrained (has a unique answer) given four appropriately orientated targets. The proof provides a guideline for target positioning in the form of a tetrahedron. Moreover, an existing Semidefinite Programming (SDP) global solver for $\mathrm{SE}(3)$ can be modified to efficiently compute the optimal calibration parameters. For spinning LiDARs, we show with experimental data that the proposed matrix Lie Group model performs equally well as physics-based models in terms of reducing the point-to-plane distance while being more robust to noise. We also demonstrate the proposed method is able to calibrate a solid-state LiDAR and is robust to noise.

Periodic intrinsic and extrinsic (re-)calibrations are essential for modern perception and navigation systems deployed on autonomous robots. In Chapter 5, we propose an automatic calibration pipeline that utilizes synchronized image-based and LiDAR-based fiducial markers to speed up the calibration process.

Image-based fiducial markers [30–38] are useful in problems such as robot pose estimation, object tracking in cluttered or textureless environments, camera (and multi-sensor) calibration tasks, and vision-based Simultaneous Localization and Mapping (SLAM). The State-of-the-Art (SOTA) fiducial marker detection algorithms rely on the consistency of the ambient lighting. In Chapter 6, we introduce the very first fiducial marker for LiDARs, called LiDARTag [39, 40]. The proposed system runs in real-time and can process data at 100 Hz, which is faster than the currently available LiDAR sensor frequencies. Because LiDAR sensors are not affected by rapidly changing ambient lighting, LiDARTags can be used where cameras cannot, and the proposed fiducial marker can even operate in a completely dark environment. In addition, the LiDARTag nicely complements and is compatible with existing visual fiducial markers, such as AprilTags [30, 31], allowing for efficient multi-sensor fusion and calibration tasks. We further propose a concept of minimizing a fitting error between a point cloud and the marker's template to estimate the marker's pose. The proposed method achieves millimeter error in translation and a few degrees in rotation. Due to the sparsity of LiDAR returns, the point cloud is lifted to a continuous function in a Reproducing Kernel Hilbert Space (RKHS) where the inner product can be used to determine a marker's ID. The experimental results, verified by a motion capture system, confirm that the proposed method can reliably provide a tag's pose and unique ID code. The rejection of false positives is validated on the Google Cartographer [41] indoor dataset and the Honda H3D outdoor dataset [42].

Targets are essential in problems such as object tracking in cluttered or textureless environments, camera (and multi-sensor) calibration tasks, and SLAM. Target shapes for these tasks typically are symmetric (square, rectangular, or circular) and work well for structured, dense sensor data such as pixel arrays (i.e., image). However, symmetric shapes lead to pose ambiguity when using

sparse sensor data such as LiDAR point clouds and suffer from the quantization uncertainty of the LiDAR. In Chapter 7, we introduce the concept of optimizing target shape to remove pose ambiguity for LiDAR point clouds [43]. A target is designed to induce large gradients at edge points under rotation and translation relative to the LiDAR to ameliorate the quantization uncertainty associated with point cloud sparseness. Moreover, given a target shape, we present a means that leverages the target's geometry to estimate the target's vertices while globally estimating the pose. Both the simulation and the experimental results (verified by a motion capture system) confirm that by using the optimal shape and the global solver, we achieve centimeter error in translation and a few degrees in rotation even when a partially illuminated target is placed 30 meters away.

Multi-objective or multi-destination path planning is crucial for mobile robotics applications such as mobility as a service, car-pooling, and electric vehicle charging for long trips. In Chapter 8, we propose an anytime iterative system to concurrently solve the multi-objective path planning problem, and determine the visiting order of destinations. The system is comprised of an anytime informable multi-objective and multi-directional RRT* algorithm to form a union of undirected weighted graph and a weighted-insertion breadth-first search algorithm to solve the relaxed traveling salesman problem in polynomial time. Moreover, an ordered list of inspection waypoints is often provided for robotics inspection so that the robot can preferentially examine certain equipment or area of interests. We show that the proposed system can inherently incorporate such knowledge and substantially reduce the computation time and the number of explored nodes. The proposed anytime system is evaluated on various large-complex graphs built for real-world applications.

After the path planner provides a sequence of intermediate goals to reach a distant destination, we deploy a novel reactive planning system to smoothly approach the intermediate goals. In Chapter 9, we propose and experimentally demonstrate a reactive planning system [44] for bipedal robots on unexplored, challenging terrains. The system consists of a low-frequency planning thread (5 Hz) to find an asymptotically optimal path and a high-frequency reactive thread (300 Hz) to accommodate robot deviation. The planning thread includes: a multi-layer local map to compute traversability for the robot on the terrain; an anytime omnidirectional CLF for use with a Rapidly Exploring Random Tree Star (RRT*) that generates a vector field for specifying motion between nodes; a sub-goal finder when the final goal is outside of the current map; and a finite-state machine to handle high-level mission decisions. The system also includes a reactive thread to obviate the non-smooth motions that arise with traditional RRT* algorithms when performing path following. The reactive thread copes with robot deviation while eliminating non-smooth motions via a vector field (defined by a closed-loop feedback policy) that provides real-time control commands to the robot's gait controller as a function of instantaneous robot pose. This reactive planning system allows Cassie Blue to autonomously avoid obstacles, complete high-level missions, and traverse sinusoidally varying terrains.

1.3 Dissertation Roadmap

Figure 1.3 shows the roadmap of this dissertation. The first part is comprised of a sensor suite design (Chapter 2) and sensor calibration techniques (Chapter 3 to Chapter 5). The second part describes robot pose estimation via LiDAR-based fiducial markers (Chapter 6 and Chapter 7). The last part explains robotics reactive planning system to guide the robot to a distant goal (Chapter 9).

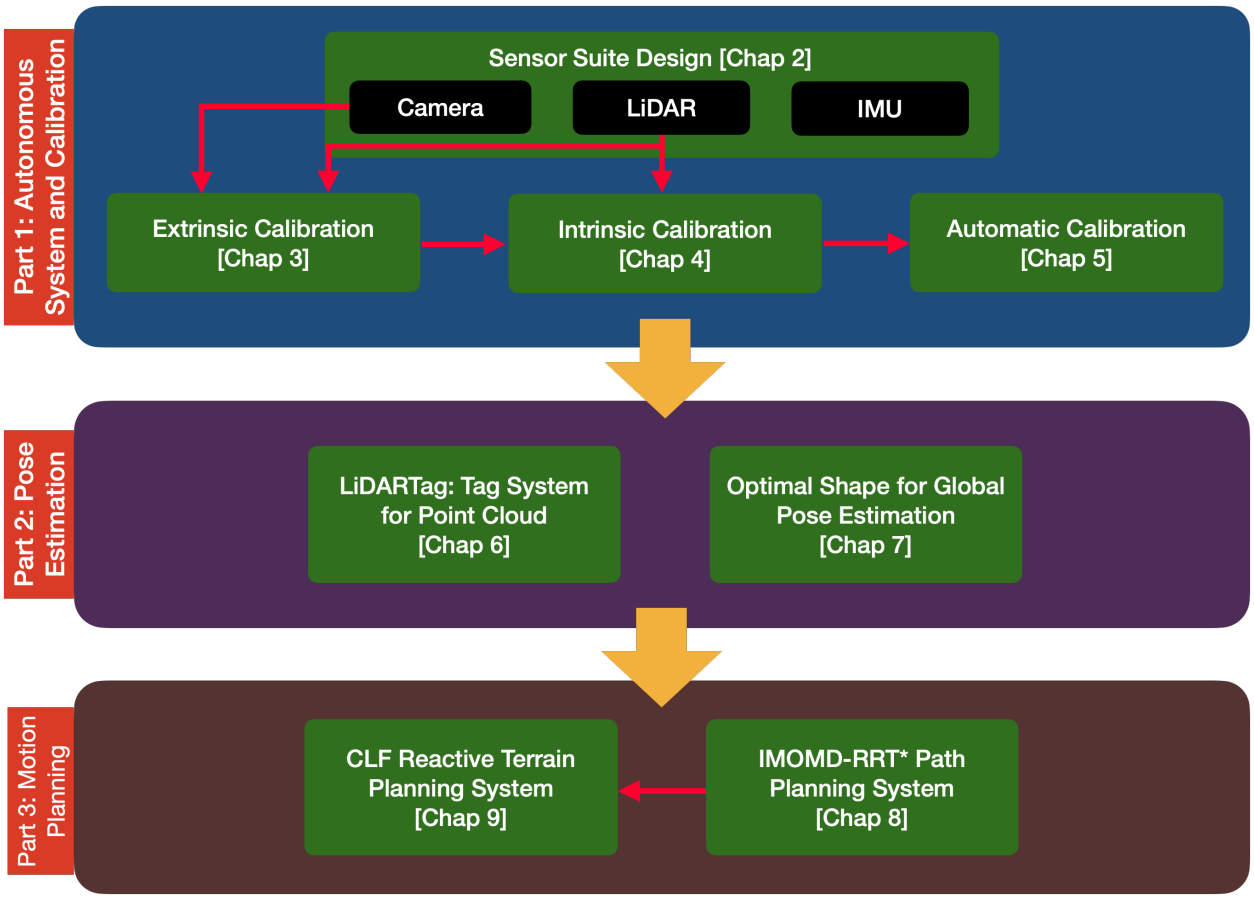


Figure 1.3: Illustration of the roadmap of the dissertation.

1.4 Contributions

In summary, this dissertation makes the following contributions:

1. We design a sensor suite [45] that rigidly houses several types of sensors, and the suite enables autonomy of Cassie Blue (Chapter 2). We conduct several autonomy experiments with the torso and the videos are uploaded to [46–49].

2. We propose a SOTA algorithm to calibrate LiDARs and cameras (Chapter 3). When evaluated against a state-of-the-art baseline, the proposed algorithm results in, on average, more than a 50% reduction in LiDAR-to-camera projection error and a 70% reduction in its variance. Two other benefits of our method are: (1) it does not require the estimation of a target normal vector from an inherently noisy point cloud; and (2) it also obviates the identification of edge points and their association with specific sides of a target.

3. We present a universal method for LiDAR intrinsic calibration (Chapter 4) that abstracts away the physics of a LiDAR type (spinning head vs. solid-state, for example), and focuses instead on the spatial geometry of the point cloud generated by the sensor. The calibration parameter becomes an element of $\mathrm{Sim}(3)$, a matrix Lie group. We mathematically prove that given four targets with appropriate orientations, the proposed model is well-constrained (i.e., a unique answer exists). We show how to profitably apply efficient, globally convergent algorithms for $\mathrm{SE}(3)$ to determine a solution to our problem in $\mathrm{Sim}(3)$. The resulting algorithm is evaluated in simulation for a solid-state LiDAR and simulation and experiment for a spinning LiDAR. The P2P distance of the validation scene for the spinning LiDAR is reduced 44.7% and 68.6% in simulation and experiment, respectively. The P2P distance of the validation scene for the solid-state LiDAR is reduced by 48.7%. Both simulation and experiments show that the proposed method can serve as a generic model for intrinsic calibration of both spinning and solid-state LiDARs.

4. We develop the very first fiducial marker for LiDAR point clouds (Chapter 6). The proposed fiducial tag system runs in real-time (faster than 100 Hz) while it can handle a full scan of raw point cloud from the employed *32-Beam Velodyne ULTRA Puck LiDAR* (up to 120,000 points per scan). The LiDARTag pose estimation block deploys an L_1-inspired cost function. It achieves millimeter error in translation and a few degrees of error in rotation compared to ground truth data collected by a motion capture system with 30 motion capture cameras. The sparse LiDAR returns on a LiDARTag are lifted to a continuous function in a Reproducing Kernel Hilbert Space (RKHS) where the inner product is used to determine the marker's ID, and this method achieves 99.7% accuracy. The presented fiducial marker system can also be used with cameras and has been successfully used for calibration [9, 21].

5. We formulate the concept of optimizing target shape to remove pose ambiguity for LiDAR point clouds (Chapter 7). A target is designed to induce large gradients at edge points under rotation and translation relative to the LiDAR to ameliorate the quantization uncertainty associated with point cloud sparseness. The resulting shape is asymmetric to elude pose ambiguity. Additionally, we propose a means that utilizes target shape to jointly estimate target vertices and pose. Because the cost function of the proposed method can be formulated as an SDP, the target's pose and vertices can be globally estimated with an open-source solver. In the simulation, we validate that the optimal shape with the global solver achieves centimeter error in translation and a few degrees of error in rotation when the targets are at a distance of 30 meters and partially illuminated. In addition, we conduct experimental evaluations where the ground truth data are provided by a motion capture system, and achieve results similar to the simulation.

6. We develop an anytime iterative system (Chapter 8) to provide paths between multiple objectives and to determine the visiting order of destinations; moreover, the system should be informable meaning it can accommodate prior knowledge of intermediate nodes, if available. The proposed system consists of two components: **1)** an anytime informable multi-objective and multi-directional RRT* (IMOMD-RRT*) algorithm to form a connected weighted-undirected graph, and **2)** a relaxed TSP solver that consists of an enhanced version of the cheapest insertion algorithm and a genetic algorithm, which together we call ECI-Gen. The proposed system is evaluated on large-complex graphs built for real-world driving applications, such as the OpenStreetMap (OSM) of Chicago containing $866,089$ nodes and $1,038,414$ edges.

7. We present a novel reactive planning system (Chapter 9) that consists of a 5-Hz planning thread to guide a robot to a distant goal and a 300-Hz CLF-based reactive thread to cope with robot deviations. In simulation, we evaluate the reactive planning system on ten challenging outdoor terrains and cluttered indoor scenes, and perform fully autonomy experiments with Cassie Blue on sinusoidally varying terrain. The planning thread uses a multi-layer, robot-centric local map to compute traversability for challenging terrains, a sub-goal finder, and a finite-state machine to choose a sub-goal location as well as omnidirectional CLF RRT* to find an asymptotically optimal path for Cassie to walk in a traversable area. The omnidirectional CLF RRT* utilizes the newly proposed CLF as the steering function and the distance measure on the CLF manifold in the RRT* algorithm. Both the proposed CLF and the distance measure have a closed-form solution. The distance measure nicely accounts for the inherent "features" of Cassie-series robots, such as high-cost for lateral movement. The robot's motion in the reactive thread is generated by a vector field depending on a closed-loop feedback policy

providing control commands to the robot in real-time as a function of instantaneous robot pose. In this manner, problems typically encountered by waypoint-following and pathway-tracking strategies when transitioning between waypoints or pathways (unsmooth motion, sudden turning, and abrupt acceleration) are resolved.

8. We open-source all the CAD models, datasets, and related software, see Chapter 1.5.

1.5 Open-source Software

This section provides a list of open-source packages released with this dissertation:

1. CAD Model for Perception Suite:
   https://github.com/UMich-BipedLab/torso_design_for_cassie

2. Extrinsic Calibration for 3D LiDAR and Camera:
   https://github.com/UMich-BipedLab/extrinsic_lidar_camera_calibration

3. Global Unifying LiDAR Intrinsic Calibration:
   https://github.com/UMich-BipedLab/LiDAR_intrinsic_calibration

4. Global Solver for Lie Group on $\mathrm{Sim}(3)$:
   https://github.com/UMich-BipedLab/global_sim3_solver

5. LiDARTag: A Real-Time Fiducial Tag System for Point Clouds:
   https://github.com/UMich-BipedLab/LiDARTag

6. LiDARTag Messages for ROS:
   https://github.com/UMich-BipedLab/LiDARTag_msgs

7. Global Pose Estimation of Optimal Shape:
   https://github.com/UMich-BipedLab/global_pose_estimation_for_optimal_shape

8. Optimal Shape Generation for LiDAR Point Clouds:
   https://github.com/UMich-BipedLab/optimal_shape_generation

9. Informable Multi-Objective and Multi-Directional RRT* System for Robot Path Planning:
   https://github.com/UMich-BipedLab/IMOMD-RRTStar

10. Efficient Anytime CLF Reactive Planning System for a Bipedal Robot on Undulating Terrain:
    https://github.com/UMich-BipedLab/CLF_reactive_planning_system

11. Planning Messages for ROS:
    https://github.com/UMich-BipedLab/planner_msgs

12. Customized Multi-layer Elevation Map:
    https://github.com/UMich-BipedLab/customized_grid_map

13. LiDAR Simulator in MATLAB:
    https://github.com/UMich-BipedLab/lidar_simulator

14. Automatic Extrinsic Calibration for 3D LiDAR and Camera:
    https://github.com/UMich-BipedLab/automatic_lidar_camera_calibration

15. AprilTag ROS:
    https://github.com/UMich-BipedLab/AprilTag_ROS

16. AprilTag Messages for ROS:
    https://github.com/UMich-BipedLab/AprilTag_msgs

17. Synchronization of LiDARTag and AprilTag:
    https://github.com/UMich-BipedLab/sync_lidartag_apriltag

18. Synchronized AprilTag Messages and LiDARTag Messages:
    https://github.com/UMich-BipedLab/alignment_msgs

19. Useful MATLAB functions:
    https://github.com/UMich-BipedLab/matlab_utils

Part I

Autonomous System for Legged Robots and Sensor Calibration

CHAPTER 2

Design of Perception Suite for Legged Robots

2.1 Introduction and Sensors

Taking sensor measurements and extracting meaningful information are essential to acquire knowledge about the robot itself and its environment so that it is able to take proper action. Sensors can be roughly classified into two categories based on their functionalities. Proprioceptive sensors such as Inertial Measurement Unit (IMU) or encoders, measure the internal states of the robot (speed, joint angles, or force) at higher frequencies (100 Hz to 2 kHz). Exteroceptive sensors such as Global Positioning System (GPS), Light Detection and Ranging (LiDAR), camera, or radar, collect information from surroundings (distance measurements, position, color, or reflectivity, intensity) at lower frequencies (5-30 Hz). We briefly introduce a few common sensors below.

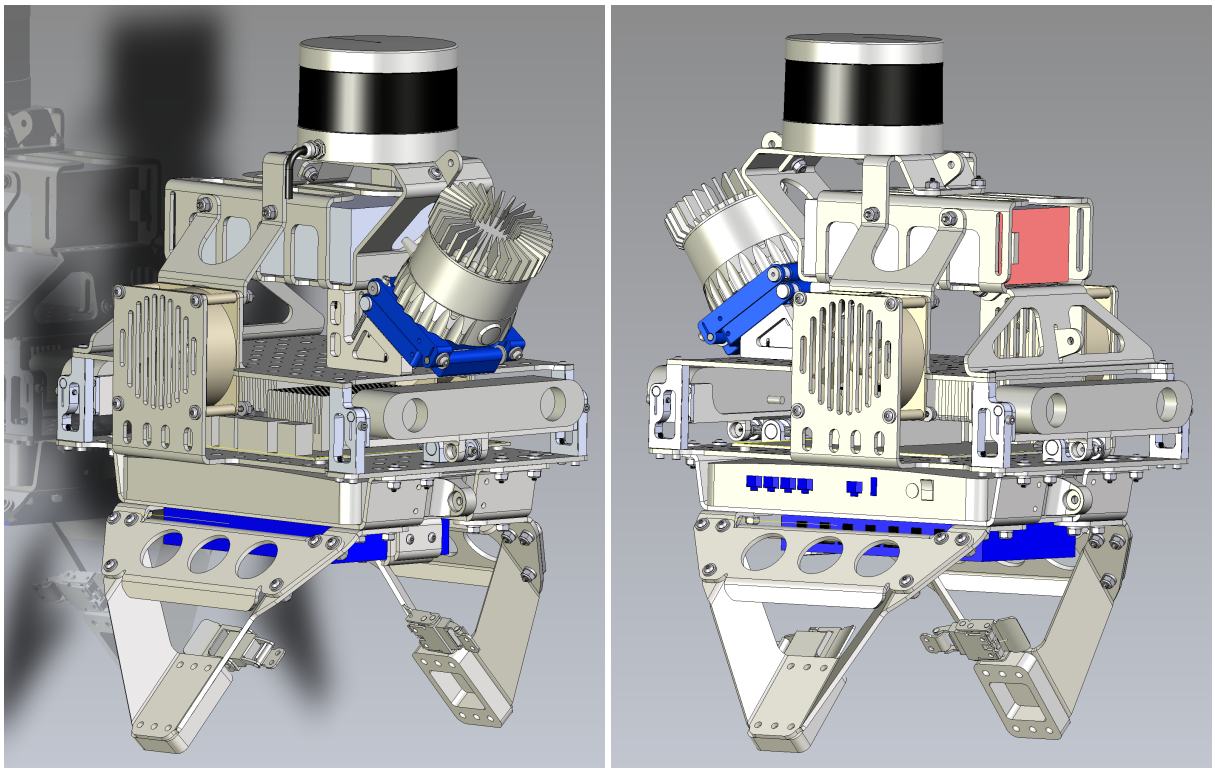
An IMU is a multi-axis sensor often composed of gyroscopes, accelerometers, and magnetometers. Gyroscopes and accelerometers measure angular velocity and linear velocity, respectively. A magnetometer measures magnetic field strength. Therefore, a nine DoF IMU includes 3-axis gyroscopes, 3-axis accelerometers, and 3-axis magnetometers. All measurements are taken in sensor or body coordinates. A typical IMU runs at least 400 Hz regardless of robot states or ambient temperature.

An RGB-camera captures color information as pixel arrays (i.e., an image), and assists the robot to semantically understand surroundings. However, the quality of an image is sensitive to ambient lighting. Thus, cameras are not suitable in the dark. A standard RGB-D camera produces images with dense depth estimated by features extracted from the environment. Typically, the estimated

depth is not accurate over eight meters and will suffer in textureless or dark environments. Currently, basic cameras provide many more data points (pixels) for a given surface size at a given distance than even high-end LiDARs. However, cameras rely on the consistency of the ambient lighting and provide less accurate depth estimation than LiDARs.

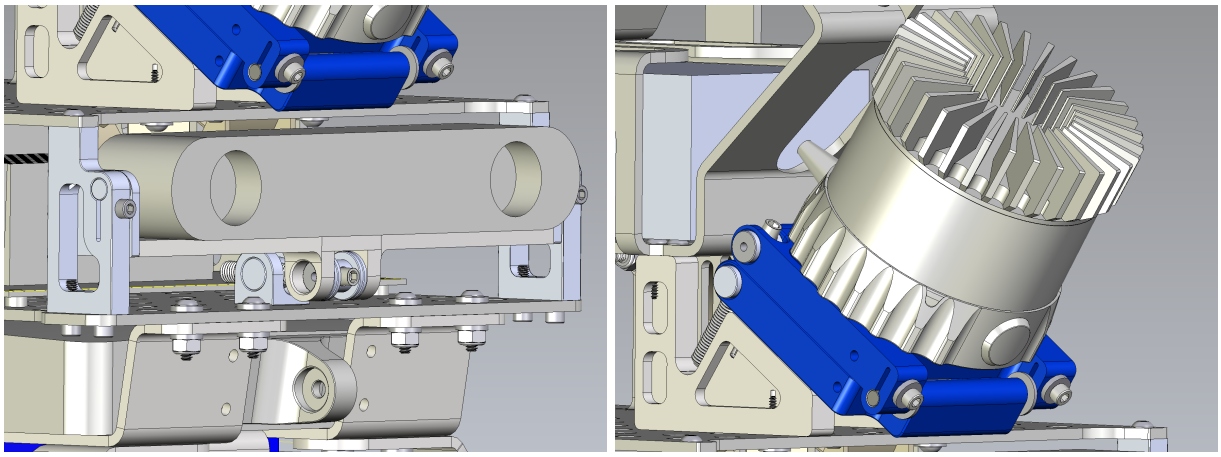
LiDARs, on the other hand, measure accurate and long-range (0.1-200 m) distance by estimating Time-of-Flight (ToF) of laser beams through a clocking system on a circuit board, resulting in a sparse but accurate point cloud. Because of the nature of LiDAR sensors, rapidly changing ambient lighting will not affect measurements of a LiDAR; hence, a LiDAR can facilitate the robot in perceiving surroundings at night, even in completely dark environments. There are different types of LiDARs (spinning LiDARs and solid-state LiDARs) coming to the market which we discuss in Chapter 4.

GPS sensors are receivers with antennas that utilize a satellite-based navigation system consisting of 24 satellites in orbit around the earth. A GPS is able to compute its location by trilateration if it receives three or more satellites. In other words, GPS sensors suffer in places with poor satellite signals such as indoor environments, populated urban cities, or forests. A typical GPS runs at 15 Hz and provides position with accuracy ± 5 meters, velocity, and timing information.



(a) (b)

Figure 2.1: The CAD design of the sensor suite. The left shows the front view of the suite and the right shows its back.

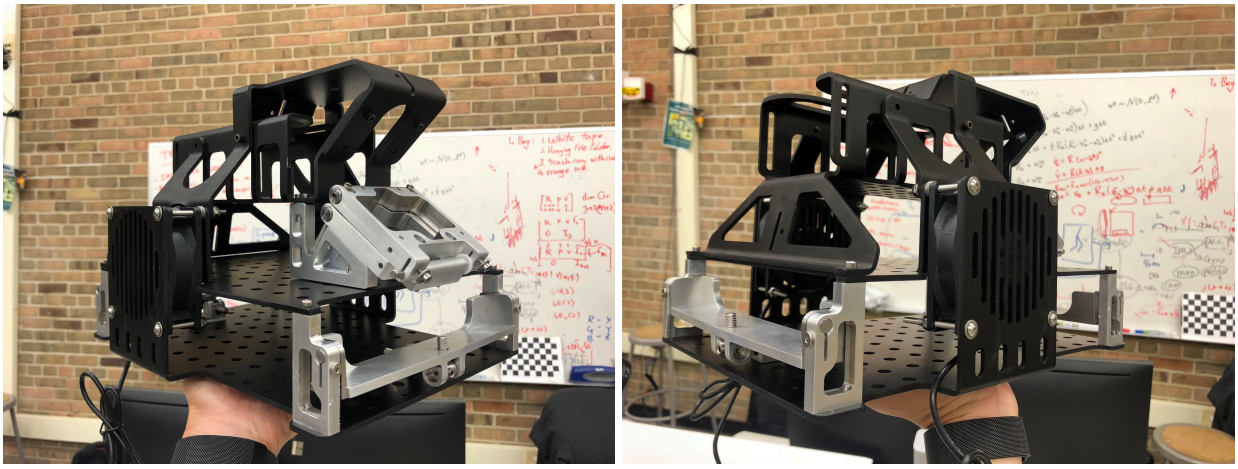


(a) (b)

Figure 2.2: The left shows the adjustable mount for the camera and the right shows the adjustable mount for the LiDAR.

2.2 Perception Suite

Due to these different inherent properties of various types of sensors in order to enable Cassie Blue to perceive surroundings under different lighting conditions and environments, we design a perception suite that consists of two RGB-D cameras with built-in IMUs, and two LiDARs. In addition, two fans are used to vacillate two Jetson AGX Xaviers with Graphics Processing Units (GPUs). Furthermore, a router, a USB hub, and an internet switch are utilized for communication from users and Cassie to the perception suite. Finally, a 12-volt Lithium Polymer (LiPo) battery powers up all the sensors. Figure 2.1 shows the design of the full sensor suite and the step files are available at [45].



(a) (b)

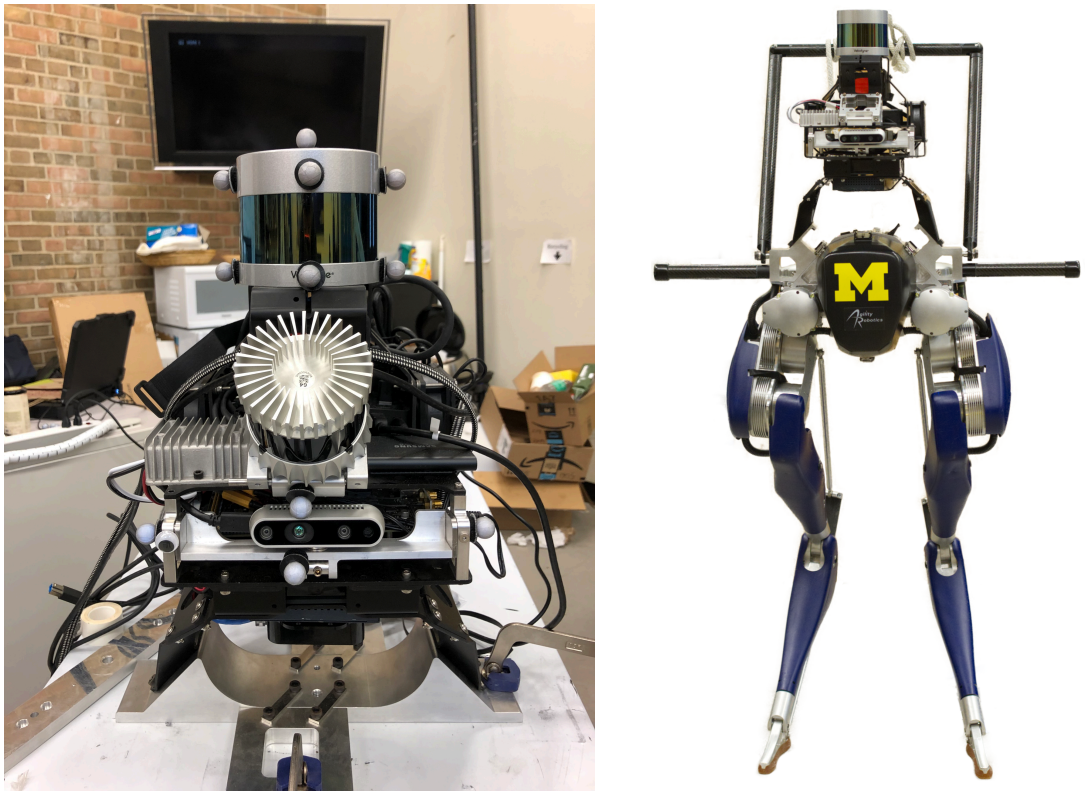
Figure 2.3: The frame of the sensor suite is made of 1/8 5052 sheet metal (black) to firmly support the weights of the sensors and the sensor mounts are made of 2024 aluminium (silver).

Remark 1. *The four sensors are placed and oriented differently to gather information around the robot. Additionally, we design adjustable mounts for the cameras and the tilted LiDAR, as shown in Fig. 2.2 and are still able to screw-lock in place.*

The frame of the suite is made of 1/8 5052 sheet metal (black) to firmly support the weights of the sensors, and the sensor mounts are made of 2024 aluminium (silver), as shown in Fig. 2.3. The weight of the suite without sensors is about 7 kilograms. We use an industrial graded router and an industrial graded internet switch to ensure stable connections among sensors[1]. To fully utilize all the sensors, sensor calibrations have to be done.

Sensor calibration, which can be intrinsic or extrinsic, is an essential step to achieve the measurement accuracy required for modern perception and navigation systems deployed on autonomous robots. Extrinsic calibration is a process of obtaining rigid-body transformations between sensors (see Chapter 3). Intrinsic calibration of a sensor, on the other hand, is the process of ensuring that obtained measurements are meaningful and valid (see Chapter 4).

[1]We previously used commercial routers and switches, but the connections between sensors often broke due to the large roll motion of Cassie Blue while walking or the extreme heat in summer.



(a) (b)

Figure 2.4: The left shows the sensor suite with different sensors, and the right shows the suite mounted on Cassie Blue.

CHAPTER 3

Improvements to Target-Based 3D LiDAR to Camera Calibration

3.1 Introduction and Related Work

The desire to produce 3D-semantic maps [50] with our Cassie-series bipedal robot [51] has motivated us to fuse 3D-LiDAR and RGB-D monocular camera data for autonomous navigation [46]. Indeed, by mapping spatial LiDAR points onto a segmented and labeled camera image, one can associate the label of a pixel (or a region about it) to the LiDAR point as shown in Fig. 3.1. An error of a few degrees in rotation or a few percent in translation in the estimated rigid-body transformation between LiDAR and camera can lead to 20 cm reprojection errors at a distance of 5 m when overlaying a LiDAR point cloud on a camera image. Such errors will lead to navigation errors.

In this chapter, we assume that the intrinsic calibration of the two sensors has already been done [22] and focus on obtaining the rigid-body transformation, i.e., rotation matrix and translation, between a LiDAR and camera. This is a well studied problem with a rich literature that can be roughly divided into methods that do not require targets: [52–57] and those that do: [12–20, 58, 59]. While many of the existing target-based methods may work well, in practice, they require many manual steps, such as carefully measuring different parts of the targets, parsing edge-points, and edge-points pairing inspection. Our method avoids these manual steps and is applicable to planar polygonal targets, such as checkerboards, triangles, and diamonds.

In target-based methods, one seeks to estimate a set of target features (e.g., edge lines, normal vectors, vertices) in the LiDAR's point cloud and the camera's image plane. If "enough" independent correspondences can be made, the LiDAR to camera transformation can be found by Perspective-n-Point (PnP) as in [60], that is, through an optimization problem of the form

$$(R_C^{L*}, T_C^{L*}) = \arg\min_{(R,T)} \sum_i \operatorname{dist}(P(H_L^C(X_i)), Y_i), \tag{3.1}$$

where X_i are the (homogeneous) coordinates of the LiDAR features, Y_i are the coordinates of the camera features, P is the often-called "projection map", H_L^C is the (homogeneous representation of) the LiDAR-frame to camera-frame transformation with rotation matrix R_C^L and translation T_C^L, and

Figure 3.1: Building a semantic map for autonomous navigation. Once the transformation from LiDAR to camera is known, it is possible to fuse the LiDAR and RGB-camera information to build a 3D-map that is annotated with semantic information. The lower right shows a single camera image; its segmentation with semantic labels is shown in the upper right. Synchronized LiDAR points are mapped onto the camera image, associated to a semantic label, and then re-projected to a local frame [61] along with the semantic labels to create a 3D semantically-labeled map [50], shown in the upper left. The sidewalk is marked in white, the grass in yellow-green, and trees in dark green. In the lower left, Cassie is using the map to plan a path around the North Campus Wave Field, while staying on the sidewalk.

dist is a distance or error measure.

3.1.1 Rough Overview of the Most Common Target-based Approaches

The works closest to ours are [10, 11]. Each of these works has noted that rotating a square target so that it presents itself as a diamond can help to remove pose ambiguity due to the spacing of the ring lines; in particular, see Fig. 2 in [10] and Fig. 1 in [11]. More generally, we recommend the literature overview in [10] for a recent, succinct survey of LiDAR to camera calibration.

The two most common sets of features in the area of target-based calibration are (a) the 3D-coordinates of the vertices of a rectangular or triangular planar target, and (b) the normal vector to the plane of the target and the lines connecting the vertices in the plane of the target. Mathematically, these two sets of data are equivalent: knowing one of them allows the determination of the other. In practice, focusing on (b) leads to use of the SVD to find the normal vector and more broadly to least squares line fitting problems [11], while (a) opens up other perspectives, as highlighted in [10].

Figure 3.2(a) shows a 3D view of 25 scans from a factory-calibrated *32-Beam Velodyne ULTRA Puck LiDAR* on a diamond shaped planar target. The zoom provided in Fig. 3.2(b) shows that some of the rings are poorly calibrated, with a "noise level" that is three times higher than the manufacturer's specification. On average, the rings are close to the specification ($\pm$ five cm at less than 50 m) of the LiDAR, which means a maximum ten centimeters difference between points on the same ring.

Systematic errors in the distance (or depth) measurement affect the estimation of the target's

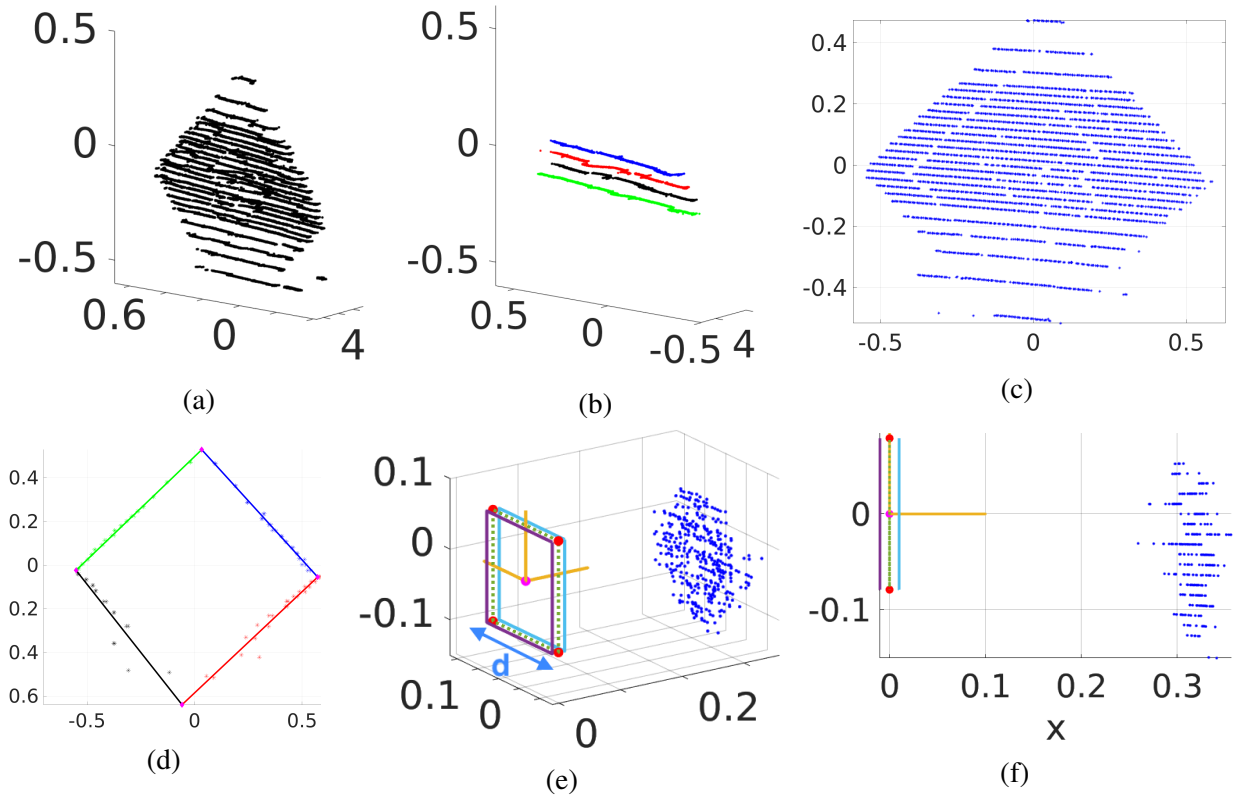


Figure 3.2: Units are meters. (a) Twenty five LiDAR scans of a planar target. The point cloud is roughly 7 cm thick. (b) The "noise" in the point cloud is not random. A zoom for four of the rings (13, 15, 17, 19) is typical and shows systematic errors in distance. (c) LiDAR points orthogonally projected to the plane defined by the normal. (d) Example edge points selected to regress a line via *Random Sample Consensus (RANSAC)*. (e) Target reference frame and real point cloud data. The dotted blue square is the reference target; its vertices are denoted $\{\bar{X}_i\}_{i=1}^{4}$. (f) A side-$(x-z)$-view highlighting the non-zero thickness of a typical point cloud. These figures and all others in the chapter are of sufficient resolution that they can be blown up to see detail.

centroid, which is commonly used to determine the translation of the target with respect to the LiDAR, and the target's *normal vector*, which is used to define the plane of the target, as shown in Fig. 3.2(c). Subsequently, the point cloud is orthogonally projected to this plane and the line boundaries of the target are found by performing *RANSAC* on the appropriate set of ring edges; see Fig. 3.2(d). The lines along the target's boundaries then define its vertices in the plane, which for later reference, we note are not constrained to be compatible with the target's geometry.

Once the vertices in the plane of the target have been determined, then knowledge of the target's normal vector allows the vertices to be lifted back to the coordinates of the point cloud. This process may be repeated for multiple scans of a target, aggregates of multiple scans of a target, or several targets, leading to a list of target vertices $\{X_i\}_{i=1}^{4n}$, where n is the number of target poses.

The target is typically designed so that the vertices are easily distinguishable in the camera image. Denoting their corresponding coordinates in the image plane by $\{Y_i\}_{i=1}^{4n}$ completes the data required for the conceptual fitting problem in (3.1). While the cost to be minimized is nonlinear and

non-convex, this is typically not a problem because CAD data can provide an adequate initial guess for local solvers, such as Levenberg-Marquardt; see [11] for example.

3.1.2 Our Contributions

Our contributions can be summarized as follows.

(a) We introduce a novel method for estimating the rigid-body transform from target to LiDAR, H_T^L. For the point cloud pulled back to the origin of the LiDAR frame via the current estimate of the transformation, the cost is defined as the sum of the distance of a point to a 3D-reference target of known geometry[1] in the LiDAR frame, for those points landing outside of the reference target, and zero otherwise. We use an L_1-inspired norm in this work to define distance. Consistent with [10], we find that this is less sensitive to outliers than L_2-line fitting to edge points[2] using *RANSAC* [11].

(b) In the above, by using the entire target point cloud when estimating the vertices, we avoid all together the delicate task of identifying the edge points and parsing them into individual edges of the target, where small numbers of points on some of the edges accentuate the quantization effects due to sparsity in a LiDAR point cloud.

(c) We provide a round-robin validation study to compare our approach to a state-of-the-art method, namely [11]. A novel feature of our validation study is the use of the camera image as a proxy for ground truth; in the context of 3D semantic mapping, this makes sense. In addition to a standard PnP method [60] for estimating the rigid-body transformation from LiDAR to camera, we also investigate maximizing IoU to estimate the rigid-body transformation. Our algorithms are validated on various numbers of targets.

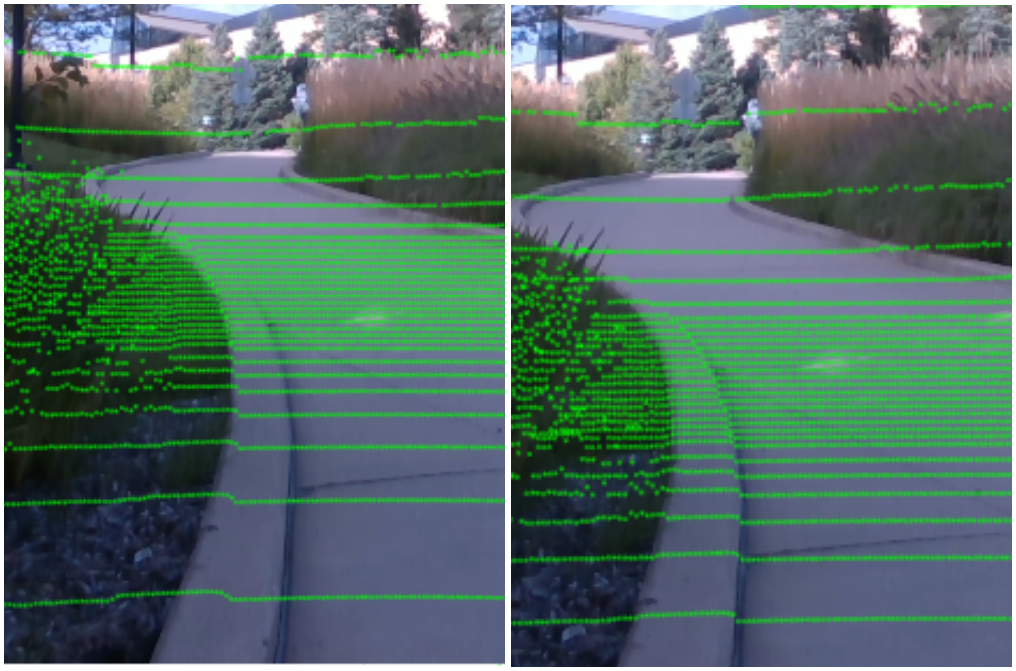
(d) Code for our method, our implementation of the baseline, and all the data sets used in this chapter are released as open source; see [62].

3.2 Finding the LiDAR Target Vertices

We assume that each target is planar, square, and rotated in the frame of the LiDAR by roughly 45° to form a diamond as in Fig. 3.2(a). As indicated in [10, 11], placing the targets so that the rings of the LiDAR run parallel to its edges leads to ambiguity in the vertical position due to the spacing

[1] It has been given non-zero volume.

[2] Defined as the left-right end points of each LiDAR ring landing on the target.



(a) (b)

Figure 3.3: (a) shows that a calibration result is not usable if it has a few degrees of rotation error and a few percent of translation error. (b) shows good alignment of a LiDAR point cloud projected onto a camera image.

of the rings. We assume that standard methods have been used to automatically isolate the target's point cloud [39] and speak no further about it.

We take as a target's features its four vertices, with their coordinates in the LiDAR frame denoted $\{X_i\}_{i=1}^4$; when useful, we abuse notation and pass from ordinary coordinates to homogeneous coordinates without noting the distinction. The vertices $\{X_i\}_{i=1}^4$ are of course not directly observable in the point cloud. This section will provide a new method for estimating their coordinates in the frame of the LiDAR using an L_1-like norm.

3.2.1 Remarks on LiDAR Point Clouds

LiDARs have dense regions and sparse regions along the z-axis, as shown in Fig. 3.2(c). For a 32-beam Velodyne Ultra Puck, we estimate the resolution along the z-axis is 0.33° and 1.36° in the dense and sparse regions, respectively. A point's distance resolution along the y-axis is roughly 0.1°. The quantization error could be roughly computed from: $d \sin \theta$ when a point is at d meter away. As a result, the quantization error could get quite large if one place the tag at a far distance. Figure 3.3(a) shows that a 1° of rotation error on each axis and 5% error in translation can significantly degrade a calibration result. As noted in [10, 11], it is essential to place a target at an appropriate angle so that known geometry can mitigate quantization error in the y- and z-axes.

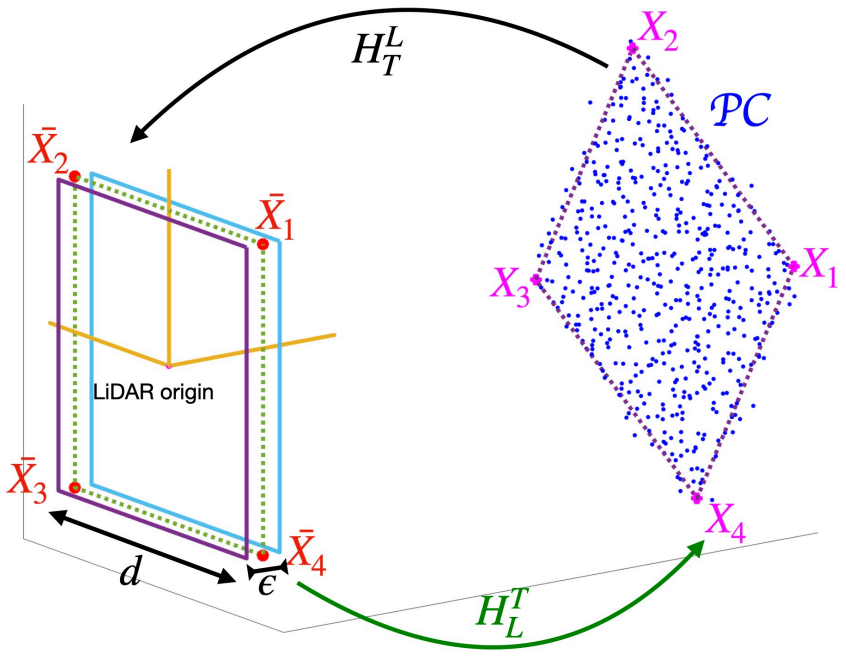


Figure 3.4: This conceptual figure illustrates the proposed method to estimate LiDAR vertices. The target's reference in the LiDAR frame is defined by $(\bar{X}_1, \cdots, \bar{X}_4)$ with depth ϵ. The rigid-body transformation H_T^L (black arrow) pulls back the target's $\mathcal{PC}$ to a target reference about the LiDAR origin. The actual vertices $(X_1, \cdots, X_4)$ of the target are estimated by (3.5), using the inverse transformation H_L^T (green arrow).

3.2.2 New Method for Determining Target Vertices

Let $\mathcal{PC}$ denote the LiDAR point cloud of the target and let the collection of 3D points be $\mathcal{X}_i$ so that $\mathcal{PC} = \{\mathcal{X}_i\}_{i=1}^N$, where N is the number of points on the target. For the extrinsic calibration problem, we need to estimate the target vertices in the LiDAR frame. As in Fig. 3.4, let H_L^T be the rigid-body transformation from a reference target in the LiDAR frame with vertices $\{\bar{X}_i\}_{i=1}^4$, onto the point cloud. We use the inverse transform $H_T^L := (H_L^T)^{-1}$ to pull back the target's point cloud to the origin of the LiDAR and measure the error there.

For $a \geqslant 0$ and $\lambda \in \mathbb{R}$, define

$$c(\lambda, a) := \begin{cases} \min\{|\lambda - a|, |\lambda + a|\} & \text{if } |\lambda| > a \\ 0 & \text{otherwise} \end{cases}; \tag{3.2}$$

see also the "soft L_1 norm" in [10]. Let $\{\tilde{\mathcal{X}}_i\}_{i=1}^N := H_T^L(\mathcal{PC}) := \{H_T^L(\mathcal{X}_i)\}_{i=1}^N$ denote the pullback of the point cloud by H_T^L, and denote a point's Cartesian coordinates by $(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)$. The cost is defined as

$$C(H_T^L(\mathcal{PC})) := \sum_{i=1}^N c(\tilde{x}_i, \epsilon) + c(\tilde{y}_i, d/2) + c(\tilde{z}_i, d/2), \tag{3.3}$$

where d is determined by the size of the (square) target, and the only tuning parameter is $\epsilon > 0$, the thickness of the ideal target. The value of ϵ is based on the standard deviation of a target's return points to account for the noise level of the depth measurement; see Fig. 3.2(f).

We propose to determine the optimal rigid-body transformation, with rotation matrix R_T^L and translation vector T_T^L, by

$$H_T^{L*} := \underset{R_T^L, T_T^L}{\arg\min}\, C(H_T^L(\mathcal{PC})), \tag{3.4}$$

and to define the estimated target vertices by

$$X_i^* := H_L^{T*}(\bar{X}_i), \;\; 1 \leqslant i \leqslant 4. \tag{3.5}$$

Remark 2. *It is emphasized that the target vertices are being determined without extraction of edge points and their assignment to a side of the target. The correspondences of the estimated vertices with the physical top, bottom, and left or right sides of the target are not needed at this point.*

Remark 3. *The cost in* (3.3) *treats the target as a rectangular volume. To be clear, what we seek is a "best estimate" of the target vertices in the LiDAR frame and not the transformation itself. Our method is indirect because from the point cloud, we estimate a "best" LiDAR to target transformation, H_L^{T*}, and use it to define the vertices by* (3.5)*.*

3.3 Image Plane Corners and Correspondences with the LiDAR Vertices

For a given camera image of a LiDAR target, let $\{{}_CY_i\}_{i=1}^4$ denote the corners of the camera image. We assume that these have been obtained through the user's preferred method, such as corner detection [63–65], edge detection [66–68], or even manual selection. The resulting camera corners are used in both the proposed method and the baseline. This is not the hard part of the calibration problem. To achieve simple correspondences $X_i \leftrightarrow {}_CY_i$, the order of the indices of $\{X_i\}_{i=1}^4$ may need to be permuted; we use the vertical and horizontal positions to sort them; see [62].

Once we have the correspondences, the next step is to project the vertices of the LiDAR target, $\begin{bmatrix} x_i & y_i & z_i & 1 \end{bmatrix}^\mathsf{T} = X_i$, into the image coordinates. The standard relations are [69, 70]

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbb{1}_{3\times 3} \\ \mathbf{0}_{1\times 3} \end{bmatrix}^\mathsf{T} \begin{bmatrix} R_L^C & T_L^C \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{3.6}$$

$${}_LY_i = \begin{bmatrix} u & v & 1 \end{bmatrix}^\mathsf{T} = \begin{bmatrix} \frac{u'}{w'} & \frac{v'}{w'} & 1 \end{bmatrix}^\mathsf{T}, \tag{3.7}$$

where (3.6) includes the camera's intrinsic parameters and the extrinsic parameters (R_L^C, T_L^C) that we seek.

For later use, we combine (3.6) and (3.7) to define

$$\Pi\left(X_i; R_L^C, T_L^C\right) := {}_L Y_i, \tag{3.8}$$

the projection map from LiDAR coordinates to image coordinates. Note that it is a function of both the extrinsic variables and the LiDAR vertices.

3.4 Extrinsic Transformation Optimization

This section assumes the vertices of the target in the LiDAR frame and in the camera's image plane have been determined, along with their correspondences. The optimization for the extrinsic transformation can be formulated in a standard PnP problem: minimize Euclidean distance of the corresponding corners. We also propose maximizing the IoU of the corresponding projected polygons.

3.4.1 Euclidean distance

The standard PnP formulation is

$$\begin{aligned}\left(R_L^{C*}, T_L^{C*}\right) &:= \arg\min_{R,T} \sum_{i=1}^{4n} \left\|\Pi\left(X_i; R, T\right) - {}_C Y_i\right\|_2^2 \\ &= \arg\min_{R,T} \sum_{i=1}^{4n} \left\|{}_L Y_i - {}_C Y_i\right\|_2^2,\end{aligned} \tag{3.9}$$

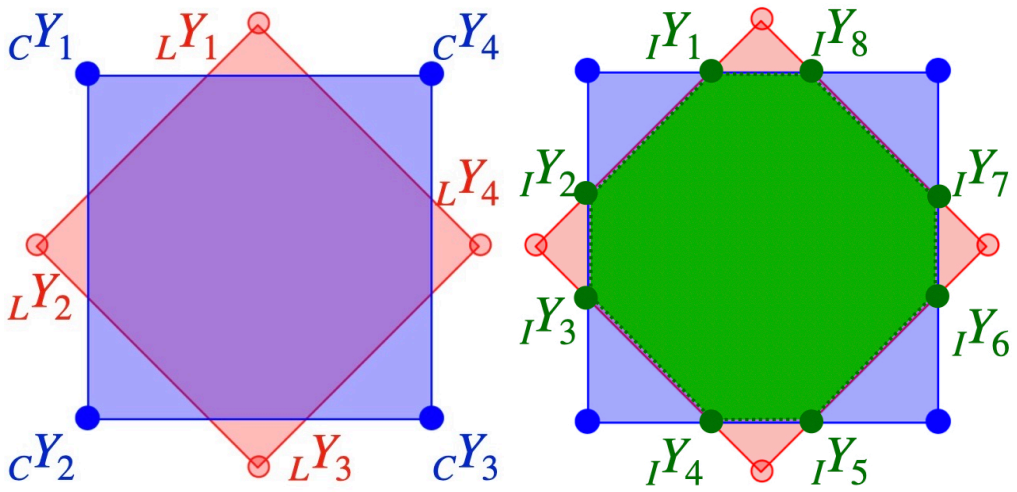
where ${}_C Y_i \in \mathbb{R}^2$ are the camera corners, ${}_L Y_i \in \mathbb{R}^2$ are defined in (3.8), and n is the number of target poses.

3.4.2 IoU optimization

For a given polygon, let $\mathcal{V} := \{Y_i | Y_i =: (x_i,\ y_i)\}_{i=1}^N$ be the coordinates of the N vertices, ordered counterclockwise. The area of the polygon is computed via the Shoelace algorithm [71, eq. (3.1)] [72],

$$\mathbf{A}(\mathcal{V}) = \mathbf{A}(Y_1, \cdots, Y_N) = \frac{1}{2}\left|\sum_{i=1}^{N} \det\begin{bmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{bmatrix}\right|, \tag{3.10}$$

where $x_{N+1} := x_1$ and $y_{N+1} := y_1$. If $\mathcal{V}$ is empty, the area is taken as zero. We now apply this basic area formula to propose an IoU-based cost function for extrinsic calibration.



(a) (b)

Figure 3.5: Two polygons in an image plane. (a) The red polygon ${}_L\mathcal{V} = \{{}_L Y_i\}_{i=1}^4$ represents the estimated vertices projected from the LiDAR frame to the image plane, while the blue polygon ${}_C\mathcal{V} = \{{}_C Y_i\}_{i=1}^4$ represents the vertices of the actual camera image of the target. (b) shows the intersection (marked in green) of the two polygons, with the vertices labeled as ${}_I\mathcal{V} = \{{}_I Y_i\}_{i=1}^8$.

Let ${}_L\mathcal{V} := \{{}_L Y_i\}_{i=1}^{4n}$ be the vertices of the estimated target polygons projected from the LiDAR frame to the camera frame as in (3.8), and let ${}_C\mathcal{V} := \{{}_C Y_i\}_{i=1}^{4n}$ be the vertices of the corresponding camera images of the targets, as in Fig. 3.5(a). If their intersection is nonempty, we define it as a polygon with known coordinates[3], denoted as ${}_I\mathcal{V} := \{{}_I Y_i\}_{i=1}^{M}$, where $M \geqslant 0$ is the number of intersection points; see Fig. 3.5(b). We sort the three sets of vertices of the polygons counterclockwise using Graham's scan algorithm, a method to find the convex hull of a finite set of points in a plane [73, 74]. The IoU of the two polygons is then

$$IoU({}_L\mathcal{V}, {}_C\mathcal{V}, {}_I\mathcal{V}) = \frac{\mathbf{A}({}_I\mathcal{V})}{\mathbf{A}({}_L\mathcal{V}) + \mathbf{A}({}_C\mathcal{V}) - \mathbf{A}({}_I\mathcal{V})}. \tag{3.11}$$

The resulting optimization problem is

$$\begin{aligned} \left(R_L^{C*}, T_L^{C*}\right) &:= \arg\max_{R,T} IoU({}_L\mathcal{V}, {}_C\mathcal{V}) \\ &= \arg\max_{R,T} IoU(\Pi\left(\{X_i\}_{i=1}^{4n}; R, T\right), {}_C\mathcal{V}), \end{aligned} \tag{3.12}$$

where (3.8) has been used to make the dependence on the rigid-body transformation explicit.

[3]The vertices of the polygon consist of the 2D corners and the 2D line intersections and thus can be computed efficiently; see Fig. 3.5(b)

3.5 Experimental Results

In this section, we extensively evaluate our proposed method on seven different scenes through a form of "cross-validation": in a round-robin fashion, we estimate an extrinsic transformation using data from one or more scenes and then evaluate it on the remaining scenes. The quantitative evaluation consists of computing pixel error per corner, where we take the image corners as ground truth. We also show qualitative validation results by projecting the LiDAR scans onto camera images; we include here as many as space allows, with more scenes and larger images available at [62].

3.5.1 Data Collection

The scenes include both outdoor and indoor settings. Each scene includes two targets, one approximately 80.5 cm square and the other approximately 15.8 cm square, with the smaller target placed closer to the camera-LiDAR pair. We use an *Intel RealSense Depth Camera D435* and a *32-Beam Velodyne ULTRA Puck LiDAR*, mounted on an in-house designed torso for a Cassie-series bipedal robot [46]. From the CAD file, the camera is roughly 20 cm below the LiDAR and 10 cm in front of it. The angle of the camera is adjustable. Here, its "pitch", in the LiDAR frame, is approximately zero.

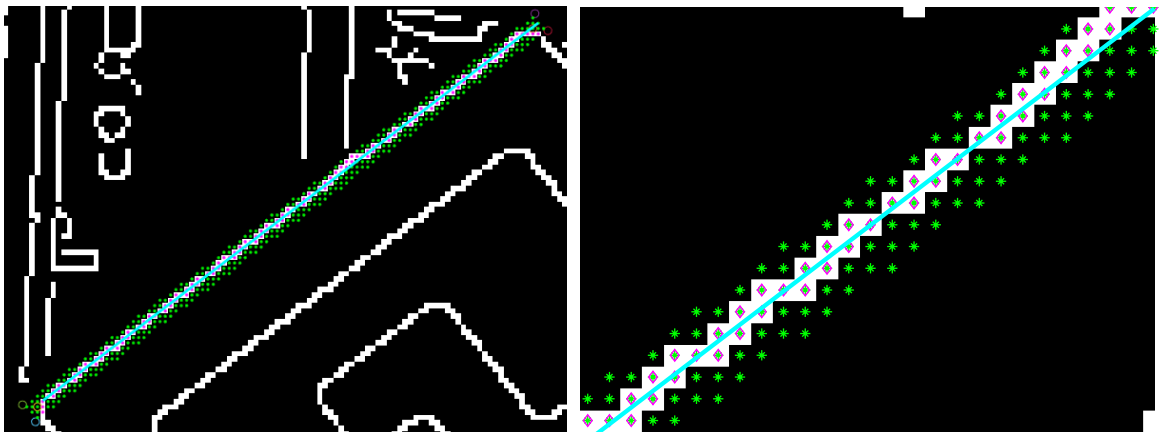
A scan consists of the points collected in one revolution of the LiDAR's 32 beams. The data corresponding to a single beam is also called a ring. For each scene, we collect approximately 10 s of synchronized data, resulting in approximately 100 pairs of scans and images.

For each target, five consecutive pairs of LiDAR scans and camera images are selected as a data set. For each data set, we apply two methods to estimate the vertices of the targets, a baseline and the method in Sec. 3.2.2. We then apply both PnP and IoU optimizations to find the rigid-body transformation, see Sec. 3.4.

3.5.2 Baseline Implementation for LiDAR Vertices

As a baseline, we use the method in [11]. Because an open-source implementation was not released, we built our own, attempting to be as faithful as possible to the described method.

For each scan, the large and small targets are individually extracted from background [39]. For each target and group of five scans, we compute the extracted point cloud's centroid and center the point cloud about the origin. Singular Value Decomposition (SVD) is then used to find the target's normal and to orthogonally project the point cloud onto the plane defined by the normal and the centroid. For each scan, and for each ring hitting the target, the left and right end points of the ring are selected and then associated with one of the four edges of the target. Lines are fitted to each collection of edge points using least-squares and *RANSAC*. The vertices are obtained as the



(a) (b)

Figure 3.6: (a) shows the result of edge detection. (b) shows the interior pixels (marked in green) of a bounding box given two clicked corners. The edge points and the edge line (as found by *RANSAC*) are marked in magenta and cyan, respectively. The corners are defined by the intersections of the resulting lines.

intersections of the lines in the plane defined by the target's normal and centroid. The four vertices are then re-projected to 3D space, as in Fig. 3.2(d).

3.5.3 Camera Corners and Associations

The process of determining the target corners begins by clicking near them in the image. This is the only manual intervention required in the chapter and even this process will soon be automated. As shown in Fig. 3.6(a), between any given two clicked points, i.e., an edge of the target, a bounding box is drawn around the two points. Once we have roughly located the corners of the target, we process the image with Canny edge detection[4] to detect the edge points within the bounding box. A line is then fit through each edge using *RANSAC*, and the intersections of the resulting lines define the target's corners, as shown in Fig. 3.6(b). A video and an implementation of this process is released along with the code.

3.5.4 Extrinsic Calibration

On the basis of the associated target vertices and camera-image corners, both the PnP and IoU methods are used to find the rigid-body transformation from LiDAR to camera. Because the results are similar, we report only the PnP method in Table 3.1 and then include both in the summary table (Table 3.2). SVD and *RANSAC* were computed using their corresponding MATLAB commands, while the optimizations in (3.4), (3.9) and (3.12) were done with `fmincon`.

[4]We use the *edge* command in MATLAB with the 'Canny' option.

Table 3.1: Fitting and validation data. The gray boxes denote the fitting set and the white boxes contain validation data. The numbers are the RMS errors of the LiDAR vertices projected to the image plane, measured in units of pixels per corner; see (3.13). For two targets, a complete round-robin study was done. For more targets, the number of combinations became prohibitive. S_1 through S_7 denote scene (experiment) number. The mean and standard deviation of each row—excluding the fitting set—are given in the last two columns.

Fitting\Validation	Method	# Tag	S1	S2	S3	S4	S5	S6	S7	mean	std
S1	RN	2	2.6618	8.7363	4.4712	7.3851	4.1269	7.1884	11.9767	7.3141	2.8991
	GL_1	2	0.7728	**2.3303**	**2.3230**	**1.6318**	**1.3694**	**1.9637**	**2.7427**	**2.0602**	**0.5055**
S2	RN	2	3.3213	7.6645	4.9169	5.2951	4.0811	4.4345	7.7397	4.9648	1.5215
	GL_1	2	**2.1368**	1.5181	**3.2027**	**3.2589**	**2.4480**	**4.1563**	**4.4254**	**3.2713**	**0.9039**
S3	RN	2	4.9909	9.5620	3.6271	8.7533	4.6421	10.1308	15.7764	8.9759	4.0654
	GL_1	2	**4.6720**	**5.9350**	1.8469	**6.9331**	**4.4352**	**8.9493**	**15.3862**	**7.7185**	**4.1029**
S4	RN	2	21.2271	22.1641	17.5779	2.9452	15.9909	8.8797	15.3636	16.8672	4.7833
	GL_1	2	**4.3681**	**4.7176**	**4.4804**	0.4986	**3.9004**	**3.3891**	**3.7440**	**4.0999**	**0.5040**
S5	RN	2	3.4621	8.3131	4.8500	7.6217	3.5197	7.4838	12.4364	7.3612	3.1066
	GL_1	2	**2.0590**	**2.9541**	**3.0224**	**3.5483**	0.7392	**3.1386**	**6.0885**	**3.4685**	**1.3733**
S6	RN	2	29.4400	27.5404	27.9955	9.7005	20.6511	1.4253	9.5050	20.8054	9.1941
	GL_1	2	**5.1207**	**5.0574**	**5.3537**	**2.0539**	**4.1739**	1.0012	**2.4194**	**4.0298**	**1.4503**
S7	RN	2	7.7991	9.9647	7.6857	4.1640	6.1619	2.3398	2.3708	6.3525	2.7512
	GL_1	2	**2.1563**	**2.7837**	**3.1058**	**1.3234**	**2.4537**	**2.0838**	1.5389	**2.3178**	**0.6207**
S6-S7	RN	4	6.9426	9.6281	6.9136	3.9650	5.6420	2.2399	2.2399	6.6183	2.0764
	GL_1	4	**2.2409**	**2.5607**	**2.9773**	**1.8215**	**2.1995**	0.3417	0.3417	**2.3600**	**0.4334**
S4-S6	RN	4	4.2476	8.5054	4.6712	2.8686	4.3311	2.8686	2.8748	4.9260	2.1153
	GL_1	4	**1.0746**	**1.8871**	**2.3619**	0.2341	**1.5350**	0.2341	**2.6191**	**1.8955**	**0.6215**
S5-S6	RN	4	2.9309	8.0801	4.1196	3.4985	3.1519	3.1519	3.2204	4.3699	2.1201
	GL_1	4	**1.1541**	**2.1026**	**2.5017**	**1.4267**	0.3291	0.3291	**2.5693**	**1.9509**	**0.6361**
S2-S5	RN	4	3.1991	6.0812	4.7311	5.4558	6.0812	4.8344	8.6576	5.3756	2.0139
	GL_1	4	**0.9538**	0.4803	**2.3810**	**1.4556**	0.4803	**1.1592**	**2.5277**	**1.6955**	**0.7172**
S2-S3	RN	4	2.8820	6.2167	6.2167	4.4672	3.9536	3.7802	6.2324	4.2631	1.2406
	GL_1	4	**1.0896**	0.4826	0.4826	**1.5349**	**1.5938**	**1.5597**	**2.7543**	**1.7065**	**0.6209**
S1-S7	RN	4	2.8595	8.2471	4.3297	3.7712	4.1023	2.4679	2.8595	4.5836	2.1710
	GL_1	4	1.3542	**1.9973**	**2.3987**	**1.5926**	**1.4978**	**1.6130**	1.3542	**1.8199**	**0.3757**
S1-S6	RN	4	2.5633	8.0573	4.1762	3.6859	3.8973	2.5633	3.2569	4.6147	1.9535
	GL_1	4	0.9753	**1.9154**	**2.3361**	**1.2906**	**1.2827**	0.9753	**2.3205**	**1.8291**	**0.5231**
S4-S5-S6	RN	6	2.9824	8.0943	4.1273	3.2821	3.2821	3.2821	3.0841	4.5720	2.4045
	GL_1	6	**0.9713**	**1.9887**	**2.4395**	0.3288	0.3288	0.3288	**2.4369**	**1.9591**	**0.6918**
S2-S3-S5	RN	6	2.7816	5.5382	5.5382	4.5817	5.5382	3.8262	6.6229	4.4531	1.6239
	GL_1	6	**1.0022**	0.4712	0.4712	**1.4755**	0.4712	**1.1722**	**2.7791**	**1.6073**	**0.8054**
S1-S3-S7	RN	6	3.3906	8.1677	3.3906	3.6688	3.9558	2.2913	3.3906	4.5209	2.5374
	GL_1	6	1.7666	**1.9994**	1.7666	**1.5854**	**1.4754**	**1.5124**	1.7666	**1.6432**	**0.2419**
S2-S3-S4	RN	6	2.9428	5.5357	5.5357	5.5357	3.8913	2.4828	3.7673	3.2711	0.6733
	GL_1	6	**0.9523**	1.8195	1.8195	1.8195	**1.5275**	**1.3631**	**2.4347**	**1.5694**	**0.6255**
S1-S2-S3	RN	6	5.3281	5.3281	5.3281	4.5086	3.8577	3.7165	6.4413	4.6310	1.2552
	GL_1	6	1.7755	1.7755	1.7755	**1.3698**	**1.4807**	**1.3422**	**2.3156**	**1.6271**	**0.4629**
S5-S6-S7	RN	6	3.2743	8.2067	4.2628	3.5584	3.0628	3.0628	3.0628	4.8256	2.2921
	GL_1	6	**0.9594**	**2.0149**	**2.3877**	**1.3754**	1.4905	1.4905	1.4905	**1.6843**	**0.6390**
S1-S3-S5-S7	RN	8	3.5316	8.1362	3.5316	3.4616	3.5316	2.2642	3.5316	4.6207	3.1029
	GL_1	8	1.6904	**1.9886**	1.6904	**1.4743**	1.6904	**1.4136**	1.6904	**1.6255**	**0.3159**
S1-S2-S5-S7	RN	8	4.9240	4.9240	4.2049	3.6159	4.9240	2.2928	4.9240	3.3712	0.9793
	GL_1	8	1.5269	1.5269	**2.4070**	**1.4788**	1.5269	**1.4650**	1.5269	**1.7836**	**0.5399**
S1-S3-S4-S5	RN	8	3.6113	8.0593	3.6113	3.6113	3.6113	2.4532	3.8397	4.7841	2.9199
	GL_1	8	1.4986	**2.0563**	1.4986	1.4986	1.4986	**1.2256**	**2.6797**	**1.9872**	**0.7295**
S2-S4-S5-S7	RN	8	3.0803	5.0057	4.1893	5.0057	5.0057	2.3307	5.0057	3.2001	0.9351
	GL_1	8	**0.9530**	1.5921	**2.4280**	1.5921	1.5921	**1.4865**	1.5921	**1.6225**	**0.7469**

Table 3.2: A summary of validation data in Table 3.1. This table compares mean and standard deviation for baseline and our approach as a function of the number of targets used in fitting. Units are pixel per corner.

Method	# Tag	2	4	6	8
Baseline-RN	mean	10.3773	4.9645	4.3789	3.9940
GL_1-PnP	mean	**3.8523**	**1.8939**	**1.6817**	**1.7547**
GL_1-IoU	mean	4.9019	2.2442	1.7631	1.7837
Baseline-RN	std	7.0887	1.9532	1.7771	2.0467
GL_1-PnP	std	**2.4155**	**0.5609**	0.5516	0.5419
GL_1-IoU	std	2.5060	0.7162	**0.5070**	**0.4566**

3.5.5 Computation Performance

The calibration is done offline. The round-robin cross-validation study given in Table 3.1 was generated in MATLAB in less than an hour. Each dataset (including the baseline and the proposed method) takes about 1.5 minutes in MATLAB.

3.5.6 Quantitative Results and Round-robin Analysis

In Table 3.1, we present the Root-Mean-Square (RMS) error of the LiDAR vertices projected into the camera's image plane for the baseline[5], labeled RANSAC-normal (**RN**), and our method in (3.3) and (3.4), labeled geometry-L_1 (**GL**$_1$). In the case of two targets, a full round-robin study is performed: the rigid-body transformation from LiDAR to camera is "fit" to the combined set of eight vertices from both targets and then "validated" on the eight vertices of each of the remaining six scenes.

A complete round-robin study for four targets from two scenes would require 21 validations, while for six and eight targets, 35 validations each would be required. For space reasons, we report only a subset of these possibilities.

To be clear, we are reporting in units of pixel per corner

$$\sqrt{\frac{1}{4n}\sum_{i=1}^{4n}\left\|{}_{L}Y_i - {}_{C}Y_i\right\|_2^2}, \tag{3.13}$$

where $4n$ is the total number of target corners. In the case of two targets and one scene, $n = 2$. In the case of six targets from three scenes, $n = 6$. Figure 3.7 illustrates several point clouds projected to the corresponding camera images. Summary data is presented in Table 3.2.

[5]SVD to extract the normal and *RANSAC* for individual line fitting, with the vertices obtained as the intersections of the lines.

Figure 3.7: Visual depiction of the validation data in the last row of Table 3.1. For the method $\mathbf{GL}_1$, five sets of estimated LiDAR vertices for each target have been projected into the image plane and marked in green, while the target's point cloud has been marked in red. Blowing up an image allows the numbers reported in the table to be visualized. The vertices are key.

Figure 3.8: Qualitative validation results. Using the extrinsic transformation obtained by the method $\mathbf{GL}_1$ applied to S_1, the LiDAR point cloud has been projected into the image plane for the other data sets and marked in green. The red circles highlight various poles, door edges, desk legs, monitors, and sidewalk curbs where the quality of the alignment can be best judged. The reader may find other areas of interest. Enlarge in your browser for best viewing.

3.6 Qualitative Results and Discussion

In LiDAR to camera calibration, due to the lack of ground truth, it is common to show projections of LiDAR point clouds onto camera images. Often it is unclear if one is viewing fitting data or validation data. In Figure 3.8, we show a set of projections of LiDAR point clouds onto camera

images for validation data. An additional set of images can be found in [62]. All of them show that the key geometric features in the image and point cloud are well aligned. The quality of the alignment has allowed our laboratory to build a high-quality (real-time) 3D semantic map with our bipedal robot Cassie Blue [50]. The map fuses LiDAR, camera, and IMU data; with the addition of a simple planner, it led to autonomous navigation [75].

Tables 3.1 and 3.2 show that $\mathbf{GL}_1$ outperforms the baseline: on average, there is more than a 50% reduction in projection error and a 70% reduction in its variance. As for the sources of this improvement, we highlight the following points:

(a) Least-squares-based methods estimate a normal vector for the target's point cloud. As shown in Fig. 3.2(a) and Fig. 3.2(b), even though the target is flat and has a well-defined normal, the returns in the LiDAR point cloud do not lie on a plane. Hence, a global normal vector does not exist.

(b) Least-squares-based methods extract the target edge points from the point cloud for use in line fitting. The line fitting must be done in the plane defined by the estimated normal because, in 3D, non-parallel lines do not necessarily intersect to define a vertex.

(c) $\mathbf{GL}_1$ explicitly uses the target geometry in formulating the cost function. By assigning zero cost to interior points in the "ideal target volume" and non-zero otherwise, the optimization problem is focused on orienting the boundary of the target volume within the 3D point cloud. This perspective seems not to have been used before.

(d) Hence, our approach does not require the (tedious and error-prone) *explicit* extraction and assignment of points to target edges. The determination of *boundary points* in 3D is *implicitly* being done with the cost function.

At the present time, our extrinsic calibration method is not "automatic". The one manual intervention, namely clicking on the approximate target corners in the camera image, will be automated soon.

We have not conducted a study on how to place the targets. This is an interesting piece of future work because of the nonlinear operation required when projecting LiDAR points to the camera plane; see (3.6) and (3.7).

3.7 Conclusions

We proposed a new method to determine the extrinsic calibration of a LiDAR camera pair. When evaluated against a state-of-the-art baseline, it resulted in, on average, more than a 50% reduction in LiDAR-to-camera projection error and a 70% reduction in its variance. These results were

established through a round-robin validation study when two targets are used in fitting, and further buttressed with results for fitting on four, six, and eight targets.

Two other benefits of our L_1-based method are: (1) it does not require the estimation of a target normal vector from an inherently noisy point cloud; and (2) it also obviates the identification of edge points and their association with specific sides of a target. In combination with lower RMS error and variance, we believe our results may provide an attractive alternative to current target-based methods for extrinsic calibration.

CHAPTER 4

Global Unifying Intrinsic Calibration on Lie Group for Spinning LiDAR and Solid-state LiDAR

4.1 Introduction

Camera and LiDARsensors and supporting software are common system elements in current robotic and autonomous systems. For real-world operation, such systems' performance depends on the quality of intrinsic and extrinsic calibration parameters. Intrinsic calibration of a sensor is the process of ensuring that obtained measurements are meaningful and valid. Intrinsic calibration of cameras is relatively mature [76–78] and many open-source packages are available [79–81]. Similarly, intrinsic calibration is also required for LiDARs; otherwise, the obtained point clouds are potentially inaccurate. However, even though spinning LiDARs have been on the market for over ten years, no publication supported by open-source calibration software is currently available, forcing academic researchers to either spend time developing their own solutions or to ignore concerns about the accuracy of their LiDAR point clouds.

Recently, another type of LiDAR, solid-state LiDAR, is being brought to the market. Industry is turning to solid-state LiDARs because they are small, quiet and produce less vibration. Due to their small size, they can be nicely integrated with headlights or carried by mobile robots such as drones. Moreover, solid-state LiDARs are more reliable especially for automotive applications due to the absence of the rotating mechanism. Because spinning LiDARs have a relatively simple working mechanism, it has been possible to propose physics-based calibration models [22–29], see Sec. 4.2.1. For solid-state LiDARs, however, multiple types and design options are being proposed (see Sec. 4.2.2), and thus it is going to be difficult to hand-craft a calibration model for each design of a solid-state LiDAR. We ask, is it necessary to do this? Are type-specific calibration models even the right way to approach the problem?

In this work, we propose a unifying view of calibration for different types of LiDARs, as shown in Fig. 4.1: we tie the calibration process to the data produced by the sensor, and not the sensor itself, by applying shape-preserving transformations to a LiDAR's point cloud. Additionally, we prove mathematically that the proposed calibration model is well-constrained (i.e., a unique

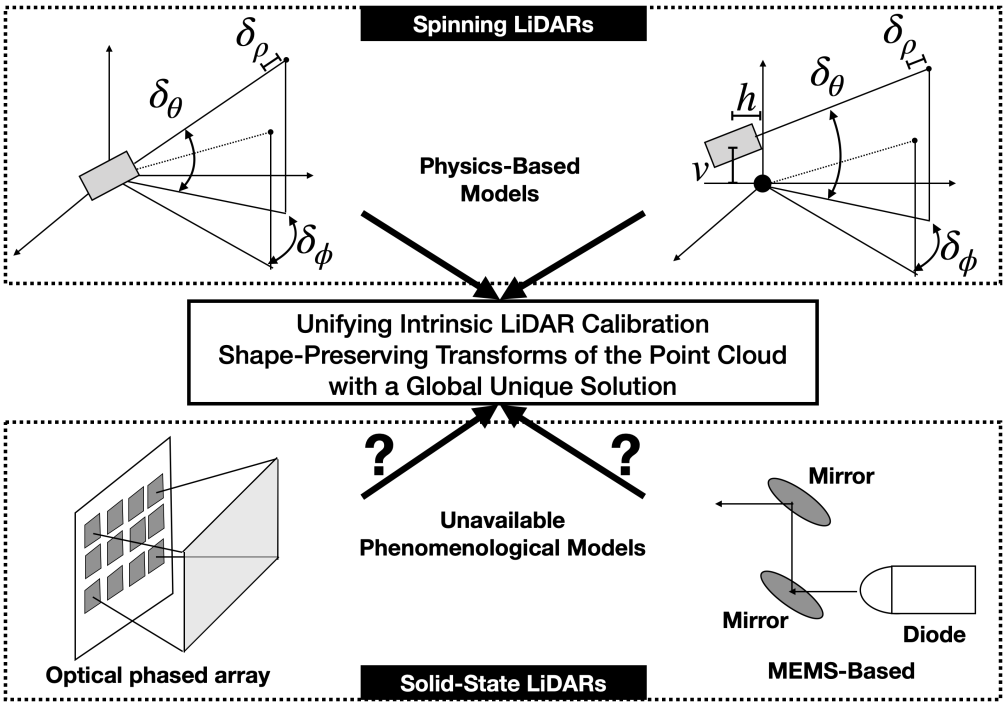


Figure 4.1: Is it possible to calibrate a LiDAR without modeling the physical mechanism itself? This chapter says YES it is and it allows a unifying means to calibrate LiDARs of all types. Top left: shows a simple physics-based calibration model for a spinning LiDAR. Top right: shows a similar model with more parameters, also for a spinning LiDAR. In the literature, the number of parameters can vary from three to ten. Bottom left and bottom right show, respectively, an optical phased array solid-state LiDAR and a Micro-Electro-Mechanical System (MEMS)-based solid-state LiDAR. There are many design options and many manufacturing steps to fabricate these LiDARs. This chapter moves the attention to the geometry of the LiDAR's point cloud and proposes that calibration can be achieved via shape-preserving transformations.

answer exists), whereas existing physics-based calibration models do not guarantee a unique set of calibration parameters. The uniqueness proof for the proposed method provides a guideline for target positioning. In addition, uniqueness of the parameters is necessary if one is to investigate a globally convergent parameter solver.

4.1.1 Our Contributions

1. We propose a model for intrinsic LiDAR calibration that is independent of the underlying technology. The new error model is based on shape-preserving transformations[1], which is a rigid body transformation plus a scale factor. Currently, intrinsic calibration models have been published for spinning LiDARs, while for solid-state LiDARs, the calibration models used by manufacturers have not found their way into the published literature. Current calibration models are tied to postulated physical mechanisms for how a LiDAR functions, resulting in anywhere from three to ten parameters to be estimated from data. Instead, we *abstract*

[1]Called *similarity transformations* in projective geometry.

away from the physics of a LiDAR type (spinning head vs. solid-state, for example) and focus on the spatial geometry of the point cloud generated by the sensor. *This leads to a unifying view of calibration* and results in an error model that can be represented as the action of the seven-dimensional matrix Lie group, $\mathrm{Sim}(3)$, on subsets of the measurements returned by the LiDAR.

2. One may suspect that our proposed model is over parameterized. We, therefore, mathematically prove that given four targets with appropriate orientations, the proposed model is well-constrained (i.e., only one answer exists). The proof provides a guideline for target placement, which is an oriented tetrahedron.

3. We give a globally convergent means to determine the calibration parameters. It utilizes an alternating optimization technique to solve the calibration problem, where the scaling parameter is determined by the bisection method, and the rest of the parameters ($\mathrm{SE}(3)$) are formulated as a Quadratically Constrained Quadratic Program (QCQP) where the Lagrangian dual relaxation is used. The relaxed problem on $\mathrm{SE}(3)$ becomes a Semidefinite Programming (SDP) and convex [1, 82–84]. Therefore, the calibration problem can be solved globally and efficiently by off-the-shelf solvers [85].

4. We provide a MATLAB-based simulator for intrinsic LiDAR calibration and validation that allows one to study the effects of target placement and the effectiveness of various calibration methods.

5. In simulations, we validate that positioning the four targets as a tetrahedron can calibrate equally well a spinning LiDAR and a solid-state LiDAR. In experiments on a spinning LiDAR, we show that the proposed method works as well as two physics-based models in terms of P2P distance, while it is more robust when we induce systematic errors to the experimental measurements. We also show that the proposed methods is able to calibrate a solid-state LiDAR and is robust to noise.

6. We open-source all of the related software for intrinsic calibration, including implementations of the baseline methods, the simulator, the data sets and the solver; see [86–88].

4.2 Related Work

This section overviews related work on spinning and solid-state LiDARs.

4.2.1 Spinning LiDAR

A spinning LiDAR as used in this chapter consists of K laser emitter-receiver pairs mounted at various positions and elevation angles on a rotating head, as shown in Fig. 4.1. The lasers are pulsed at a fixed frequency. Each revolution of the rotating head sweeps out a cone in space, called a LiDAR scan, composed of discrete return points traced out by each pulsed emitter-receiver. The points associated with a single laser beam are called a ring.

The measurements are typically viewed as being made in a spherical coordinate system. The range measurement is made by estimating ToF through a clocking system on a circuit board, the azimuth is estimated from the position of the rotating head, and the elevation angle of a beam is determined by its mounting angle on the rotating head.

Existing error models for LiDARs are phenomenological in nature, that is, they are tied to physical explanations of a sensor's design and/or operation [22–29]. Broadly speaking, two common sources of uncertainty considered in spinning LiDAR models are (1) errors in the clocking systems, (one for ToF and one for pulsing the lasers) and (2) mechanical errors associated with positioning of the beams on the spinning LiDAR. Less commonly considered are affects are due to ambient temperature or target size. Some authors overlook that the measurements are actually made in a local coordinate frame for each emitter-receiver pair, and are then transformed to a single "global" frame for the LiDAR unit, resulting in models with three to six parameters [23, 24, 28], while others take this into account and use as many as ten parameters [29]. Moreover, none of these models provides a guideline for target placement so that the calibration model is well-constrained (i.e., a unique solution exists), see Sec. 4.3.3.

4.2.2 Solid-State LiDAR

In comparison to spinning LiDARs, solid-state LiDARs are smaller, quieter, lower-cost, and produce less vibration. Several types of solid-state LiDAR are coming to the market, including those based on (1) Optical Phased Array (OPA) [89–91], and (2) MEMS mirrors[2] [92–97].

OPA-based solid-state LiDARs function similarly to a phased array in antenna theory. A single OPA contains a significant number of emitters patterned on a chip. By varying the phase shift (i.e., time delay) of each emitter, the direction of the resulting wave-front or the spread angle can be adjusted. In particular, the sensor is able to zoom in or out of an object on command. On the other hand, MEMS-based solid-state LiDARs use only one or a few emitters, with the beams guided by a mirror system regulated by a sophisticated controller. The controller seeks to steer the beam (or beams) in a scanning pattern, such as a zig-zag [92].

[2]Some authors argue MEMS-based solid-state LiDARs are not truly solid-state devices due to the movable mirrors; the mirrors, however, are tiny compared to a spinning LiDAR.

To perfectly align OPA emitters or MEMS mirrors is challenging. It is also hard to maintain high accuracy under different temperatures and weather conditions. Thus, how to properly calibrate a solid-state LiDAR is a critical problem.

The variety of solid-state LiDARs raises the question: is it necessary, or even possible, to design/create parameterized models for every type of solid-state LiDAR? This chapter seeks to present a unifying view of LiDAR calibration by abstracting away from the physics of a LiDAR type (spinning vs solid state, for example), and focusing instead on the spatial geometry of the point cloud generated by the sensor. This leads to a unifying view of calibration.

4.3 Proposed LiDAR Intrinsic Model and Analysis

The proposed calibration parameters are modeled as an element of the similarity Lie group $\mathrm{Sim}(3)$, which can be applied to both spinning LiDARs and solid-state LiDARs. An element of the group is an isometry composed with an additional isotropic scaling, i.e., a rigid-body transformation with scaling. We note that this is the most general form of of a shape-preserving transformation in 3D space (i.e. preserves ratios of lengths and angles). We also note that (4.8) and (4.9) are not shape preserving. It will be important to see in real data if our assumption of shape preservation is general enough to provide useful calibrations.

An element of the group is given by

$$\mathbf{H} = \begin{bmatrix} s\mathbf{R} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathrm{Sim}(3), \tag{4.1}$$

where $\mathbf{R} \in \mathrm{SO}(3)$ is the 3D rotation matrix, $\mathbf{v} \in \mathbb{R}^3$ is the translation vector, and $s \in \mathbb{R}^+$ is the scale parameter. In particular, an element has seven degrees of freedom and the action of $\mathrm{Sim}(3)$ on $\mathbb{R}^3$ is $\mathbf{H} \cdot \mathbf{x} = s\mathbf{R}\mathbf{x} + \mathbf{v}$, where $\mathbf{x} \in \mathbb{R}^3$ and where $\mathbf{H} \cdot \mathbf{x}$ uses homogeneous coordinates while $s\mathbf{R}\mathbf{x} + \mathbf{v}$ uses Cartesian coordinates. From hereon, wherever convenient, we abuse notation by passing from regular coordinates to homogeneous coordinates without noting the distinction.

4.3.1 Point-to-Plane Distance

Let $\mathbf{n}_t$ be a unit normal vector of a planar target and let $\mathbf{p}_{0,t}$ be a fixed point on the target. Let $\mathcal{X} = \{\mathbf{x}_i | i = 1, \ldots, M \text{ and } \mathbf{x}_i \in \mathbb{R}^3\}$ denote a collection of LiDAR returns on the target, and M be the number of points in the collection. Given the collection of LiDAR returns $\mathcal{X}$, the cost of the intrinsic calibration problem is commonly defined by the P2P distance:

$$J_t := \sum_{i=1}^{M} J_i(\mathbf{x}_i, \mathbf{n}_t, \mathbf{p}_{0,t}) := \sum_{i=1}^{M} \left(\mathbf{n}_t^{\mathsf{T}} \left(\mathbf{x}_i - \mathbf{p}_{0,t}\right)\right)^2, \tag{4.2}$$

where $\mathbf{n}_t^{\mathsf{T}} \left(\mathbf{x}_i - \mathbf{p}_{0,t}\right)$ is the orthogonal projection of the measurement onto the normal vector. Then the calibration problem can be formulated as follows

Problem 1. *For a given collection of points $\mathcal{X}_t = \{\boldsymbol{x}_i\}_{i=1}^{M_t}$, possibly from multiple targets $t \in \{1, \ldots, T\}$, we seek a similarity transformation $\boldsymbol{H}^\star$ that solves*

$$\min_{\boldsymbol{H} \in \mathrm{Sim}(3)} \sum_{t=1}^{T} \sum_{i=1}^{M_t} \left(\boldsymbol{n}_t^{\mathsf{T}} \left(\boldsymbol{H} \cdot \boldsymbol{x}_i - \boldsymbol{p}_{0,t}\right)\right)^2. \tag{4.3}$$

Remark 4. *The transformation H applies to the entire collection of points $\mathcal{X}$. How to define the collection is addressed in Sec. 4.3.4.*

In practice, each target's normal vector and a point on the target must be estimated from data. For completeness, we briefly summarize how to do this; see [9,40] for details. Given a collection of LiDAR returns on a planer target and a template of the target with vertices located at the origin of the LiDAR and aligned with the y-z plane, we seek a rigid-body transformation from the template to the target, $H_L^T \in \mathrm{SE}(3)$, that "best fits" the template onto the LiDAR returns of the target. The normal vector of the LiDAR returns can be computed from transformed vertices of the template. Furthermore, in the raw data, the rings are mis-calibrated (by assumption). Because the target planes are estimated using data corresponding to mis-calibrated rings, the estimated normal vectors are inaccurate. We therefore propose a refinement of the normal vectors conditioned on the current estimate of the similarity transformation $\mathrm{Sim}(3)$, that is, based on the current estimate of the intrinsic parameters $(s, \mathbf{R}, \mathbf{v})$. Once the normal vectors are updated, new estimates for $(s, \mathbf{R}, \mathbf{v})$ can be obtained, resulting in an alternating two-step process.

4.3.2 Overfitting, Underfitting, and Model Mismatch

We first point out that the proposed calibration model can suffer from overfitting (i.e., a unique $\mathbf{H}^*$ does not exist) if less than four targets are used to calibrate a LiDAR. In particular, for a single planar target, the rotation about the target normal $\mathbf{n}_t$, two components of the translation vector $\mathbf{v}$ (translation in the plane of the target), and the scale s are unconstrained. For two planar targets, translation along the line orthogonal to the targets' normal vectors and scale are unconstrained. Therefore, understanding how target positioning constrains the calibration model is essential. As mentioned above, underfitting could occur if the data is parsed into rings, but say one of the rings has distance errors that vary with rotation angle. In this case, splitting each ring into different arcs will eliminate the underfitting.

Our calibration model is not a phenomenological model. The existing published models postulate how a LiDAR works and hence are phenomenological models. These postulated models result in different numbers of parameters, ranging from three to eight. Our "calibration model" abstracts

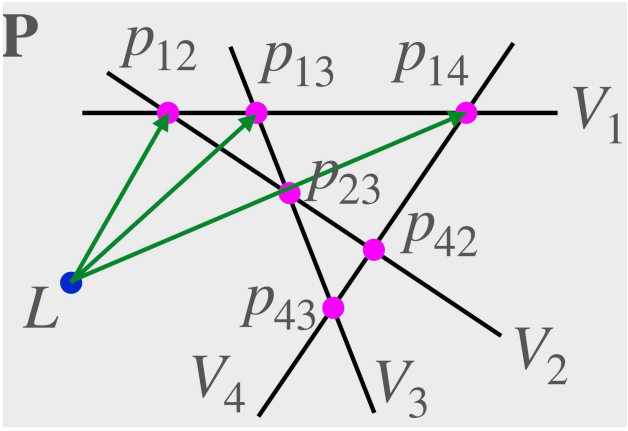


Figure 4.2: Illustration of the basis' of a ring plane, where L is the LiDAR origin, $\mathbf{P}$ is the ring plane, $\{\mathbf{V}_i\}_{i=1}^4$ are four targets, and $\mathbf{p}_{ij} := \mathbf{P} \cap \mathbf{V}_i \cap \mathbf{V}_j$ are the unique points under Assumption N.

away from the physical device and focuses on transforming the point cloud so that the (x, y, z) values are well calibrated.

4.3.3 Target Placement Guideline

Theorem 1 given below provides practically realizable conditions under which a unique answer exists to Problem 1. The proof is given in A.1. We further propose a method to globally compute the unique solution, see Sec. 4.5.

For a subset $\mathcal{S} \subset \mathbb{R}^3$, let $[\mathcal{S}]$ and $[\mathcal{S}]^\perp$ denote its span and its orthogonal complement, respectively. Let $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ be the canonical basis for $\mathbb{R}^3$. We also denote $\mathbf{P} \subset \mathbb{R}^3$ a set traced out by a single ring (i.e., a ring plane) of a perfectly calibrated spinning LiDAR. Without loss of generality, we assume $\mathbf{P} = [\mathbf{e}_3]^\perp$.

Consider four targets with unit normal vectors $\mathbf{n}_i$ and let $\mathbf{p}_{0,i}$ be a point on the i-th target. For $1 \leqslant i \leqslant 4$, the plane defined by the i-th target is

$$\begin{aligned}\mathbf{V}_i &:= \mathbf{p}_{0,i} + [\mathbf{n}_i]^\perp \\ &:= \mathbf{p}_{0,i} + \{v \in \mathbb{R}^3 \mid \mathbf{n}_i^\top v = 0\},\end{aligned}$$

the plane formed by the set of vectors orthogonal to the normal vector $\mathbf{n}_i$, and subsequently translated by $\mathbf{p}_{0,i}$.

Assumption N (Normal Vectors)
All sets of three distinct vectors from $\{\boldsymbol{n}_1, \boldsymbol{n}_2, \boldsymbol{n}_3, \boldsymbol{n}_4, \boldsymbol{e}_3\}$ *are linearly independent.*

Under Assumption N, for each $i \neq j \in \{1, 2, 3, 4\}$, there exist unique points $\mathbf{p}_{ij} := \mathbf{P} \cap \mathbf{V}_i \cap \mathbf{V}_j$. All total, there are $\binom{4}{2} = 6$ such intersection points.

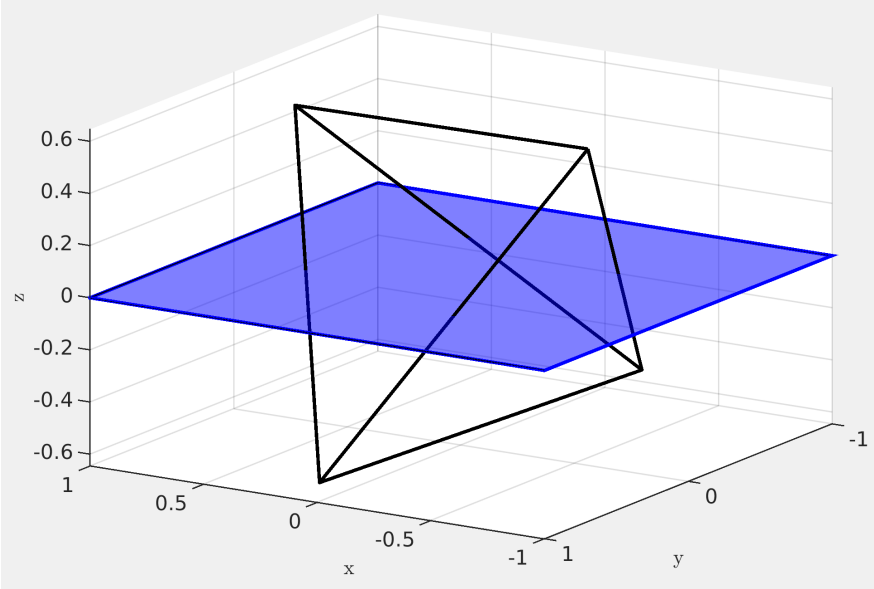


Figure 4.3: An oriented tetrahedron (black) and a ring plane (blue) for target positioning that satisfies the conditions of Theorem 1.

Assumption B (Basis Vectors)

Each of the following sets of vectors is a basis of the ring plane:

(a) $\{\boldsymbol{p}_{12},\boldsymbol{p}_{13}\}$, $\{\boldsymbol{p}_{13},\boldsymbol{p}_{14}\}$, $\{\boldsymbol{p}_{14},\boldsymbol{p}_{12}\}$,

(b) $\{\boldsymbol{p}_{12},\boldsymbol{p}_{23}\}$, $\{\boldsymbol{p}_{23},\boldsymbol{p}_{24}\}$, $\{\boldsymbol{p}_{24},\boldsymbol{p}_{12}\}$,

(c) $\{\boldsymbol{p}_{13},\boldsymbol{p}_{23}\}$, $\{\boldsymbol{p}_{23},\boldsymbol{p}_{34}\}$, $\{\boldsymbol{p}_{34},\boldsymbol{p}_{13}\}$,

(d) $\{\boldsymbol{p}_{14},\boldsymbol{p}_{24}\}$, $\{\boldsymbol{p}_{24},\boldsymbol{p}_{34}\}$, $\{\boldsymbol{p}_{34},\boldsymbol{p}_{14}\}$,

(e) $\{\boldsymbol{p}_{14},\boldsymbol{p}_{23}\}$

We note that for a perfectly calibrated LiDAR, if $\mathbf{H}$ is the identity element of $\mathrm{SE}(3)$, then for all $i \neq j \in \{1,2,3,4\}$, it must be true that $\mathbf{H} \cdot \mathbf{p}_{ij} \in \mathbf{V}_i \cap \mathbf{V}_j$. The following shows that the converse is true when the targets are appropriately positioned and oriented.

Remark 5. *In fact, there are* $\binom{6}{2} = 15$ *possible pairs of vectors, of which we use 13 in Assumption B to establish uniqueness of the proposed calibration model. In Fig. 4.2, we illustrate* ***(a)*** *in Assumption B, where* L *is the LiDAR origin,* $\boldsymbol{P}$ *is the ring plane,* $\{\boldsymbol{V}_i\}_{i=1}^4$ *are four targets, and* $\boldsymbol{p}_{ij} := \boldsymbol{P} \cap \boldsymbol{V}_i \cap \boldsymbol{V}_j$ *are the unique points under Assumption N.*

Theorem 1. *Assume that Assumptions N and B hold and let* $\boldsymbol{H} \in \mathrm{Sim}(3)$. *If for each* $i \neq j \in \{1,2,3,4\}$, $\boldsymbol{H} \cdot \boldsymbol{p}_{ij} \in \boldsymbol{V}_i \cap \boldsymbol{V}_j$, *then*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{bmatrix}. \tag{4.4}$$

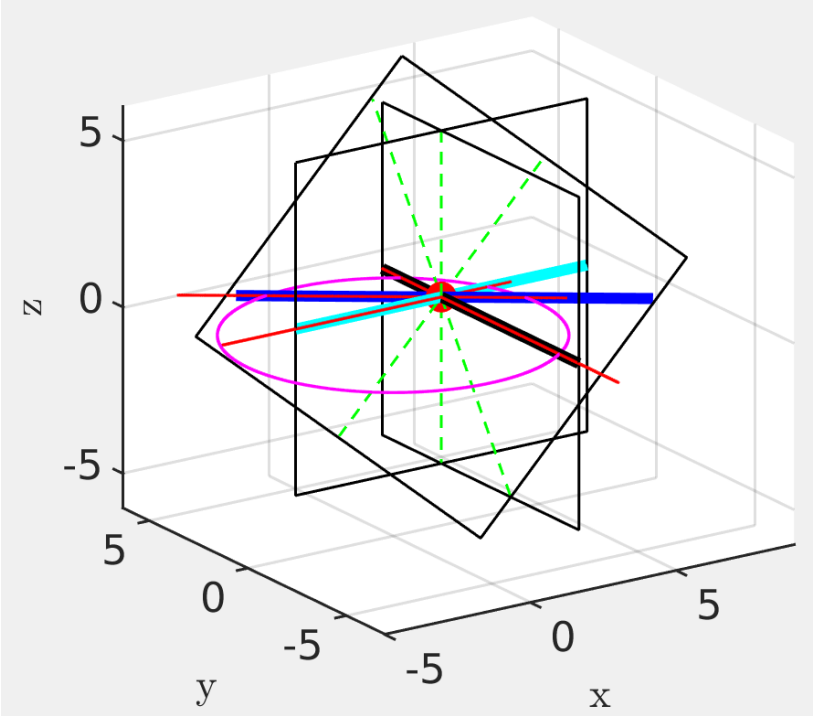


Figure 4.4: Illustration of a degenerate case. The black squares are different planes with the normal vectors being linearly independent. The dotted green lines indicate the intersection of two planes. The magenta circle shows the LiDAR ring passing through q_0 (red center bullet) in (4.5). The thick blue, cyan and black lines are the intersections of the ring planes and the targets. The P2P cost of the intersections are therefore zero. The red lines show the results of applying a transformation $\mathbf{H} = \begin{bmatrix} s\mathbf{I} & (1-s)\mathbf{q}_0 \\ \mathbf{0} & 1 \end{bmatrix}$ to the intersection lines. The cost remains zero because the transformed lines stay on the targets.

Proof: See A.1.

An immediate Corollary of Theorem 1 is the uniqueness of the transformation, as stated below.

Corollary. Let $\widetilde{\mathbf{H}} \in \mathrm{Sim}(3)$ and define $\widetilde{\mathbf{p}}_{ij} := \widetilde{\mathbf{H}}(\mathbf{p}_{ij})$. Then under Assumptions N and B, the only element $\mathbf{G} \in \mathrm{Sim}(3)$ satisfying $\mathbf{G}(\widetilde{\mathbf{p}}_{ij}) \in \mathbf{V}_{ij}$ all $i \neq j$ is $\mathbf{G} = \widetilde{H}^{-1}$.

Assumption N assumes linear independence ($\binom{5}{3} = 10$) for any three of the target normal vectors, but does not directly address their "degree of independence," that is, their condition number. Similarly, Assumption B only involves linear independence. In practice, one wonders if it is important to position the targets so that the two sets of vectors are well conditioned? We will later show that the normal vectors are most important. In Fig. 4.3, we show an oriented tetrahedron[3] where the normal vector of each face[4] satisfies Assumption N and the orientation of the tetrahedron intersecting with the ring plane fulfills Assumption B.

[3]Each of the four planar targets defines an infinite plane. The infinite planes create a tetrahedron.

[4]Faces of the tetrahedron are extended planes of the targets. Therefore, normal vectors of faces are normal vectors of targets.

Remark 6. *The plane defined by the i-th target is $\mathcal{V}_i := \boldsymbol{p}_{0,i} + \boldsymbol{n}_i^{\perp}$. If we were to only use three targets, there would exist a unique point $\boldsymbol{q}_0 \in \mathbb{R}^3$ defined by*

$$\boldsymbol{q}_0 := \bigcap_{i=1}^{3} \mathcal{V}_i. \tag{4.5}$$

When (4.5) *holds, the scale factor, s, is not unique; see Fig. 4.4. In other words, a ring plane passing through the point $\boldsymbol{q}_0$ leads to the scale factor s being unconstrained and therefore a unique answer does not exist. Therefore, it was critical to analyze target placement for the proposed method. A similar analysis seems to be missing for the baseline methods.*

4.3.4 Parsing Points on a Target to Collections of Points

Each cluster of points is assigned its own calibration parameters. To ensure uniqueness of the calibration parameters, a cluster must contain returns from multiple target planes that are oriented so as to meet an "independence" condition. We establish conditions on the relative geometry of the target planes to achieve uniqueness of the calibration parameters. This geometry can even be achieved by rotating a LiDAR with respect to a single fixed target as it is the relative geometry that matters.

The specific method for parsing a target's point cloud and the number of calibration transformations depends on the nature of the sensor. For a spinning LiDAR, the points $\mathcal{PC}$ on a target are typically separated by beam number so each beam has its own optimization problem (4.3). For example, for a K-beam spinning LiDAR (shown in Fig. 4.5), a natural choice for the set of calibration parameters is $\{\mathbf{H}_k | k = 1, \ldots, K \text{ and } \mathbf{H}_k \in \mathrm{Sim}(3)\}$, where H_k is the similarity transform for the k-th ring. Similarly, one could also split each ring into quarter circles (90° arcs), which would lead to $4K$ calibration parameters. On the one hand, for an OPA-based solid-state LiDAR, we propose to form an $m \times n$ grid over the planar target and then parse the point cloud based on each point's projection onto the target along the normal of each grid. This way, the set of calibration parameters becomes $\{\mathbf{H}_k | k = 1, \ldots, m \times n \text{ and } \mathbf{H}_k \in \mathrm{Sim}(3)\}$. On the other hand, one may wish to stick with parsing via "rows or columns". In the end, it's a choice that has to be made by the calibrator.

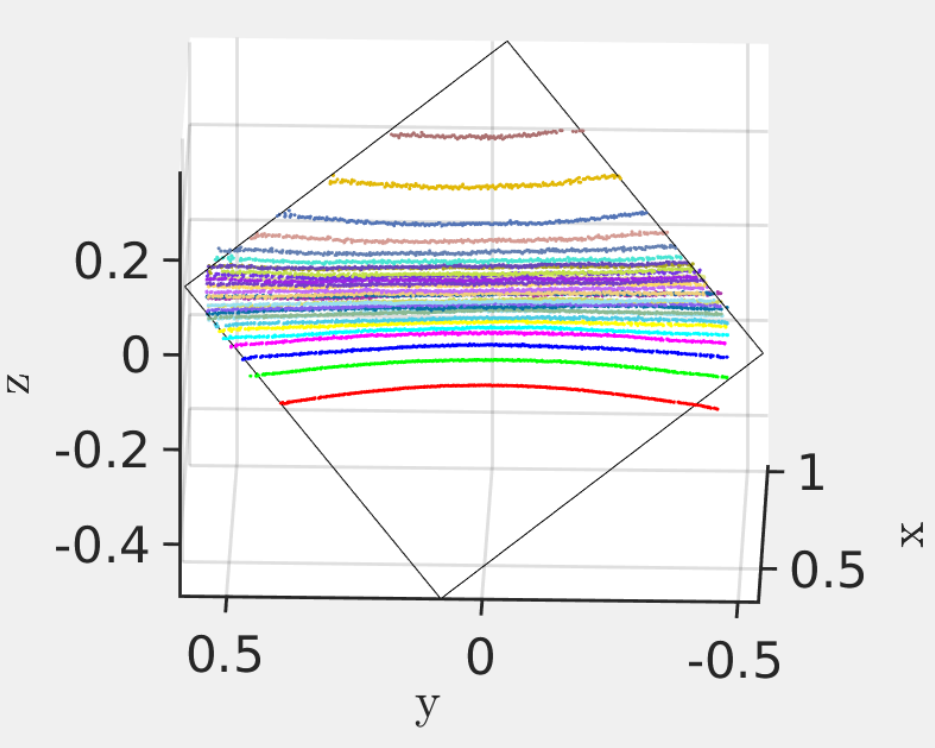


Figure 4.5: Illustration of collections of LiDAR returns of the target from a spinning LiDAR. Different colors stand for different collections.

4.4 Baseline Intrinsic Calibration Methods

There are two standard calibration models for spinning LiDARs. They are based on spherical coordinates (ρ, θ, ϕ), referred to as range, elevation and azimuth,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = f(\rho, \theta, \phi) = \begin{bmatrix} \rho \cos\theta \sin\phi \\ \rho \cos\theta \cos\phi \\ \rho \sin\theta \end{bmatrix}, \tag{4.6}$$

and the inverse function,

$$\begin{bmatrix} \rho \\ \theta \\ \phi \end{bmatrix} = f^{-1}(x, y, z) = \begin{bmatrix} \sqrt{x^2 + y^2 + z^2} \\ \sin^{-1}(\frac{z}{\sqrt{x^2+y^2+z^2}}) \\ \text{atan2}(x, y) \end{bmatrix}. \tag{4.7}$$

4.4.1 3-Parameter Model

The most basic model [23] assumes the LiDAR measurements are made in spherical coordinates, (ρ, θ, ϕ). Corrections to a measurement are given by a collection of offsets $\alpha := (\delta_\rho, \delta_\theta, \delta_\phi)$.

Expressing the calibrated measurement in Cartesian coordinates gives

$$\Gamma_\alpha(\rho,\theta,\phi) := \begin{bmatrix} (\rho+\delta_\rho)\cos(\delta_\theta)\sin(\phi-\delta_\phi) \\ (\rho+\delta_\rho)\cos(\delta_\theta)\cos(\phi-\delta_\phi) \\ (\rho+\delta_\rho)\sin(\delta_\theta) \end{bmatrix} \tag{4.8}$$

Remark 7. ***(a)*** *The nominal elevation for each ring is taken as zero;, i.e., $\theta = 0$.* ***(b)*** *While it is not necessarily a drawback, most LiDAR interfaces only return a Cartesian representation of a measured point. For example, the ROS driver from Velodyne only provides the Cartesian coordinate and the intensity of a point. Hence, the user must assure the transformation to spherical coordinates, make the measurement correction coming from the calibration, and then transform back to Cartesian coordinates.*

4.4.2 6-Parameter Model

This model [24, 25] also works in spherical coordinates. In addition to the three offsets above, it includes s, a scale factor of ρ, h, a horizontal offset of the origin, and v, a vertical offset for the origin. The correction model becomes

$$\bar{\Gamma}_\alpha := \begin{bmatrix} (s\rho+\delta_\rho)\cos\delta_\theta\sin(\phi-\delta_\phi) - h\cos(\phi-\delta_\phi) \\ (s\rho+\delta_\rho)\cos\delta_\theta\cos(\phi-\delta_\phi) + h\sin(\phi-\delta_\phi) \\ (s\rho+\delta_\rho)\sin\delta_\theta + v \end{bmatrix}, \tag{4.9}$$

and therefore $\alpha := (\delta_\rho, \delta_\theta, \delta_\phi, s, h, v)$.

4.4.3 Can the models be expressed by elements of $\mathrm{Sim}(3)$?

The short answer is no. When viewed in Cartesian coordinates, the calibration model (4.9) is fundamentally nonlinear and can be expressed as $\mathbf{R}_1(\mathbf{R}_2\mathbf{t}_1 + \mathbf{t}_2)$, where $\mathbf{R}_1, \mathbf{R}_2, \mathbf{t}_1$ and $\mathbf{t}_2$ depend on the measured point,

$$\mathbf{x}^\mathsf{T} = [x, y, z] \xrightarrow{f^{-1}} (\rho, \theta, \phi),$$

$$\mathbf{R}_1 = \begin{bmatrix} \sin(\phi-\delta_\phi) & -\cos(\phi-\delta_\phi) & 0 \\ \cos(\phi-\delta_\phi) & \sin(\phi-\delta_\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.10}$$

$$\mathbf{R}_2\mathbf{t}_1 + \mathbf{t}_2 = \begin{bmatrix} \cos(\delta_\theta) & 0 & -\sin(\delta_\theta) \\ 0 & 1 & 0 \\ \sin(\delta_\theta) & 0 & \cos(\delta_\theta) \end{bmatrix} \begin{bmatrix} s\rho+\delta_\rho \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ h \\ v \end{bmatrix}.$$

The calibration first applies a corrected elevation angle to a range value that has been offset and scaled. Next, the (y, z) position of the origin is offset, and lastly, a corrected rotation about the z-axis is applied.

4.4.4 Cost function for baseline models

For a given collection of points $\mathcal{PC}$, possibly from multiple targets $t \in \{1, \ldots, T\}$, we seek calibration parameters that solve

$$\min_{\alpha} \sum_{t=1}^{T} \sum_{i=1}^{M_t} |\mathbf{n}_t^{\mathsf{T}} \left(F(\mathbf{x}_i, \alpha) - \mathbf{p}_{0,t} \right)|, \tag{4.11}$$

where

1. for baseline model $\mathbf{BL}_1$ [23] in (4.8), $\alpha = (\delta_\rho, \delta_\theta, \delta_\phi)$ and

$$F(\mathbf{x}, \alpha) := \Gamma_\alpha \circ f^{-1}(\mathbf{x});$$

2. for baseline model $\mathbf{BL}_2$ [24, 25] in (4.9), $\alpha = (\delta_\rho, \delta_\theta, \delta_\phi, s, h, v)$ and

$$F(\mathbf{x}, \alpha) := \bar{\Gamma}_\alpha \circ f^{-1}(\mathbf{x}).$$

Remark 8. *For the 3-parameter model $\boldsymbol{BL}_1$, a single planar target is sufficient to uniquely determine a calibration. For the 6-parameter model $\boldsymbol{BL}_2$, non-uniqueness can occur as outlined in Theorem 1 and Fig. 4.4.*

4.5 Global Optimization: from Min to Min-Min

Theorem 1 states that a unique answer exists. In this section, we propose a method to find it. In particular, we show a globally convergent algorithm to determine the element of the Lie Group that globally minimizes the cost function (4.2). This is done by converting the minimization problem in (4.3) to a Min-Min problem. The inner minimization problem has an efficient global solution by results in [1, 84]. The outer minimization is scalar, and it is easy to limit the range of interest to a compact set.

Algorithm 1: Proposed Global Optimizer for $\mathrm{Sim}(3)$

Input: Collections of points $(\mathcal{X}_t)$ and corresponding targets $(\mathbf{n}_t, \mathbf{p}_{0,t})$

Output: Global optimum $\mathbf{H}^* \in \mathrm{Sim}(3)$

Initialization: Arbitrary initialization is allowable. We use $\mathbf{R} = \mathbf{I}, \mathbf{v} = \mathbf{0}$, and $s \in [0.8, 1.2]$

while $k <$ *MAX_ITER* **do**

// Scale the collections of points

${}^{U}_{k}\mathcal{X}^{s}_{t} \leftarrow {}^{U}_{k}s \cdot \mathcal{X}_t, \quad {}^{L}_{k}\mathcal{X}^{s}_{t} \leftarrow {}^{L}_{k}s \cdot \mathcal{X}_t$

// Compute the optimum $\mathbf{R}, \mathbf{v}$ given the scaled points

${}^{L}_{k}\mathbf{R} \leftarrow {}^{L}_{k}\mathcal{X}^{s}_{t}$ and (42) in [21], ${}^{L}_{k}\mathbf{v} \leftarrow {}^{L}_{k}\mathbf{R}$ and (36) in [21]

${}^{U}_{k}\mathbf{R} \leftarrow {}^{U}_{k}\mathcal{X}^{s}_{t}$ and (42) in [21], ${}^{U}_{k}\mathbf{v} \leftarrow {}^{U}_{k}\mathbf{R}$ and (36) in [21]

// Compute the P2P by current $\mathbf{R}, \mathbf{v}$

${}^{L}_{k}J \leftarrow$ (13) in [21], ${}^{U}_{k}J \leftarrow$ (13) in [21]

// Update ${}^{U}_{k+1}s$ and ${}^{L}_{k+1}$

${}^{L}_{k+1}s \leftarrow$ (21) in [21], ${}^{U}_{k+1}s \leftarrow$ (21) in [21]

return $s^*, \mathbf{R}^*, \mathbf{v}^*$

4.5.1 Problem Statement and Compact Sets

The intrinsic calibration problem is defined in (4.3):

$$\min_{s,\mathbf{R},\mathbf{v}} J(s, \mathbf{R}, \mathbf{v}; \mathcal{X}_t, \mathbf{n}_t, \mathbf{p}_{0,t}), \tag{4.12}$$

where the cost is defined as

$$J(s, \mathbf{R}, \mathbf{v}) := \sum_{t=1}^{T} \sum_{i=1}^{M_t} |\mathbf{n}_t^{\mathsf{T}} \left(s\mathbf{R} \cdot \mathbf{x}_i + \mathbf{v} - \mathbf{p}_{0,t} \right)|^2. \tag{4.13}$$

It is straightforward to show that J is a continuous function on $\mathrm{Sim}(3)$. As a set, we write $\mathrm{Sim}(3) = \mathbb{R}^+ \times \mathrm{SO}(3) \times \mathbb{R}^3$, where $\mathbb{R}^+ := \mathbb{R} > 0$. Note that $\mathrm{SO}(3)$ is compact. To guarantee that a minimum exists, we consider compact subsets of $s \in \mathbb{R}^+$ and $\mathbf{v} \in \mathbb{R}^3$ and define (4.12):

$$\begin{aligned} J^* &:= \min_{s,\mathbf{R},\mathbf{v}} J(s, \mathbf{R}, \mathbf{v}) \\ s^*, \mathbf{R}^*, \mathbf{v}^* &:= \arg\min_{s,\mathbf{R},\mathbf{v}} J(s, \mathbf{R}, \mathbf{v}). \end{aligned} \tag{4.14}$$

4.5.2 From Min to Min-Min

We use again the fact that $\mathrm{Sim}(3) = \mathbb{R}^+ \times \mathrm{SO}(3) \times \mathbb{R}^3$ to redefine (4.12) as a Min-Min problem.

Proposition 1.

$$\min_{s,\boldsymbol{R},\boldsymbol{v}} J(s,\boldsymbol{R},\boldsymbol{v}) = \min_{s} \min_{\boldsymbol{R},\boldsymbol{v}} J(s,\boldsymbol{R},\boldsymbol{v}), \tag{4.15}$$

where the minimums are taken over the same compact sets used in (4.14)*.*

Proof: From (4.14), we have that

$$J^* = J(s^*, \mathbf{R}^*, \mathbf{v}^*). \tag{4.16}$$

We further introduce

$$\begin{aligned} h(s) &:= \min_{\mathbf{R},\mathbf{v}} J(s,\mathbf{R},\mathbf{v}) \\ h^* &:= \min_{s} h(s) \end{aligned} \tag{4.17}$$

and note that by definition, we have

$$J^* \leqslant h^*. \tag{4.18}$$

To show $h^* \leqslant J^*$, we use that minimizing values of J exist and hence

$$h^* \leqslant h(s^*) := \min_{\mathbf{R},\mathbf{v}} J(s^*,\mathbf{R},\mathbf{v}) \leqslant J(s^*,\mathbf{R}^*,\mathbf{v}^*) = J^*. \tag{4.19}$$

Equations (4.18) and (4.19) conclude the equivalency of (4.15).

■

From Proposition 1, we can define the intrinsic calibration problem as

$$f(s) := \min_{\mathbf{R},\mathbf{v}} J(s,\mathbf{R},\mathbf{v};\mathcal{X}_t,\mathbf{n}_t,\mathbf{p}_{0,t}) \tag{4.20}$$

$$J^* = \min_{s} f(s). \tag{4.21}$$

4.5.3 Determining the Calibration Parameters

Because we can bound the scaling s to a compact set, a minimizing value can be found by dense search. In the A.3, we discuss a heuristic to speed up the algorithm if one feels the need.

At the $(k+1)$-th iteration, the scaling parameter is determined and the remaining parameters correspond to $\mathrm{SE}(3)$. To solve for them, we adopt techniques that were used to solve 3D registration or 3D SLAM [1, 82–84] where the problem is formulated as a QCQP, and the Lagrangian dual relaxation is used. The relaxed problem becomes a SDP and convex. The problem can thus be solved globally and efficiently by off-the-shelf specialized solvers [85]. This process is summarized in A.2. As shown in [84], the dual relaxation is empirically always tight (the duality gap is zero). We also use the same simulation and experimental data set in [1, 84] to verify that the proposed algorithm is able to converge globally, see A.3 for more visual results.

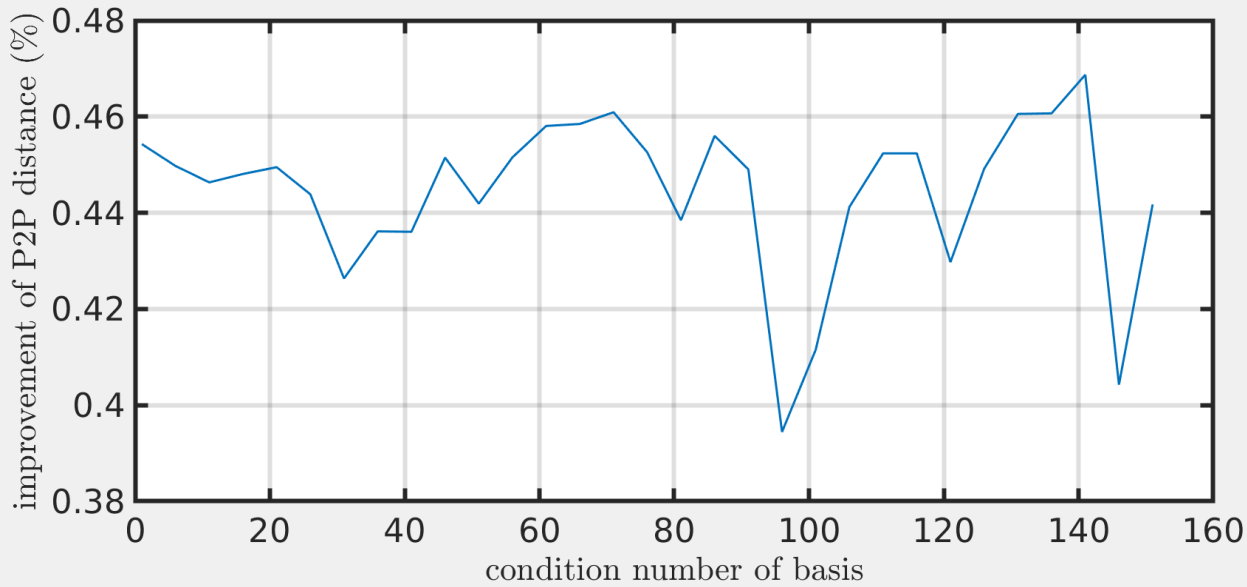


Figure 4.6: P2P cost improvement is insensitive to condition number of the basis. A condition number is always greater or equal to 1. Its reciprocal is a numerical measure of independence. The smaller the condition number, the better (greater) the independence of the target planes. The figure illustrates validation results for ten thousand orientations of four targets arranged as a tetrahedron. Each orientation is a calibration scene from which a set of calibration parameters is obtained. The calibration parameters are then applied to another complex scene for validation. This figure shows that regardless of the orientation of the tetrahedron, we are able to reduce the P2P cost by 44.7% on average.

Remark 9. *One may argue that the above is a standard optimization problem in that other solvers, such as Google ceres or g2o, can easily solve it. However, they are local solvers and will suffer if a good initial guess is not provided.*

4.6 Simulation and Experimental Results

This section first presents a simulation study of the intrinsic calibration problem that will show the importance of Theorem 1 for eliminating over parameterization of the proposed model. Results here will then be used to inform the experimental work. All experiments are conducted with a *32-Beam Velodyne ULTRA Puck LiDAR*, mounted on an in-house designed torso for a Cassie-series bipedal robot [46]. All the optimizations related to the proposed methods are solved via the proposed algorithm (see Algorithm 1). As for the two baselines, we implemented them and solved via the `fmincon` function in MATLAB. All the methods, simulator, solver, and datasets are open-sourced. All the methods, simulator, solver, and datasets are open-sourced: [86–88].

4.6.1 Simulated LiDAR Environment

Due to lack of ground truth, we built a LiDAR simulator to compare our calibration method against the two baselines and to illustrate the role of target positioning. Our simulator can model

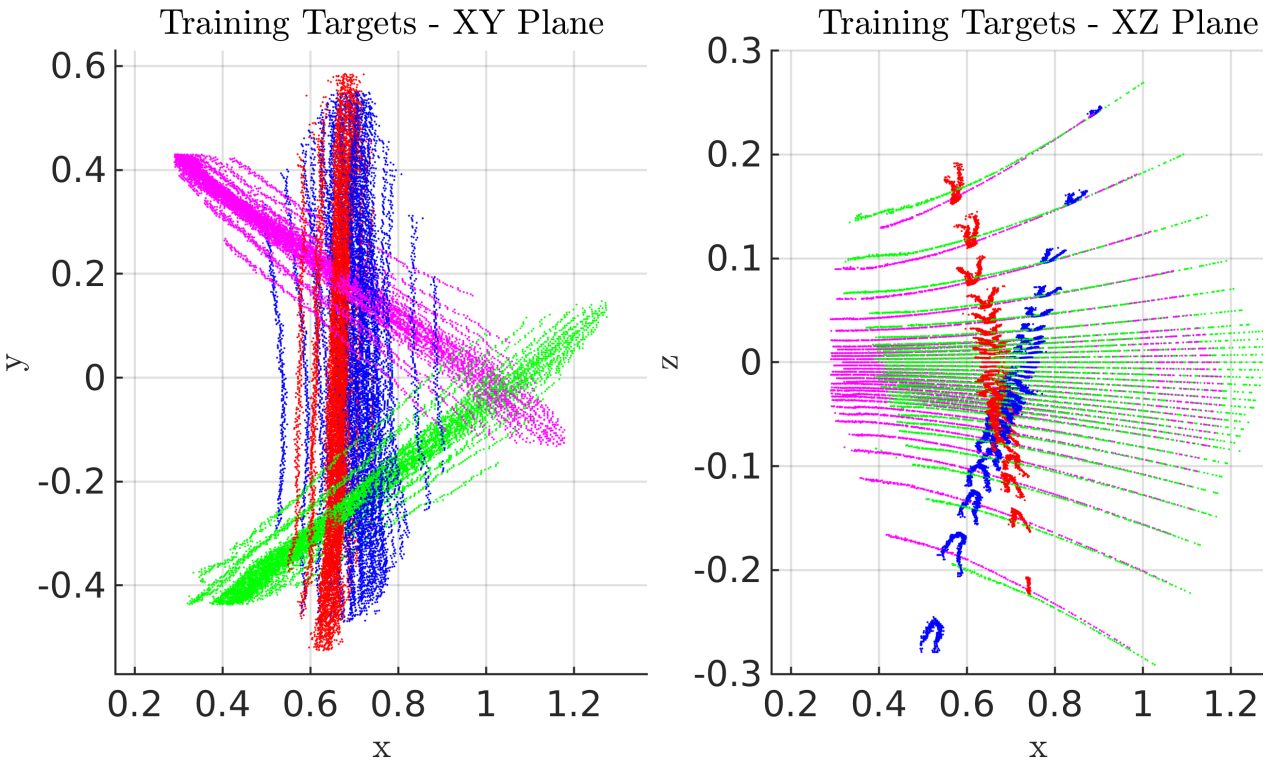


Figure 4.7: Four targets are placed to form a tetrahedron. The left shows the top-down view (X-Y plane) of a single scan of the point cloud from the calibration scene and the right shows the side view (X-Z plane) of the scene. Different colors stand for different targets.

LiDAR sensors of different working principles (spinning vs solid-state). With our simulator, it is easy to control sources of uncertainty, including both mechanical model uncertainty and measurement noise. Targets are assumed to be planar and polygonal.

To simulate 3D points on a planar target, we generate rays from the LiDAR sensor, and define a "target" point as the point at which the ray intersects the target. The simulator has an option to account for shadowing or not. After locating the exact LiDAR returns on the target, based on the LiDAR type, we then add two types of uncertainty to the returns and report them as measured data.

Remark 10. *The simulator first finds all intersection points with the (infinite) plane defined by the target. To determine if a point on the plane is within the boundary of the target polygon is a well-known problem in computer graphics, called the Point-In-Polygon (PIP) problem [98, 99]. The Winding Number algorithm [100] is implemented in this simulator. If the winding number of a point is not zero, the point lies inside the boundary; otherwise it is outside.*

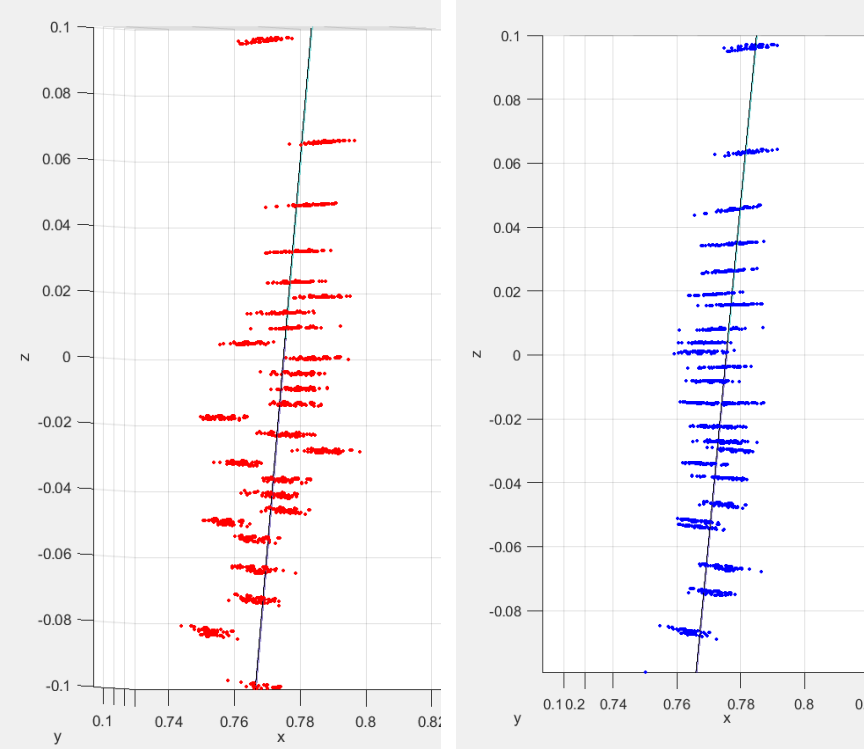


Figure 4.8: On the left shows a factory-calibrated *32-Beam Velodyne ULTRA Puck LiDAR* measuring on a planar target placed at 0.7 meter. The measurements are not consistent and lead to a thickness of 6.2 cm point cloud with 1.4 cm P2P distance. On the right shows the calibration results of the calibration parameters globally optimized over the tetrahedron calibration scene in Fig. 4.7 and then the parameters of each ring are applied to this dataset as validation. The thickness and P2P distance after calibration are 4.4 cm and 0.47 cm.

4.6.2 Intrinsic Calibration of Spinning LiDARs

4.6.2.1 Illustration of Theorem 1 in simulation

We randomly create 32 deterministic noise sequences (of length appropriate for each ring) and apply them to the ideal measurements of each ring in the LiDAR simulator. As a calibration scene, we arrange four targets around the LiDAR in the shape of a regular tetrahedron and then minimize the cost function in (4.3). For validation, we collect simulated data from a complex scene with 24 targets at various orientations and distances, using the same uncalibrated LiDAR corrupted by the same deterministic noise. We then applied the calibration parameters to the perturbed uncalibrated LiDAR measurements and reduced the P2P distance of the validation scene by 45.43% compared to the uncalibrated P2P distance. Additionally, to study the influence of different condition numbers in Assumption B on the calibration results, we rotate the tetrahedron with various orientations (ten thousand different orientations in total, yielding ten thousand different bases). We observe that the P2P cost improvement is insensitive to the condition number of the basis. In Fig. 4.6, the simulation results show that by using the tetrahedron (regardless of the orientation) to calibrate the LiDAR, the P2P distance was reduced by 44.7% on average.

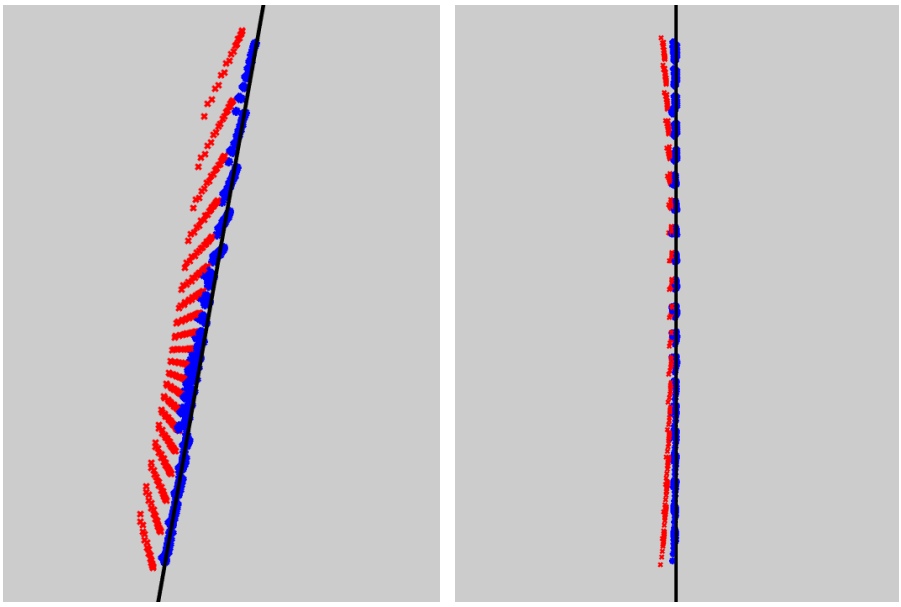


Figure 4.9: A solid-state LiDAR is (hypothetically) built on a warped wafer and measures a planar target (black). The un-calibrated measurements and calibrated measurements are marked in red and blue, respectively. The left image shows a top-down view and the right image shows a side view of a target. The proposed method reduces the P2P distance by 48.7%.

4.6.2.2 Illustration of Theorem 1 in experiments

Informed by the simulation study, two scenes (calibration and validation) of data from a *32-Beam Velodyne ULTRA Puck LiDAR* were collected using the LiDARTag package [39, 40, 101]. In Fig. 4.7, the calibration scene contains four targets[5] arranged as an oriented tetrahedron as stated in Theorem 1. The targets' normal vectors are estimated using the L1-inspired method in [9].

Figure 4.8 shows the experimental result of applying the calibration parameters to another set of targets (validation scene) placed at 0.7 meter. The proposed alternating method reduces the P2P distance by 68.6% with respect to the factory-calibrated LiDAR. Additionally, we induce varying levels of systematic errors (from 1 cm to 7 cm, corresponding to level 1 to level 7) to the experiment data to ensure the robustness of the proposed method, as shown in Table 4.1. We observe that our proposed method is insensitive to systematic error. In addition, Table 4.2 shows the consistency of the proposed method as we accumulate different numbers of scans with different noise levels. It is seen that with higher levels of induced deterministic noise, the performance of the proposed method is invariant to the number of accumulated scans, while with lower levels of deterministic noise, the more scans included, the better is the performance. Both of these properties are desirable. The number of collections K is an important hyperparameter, mentioned in Sec. 4.3.4. We group consecutive n rings into a clutter, where $n = \{1, 2, 4, 8\}$ and the results are shown in Table 4.3.

Remark 11. *There are two ways to orient the targets to form a tetrahedron:* ***1)*** *put the targets*

[5]The targets were placed a distance of 0.7 meter so that most of the rings would lie on the targets.

Table 4.1: Validation data for various calibration methods on a *32-Beam Velodyne ULTRA Puck LiDAR*. The numbers are the average P2P distance in meters. For each column, the LiDAR's calibration parameters are optimized on a common set of four targets arranged as a tetrahedron and validated on a common set of three targets arranged in different orientations and at different distances. What varies in each column is the level of systematic noise added to the raw data, as reflected in the P2P error reported in the row for Factory Calibration. Noise 0 is the raw signal from the factory-calibrated LiDAR, while Noise 1 to Noise 7 have increasing levels of systematic error. Our proposed method is insensitive to systematic error.

	Methods	Noise 0	Noise 1	Noise 2	Noise 3	Noise 4	Noise 5	Noise 6	Noise 7
P2P	Factory Calibration	0.0140	0.0196	0.0309	0.0408	0.0486	0.0581	0.0669	0.0772
	Baseline1 (3 parameters)	0.0052	0.0054	0.0064	0.0085	0.0111	0.0135	0.0154	0.0164
	Baseline2 (6 parameters)	**0.0042**	0.0059	0.0070	0.0100	0.0121	0.0148	0.0165	0.0181
	Our Method	0.0047	**0.0042**	**0.0040**	**0.0041**	**0.0043**	**0.0041**	**0.0039**	**0.0040**
Point Clouds Thickness	Factory Calibration	0.0626	0.0777	0.1182	0.1579	0.1981	0.2379	0.2788	0.3211
	Baseline1 (3 parameters)	**0.0439**	**0.0463**	0.0634	0.0817	0.0998	0.1177	0.1347	0.1501
	Baseline2 (6 parameters)	0.0457	0.0535	0.0719	0.1005	0.1110	0.1308	0.1440	0.1694
	Our Method	0.0476	0.0475	**0.0476**	**0.0582**	**0.0531**	**0.0464**	**0.0449**	**0.0450**

around the LiDAR and keep the LiDAR pose fixed; ***2)*** *keep the target poses fixed and rotate the LiDAR. The latter method will allow the targets to form a tetrahedron even if the LiDAR does not have a* 360° *field of view.*

Remark 12. *To decide if the P2P distance of a ring to a target is random noise or systematic errors, we first compute the standard deviation of the LiDAR returns on the target and consider a P2P distance greater than* 3σ *as systematic error.*

Remark 13. *The induced mechanical error is translational so that each of the calibration models is able to correct the error, specifically,* ρ *in the 3-parameter model,* s, ρ, h, v *in the 6-parameter model, and* $s, \boldsymbol{v}$ *in the proposed method. Translational errors can appear in practice if the clocking system on a LiDAR (both spinning and solid-state types) is not accurate.*

4.6.3 Intrinsic Calibration of Solid-State LiDARs

Experiments: One set each of calibration and validation of data were collected from an Intel RealSense L515. The calibration scene consists of four targets arranged as an oriented tetrahedron. The scan resolution of the Intel L515 point cloud is fixed (480×640), similar to an image. We treat each row of points as a "ring" to be calibrated. Therefore, we have 480 sets of calibration parameters. The calibration parameters are solved by the proposed alternating method. Similar to the experimental results for a spinning LiDAR, we also induce from 2 cm to 7 cm of systematic translational error in the experiment data to assess the robustness of the proposed calibration method, as shown in Table 4.4. As indicated in Remark 13, the induced translational error can appear in practice if the clocking system on the LiDAR is not accurate. The proposed unifying model slightly improves the original P2P distance and is robust to large noise.

Table 4.2: Consistency of the proposed methods. The numbers are the average P2P distance in meters and improvement of average P2P distance in percentage. The columns show the numbers of accumulated scans. It is seen that with higher levels of induced deterministic noise, the performance of the proposed method is invariant to the number of accumulated scans, while with lower levels of deterministic noise, the more scans included, the better is the performance. Both of these properties are desirable.

	Noise Level	1 scan	2 scans	4 scans	6 scans	8 scans	10 scans	12 scans
Original P2P	0	0.013998	0.013998	0.014616	0.014593	0.0144	0.01439	0.01439
Calibrated P2P		0.0047046	0.0047046	0.0040949	0.0040496	0.0040546	0.0040675	0.0040675
Improvement [%]		0.6639	0.6639	0.71983	0.7225	0.71844	0.71735	0.71735
Original P2P	2	0.030888	0.030861	0.029893	0.02983	0.029812	0.031767	0.031783
Calibrated P2P		0.0040332	0.0040739	0.0040189	0.0039899	0.0039737	0.0040665	0.0040417
Improvement [%]		0.86942	0.86799	0.86556	0.86625	0.86671	0.87199	0.87284
Original P2P	4	0.048608	0.047194	0.048691	0.050308	0.050269	0.050251	0.050254
Calibrated P2P		0.004301	0.0044295	0.004129	0.0040814	0.0040991	0.0040807	0.004067
Improvement [%]		0.91152	0.90614	0.9152	0.91887	0.91846	0.91879	0.91907
Original P2P	6	0.066834	0.066892	0.066869	0.06685	0.066823	0.066818	0.066838
Calibrated P2P		0.0043101	0.0044868	0.0044289	0.0044279	0.0044585	0.0044699	0.0044777
Improvement [%]		0.93551	0.93293	0.93377	0.93376	0.93328	0.9331	0.93301

Table 4.3: The number of cluster K is an important hyperparameter, mentioned in Sec. 4.3.4. We analyze the effect of K of the spinning LiDAR.

	Number of clusters	Noise 0	Noise 1	Noise 2	Noise 3	Noise 4	Noise 5	Noise 6	Noise 7
P2P	Factory Calibration	0.0140	0.0196	0.0309	0.0408	0.0486	0.0581	0.0669	0.0772
	Our Method (K = 1)	0.0047	0.0042	0.0040	0.0041	0.0043	0.0041	0.0039	0.0040
	Our Method (K = 2)	0.0061	0.0055	0.0058	0.0057	0.0056	0.0057	0.0061	0.0067
	Our Method (K = 4)	0.0068	0.0066	0.0073	0.0073	0.0183	0.0157	0.0182	0.0177
	Our Method (K = 8)	0.0071	0.0076	0.0099	0.0126	0.0163	0.0176	0.0185	0.0202

Simulation of a Warped Wafer: OPA solid-state LiDARs are fabricated on a planar wafer with a large number of emitters; see Sec. 4.1. For such a LiDAR, we induce geometric uncertainty into a simulation of the system by assuming the plane of the wafer is slightly warped. In the simulator, the OPA solid-state LiDAR has 20×20 emitters with $160^\circ \times 40^\circ$ (horizontal, vertical) field of view. We place four targets of the same size in the simulator.

Theorem 1 requires a ring plane, as mentioned in Sec. 4.3.3. We therefore parse the LiDAR returns into 20×4 uniform grids over the planar targets, with each grid containing five points. Each grid plays the role of a ring plane in Theorem 1, resulting in a total of 80 calibration models. We use Weighted Least Squares to estimate the plane of the target. The calibrated sensor is then validated on four other scenes that have different angles and distances. The mean P2P cost of (4.3) improves by 48.7%; the result is visualized in Fig. 4.9.

Remark 14. *The solid-state LiDAR is used to measure the same target four times with different orientations. The four target planes form a tetrahedron. Therefore, the same five points can scan the four planes and result in a ring plane. Instead of five points, we can group a row/column of points, as mentioned in Sec. 4.3.4.*

Table 4.4: Experimental results of a solid-state LiDAR. Noise 0 is the raw signal from the calibrated LiDAR, while Noise 1 to Noise 3 have increasing levels of systematic translational error.

		Noise 0	Noise 1	Noise 2	Noise 3
P2P	Original	0.00094934	0.024259	0.048401	0.07235
	Calibrated	0.00093824	0.00099253	0.00088164	0.00092039
	Improvement [%]	1.1693	95.909	98.178	98.728
Point Cloud Thickness	Original	0.007217	0.05306	0.10642	0.16318
	Calibrated	0.0066223	0.006313	0.0063051	0.0063316
	Improvement [%]	8.2401	88.102	94.075	96.12

4.7 Conclusions

We proposed a universal method for LiDAR intrinsic calibration that abstracts away the physics of a LiDAR type (spinning head vs. solid-state, for example) and focuses instead on the spatial geometry of the point cloud generated by the sensor. The calibration parameter becomes an element of $\mathrm{Sim}(3)$, a matrix Lie group. We mathematically prove that given four targets with appropriate orientations, the proposed model is well-constrained (i.e., a unique answer exists). Because $\mathrm{Sim}(3)$ is closely related to $\mathrm{SE}(3)$, we showed how to profitably apply efficient, globally convergent algorithms for $\mathrm{SE}(3)$ to determine a solution to our problem in $\mathrm{Sim}(3)$. The resulting algorithm was evaluated in simulation for a solid-state LiDAR and simulation and experiment for a spinning LiDAR. The P2P distance of the validation scene for the spinning LiDAR was reduced 44.7% and 68.6% in simulation and experiment, respectively. The point cloud thickness of the validation scene for the solid-state LiDAR was reduced 8.2% in experiment. The P2P distance of the validation scene for a warped wafer was reduced by 48.7% in simulation. Both simulation and experiments showed that the proposed method can serve as a generic model for intrinsic calibration of both spinning and solid-state LiDARs, and also is robust to different levels of systematic bias.

CHAPTER 5

Automatic Calibration Pipeline

5.1 Automatic Calibration System

Target-based LiDAR-camera extrinsic calibration methods or sensor intrinsic calibration often suffer from manual target extraction and feature association. The only manual process in our previous software release [9], determining the image corners, has now been made automatic. Because experimentally comparing and validating the intrinsic calibration methods of Sec. 4.3 and Sec. 4.4 would require significant data collection, we decided to create a fully automatic pipeline for both intrinsic and extrinsic calibration, as shown in Fig 5.1.

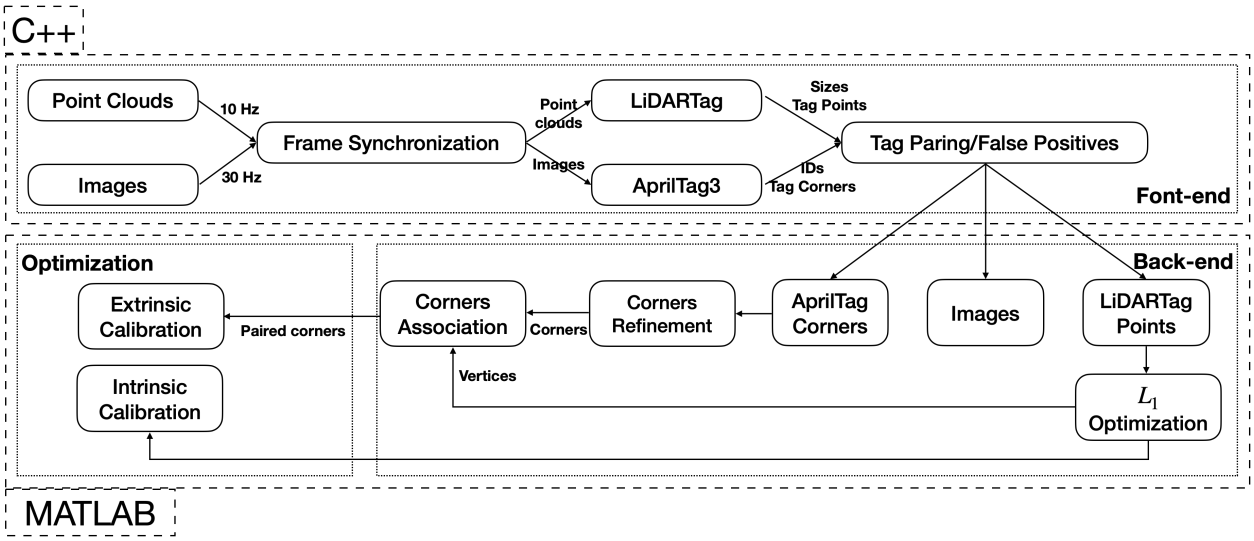


Figure 5.1: A system diagram for automatic intrinsic and extrinsic calibration. The top shows the front-end of the pipeline. Its input is raw camera and LiDAR data, which are subsequently synchronized. The AprilTag and LiDARTag packages are used to extract the target information, which is then examined to remove outliers and ensure that targets are still properly synchronized. Each run of the front-end saves all related information into a Robot Operating System (ROS) bagfile, which can then be sent to the back-end for further processing. The back-end takes in (possibly many) ROS bagfiles and does the following: (i) refines the image corners; and (ii) extracts vertices of the LiDAR targets. The correspondences are established between corners and vertices and an extrinsic transformation is determined PnP as in [60]. For intrinsic calibration, the resulting vertices of the LiDAR targets are used to extract normal vectors and a point on the plane. The calibration parameters are determined to minimize the P2P distance from the plane to the target points provided by the LiDARTag package.

5.1.1 Front-End

Fiducial markers are widely used in camera-based perception tasks because they can be easily identified and automatically extracted from the background. We introduce a similar concept for LiDAR point clouds, called a LiDARTag (See Chapter 6), consisting of a payload and a pattern that makes each marker uniquely distinguishable and identifiable in real-time by both a LiDAR and a camera [39]. An open-source package for processing LiDARTags is available [101], while AprilTag3 is available as a ROS package [102].

As LiDAR and camera data streams arrive, they are synchronized, examined for fiducial markers, and checked for false positives. For LiDAR-camera extrinsic calibration, we use target vertices and image corners as features. The relevant data (images, AprilTag corners, points on LiDARTags) are saved as ROS bagfiles for processing in MATLAB. Technical details and software are available at [103].

5.1.2 Back-End

The back-end begins by estimating the LiDARTag vertices, which are not directly observable due to the sparsity of the LiDAR point cloud. We use the method developed in Sec. 3.2 to determine the vertices of a diamond-shaped target. A rigid-body transformation is determined that "best fits" the target's LiDAR points to the target's known geometry. Best fit is defined in an L_1 sense.

From the vertices, the target normal $\mathbf{n}_t$ and point on the target, $\mathbf{p}_{0,t}$, can easily be determined for evaluating the P2P cost function (4.3) for intrinsic calibration, while the vertices in combination with the camera corners are used to determine the extrinsic calibration via a PnP problem.

The corners obtained from the AprilTag package are automatically refined using a process detailed in Sec. 3.3. Given the camera corners, denoted ($\{Y_i\}_{i=1}^4$), and LiDAR vertices, denoted $\{X_i\}_{i=1}^4$, we use the vertical and horizontal positions in their own coordinates to sort them and establish the correspondences.

5.1.3 Extrinsic and Intrinsic Calibration

For extrinsic calibration, the rigid body transformation between LiDAR to camera is found by solving a PnP problem as in [60], namely

$$(\mathbf{R}_L^{C*}, \mathbf{t}_L^{C*}) = \arg\min_{(\mathbf{R},\mathbf{t})} \operatorname{dist}(P(\mathbf{R}X_i + \mathbf{t}), Y_i), \tag{5.1}$$

where X_i are coordinates of the LiDAR vertices, Y_i are the coordinates of the camera corners, and P is the standard 3D→2D camera "projection map", based on its intrinsic parameters. Furthermore, $\mathbf{R}_L^C, \mathbf{t}_L^C$ is the LiDAR-frame to camera-frame rigid body transformation, and dist is a distance or

error measure (typically L_2). For example results, see Sec. 3.5.

With the method in Sec. 3.2, the target vertices are automatically co-planar and hence uniquely define a unit normal vector and a point on the target. The P2P distance is minimized for each of the three calibration models as mentioned in Sec. 4.4, **BL**1, **BL**2, and $\mathrm{Sim}(3)$, for a collection of targets, yielding α^* (for **BL**1 and **BL**2) and $\mathbf{H}_*$ for $\mathrm{Sim}(3)$.

Part II

Perception for Pose Estimation with Global Solution

CHAPTER 6

LiDARTag: A Real-Time Fiducial Tag System for Point Clouds

6.1 Introduction

Artificial landmark systems, referred to as *fiducial markers*, have been designed for automatic detection via specific type of sensors such as cameras [30–35]. The marker usually consists of a *payload*, that is, a pattern that makes individual markers uniquely distinguishable, and a boundary surrounding the payload that is designed to assist with isolating the payload from its background. Their supporting algorithms typically consist of modules to detect the marker, decode its payload, and estimate the pose. Such artificial landmarks have been successfully used in computer vision, Augmented Reality (AR) [32] and SLAM [104].

Images are sensitive to lighting variations, and therefore, visual fiducial markers rely heavily on the assumption of illumination consistency. As such, when lighting changes rapidly throughout a scene, the detection of visual markers can fail. Alternatively, LiDARs are robust to illumination changes due to the active nature of the sensor. In particular, rapid changes in the ambient lighting do not affect the detection of features in point clouds returned by a LiDAR. Unfortunately, however LiDARs cannot detect the fiducial markers designed for cameras. Hence, to utilize the advantages of both sensor modalities, a new type of fiducial marker that can be perceived by both LiDARs and cameras is required. Such a new marker can enable applications such as multi-sensor fusion and calibration tasks involving visual and LiDAR data [9]. Designing such fiducial markers is challenging due to inherent LiDAR properties such as sparsity, lack of structure and the varying number of points in a scan. In particular, the fact that an individual LiDAR return has no fixed spatial relation to neighboring returns makes it difficult to isolate a fiducial marker within a point cloud.

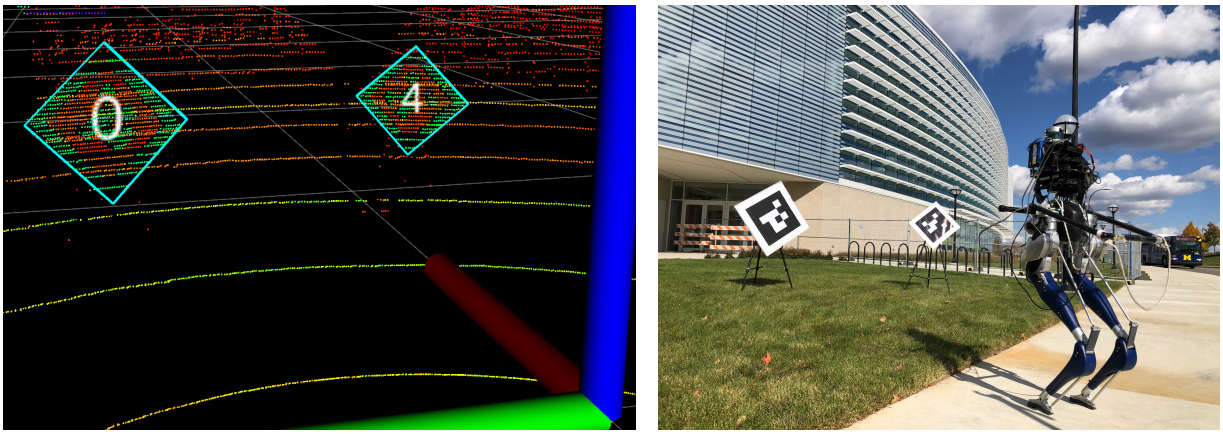


Figure 6.1: LiDAR-based markers can be used in tandem with camera-based markers to address the issue of images being sensitive to ambient lighting. This figure shows a visualization of LiDARTags of two different sizes in a full point cloud scan.

In this chapter, we propose a novel and flexible design of a fiducial tag system, called LiDARTag, as shown in Figure 6.1. A system is further developed to detect LiDARTags with various sizes and to estimate their poses. Point clouds are represented as functions in an RKHS [105] to decode their IDs. The whole system can run in real-time (over 100 Hz), which is even faster than currently available data rates of LiDAR sensors. The proposed LiDARTag can be perceived by both RGB-cameras and point clouds. In [9, 62], the authors used LiDARTags in a LiDAR-camera calibration pipeline as a means to extract the LiDAR returns on the tag. The LiDARTag system was not used to decode a payload nor to estimate pose. LiDARTags have been successfully used for LiDAR-camera calibration [9, 62]. It can be further applied to SLAM systems for robot state estimation and loop closures. Additionally, it can help improve human-robot interaction, allowing humans to give commands to a robot by showing an appropriate LiDARTag. However, the proposed system utilizes the intensity measurement of a LiDARTag to decode its ID. Therefore, LiDARs with stable (good) intensity readings are required. In particular, the present work has the following contributions:

1. We propose a novel and flexible fiducial marker for point clouds, LiDARTag, that is compatible with existing image-based fiducial marker systems, such as AprilTag.

2. We develop a robust real-time method to estimate the pose of a LiDARTag. The optimal pose estimate minimizes an L_1-inspired fitting error between the point cloud and the marker's template of known geometry.

3. To address the sparsity of LiDAR returns, we lift a point cloud to a continuous function in an RKHS and use the inner product structure to determine a marker's ID among a pre-computed function dictionary.

4. We present performance evaluations of the LiDARTag where ground truth data are provided

by a motion capture system. We also extensively analyze each step in the system with spacious outdoor and cluttered indoor environments. Additionally, we report the rate of false positives validated on the indoor Google Cartographer [41] dataset and the outdoor Honda H3D dataset [42].

5. We provide open-source implementations for the physical design of the proposed LiDARTag and all of the associated software for using them, in C++ and ROS [81]; see https://github.com/UMich-BipedLab/LiDARTag [101].

The remainder of this chapter is organized as follows. Section 6.2 presents a summary of the related work. Section 6.3 explains the tag design. Tag detection and pose estimation are discussed in Section 6.4 and Section 6.5. The construction of continuous functions and ID decoding are introduced in Section 6.6. Experimental evaluations of the proposed LiDARTags are presented in Section 6.7. Finally, Section 6.8 concludes the chapter and provides suggestions for future work.

6.2 Related Work

Fiducial marker systems were originally developed and used for augmented reality applications [32, 33] and have been widely used for object detection and tracking and pose estimation [106]. Due to their uniqueness and fast detection rate, they are also often used to improve SLAM systems [104]. Because of automatic detection, the camera-based markers are often used to extract features for cameras in target-based LiDAR-camera calibration [9, 107, 108], and the proposed algorithm can provide a means to extract features for LiDARs. To the best of our knowledge, there are no existing fiducial markers for point clouds. Among the many popular camera-based fiducial markers are ARToolKit [32], ARTag [33], AprilTag 1-3 [30, 31, 34], and CALTag [36].

In the following, we review some recent and well-known fiducial markers for cameras. ARTag [33] [37] uses a 2D barcode to make decoding easier. AprilTag 1-3 [30, 31, 34] introduced a lexicode-based [109] tag generation method in order to reduce false positive detection. ChromaTag [35] proposes color gradients to speed up the detection process. RuneTag [110] uses rings of dots to improve occlusion robustness and more accurate camera pose estimation. CCTag [111] adopts a set of rings to enhance blur robustness. More recently, LFTag [38] has taken advantage of topological markers, a kind of uncommon topological pattern, to improve longer detection range. This also enables the decoding of markers with high distortion, and these markers can be flexibly laid down. While there are some fiducial markers using deep learning technique [112, 113], to date, all of those detectors, still only work on cameras.

There are several deep-learning-based object detection architectures for LiDAR point cloud. Most of the methods for 3D object detection deploy a voxel grid representation [114–116]. Recently, [115, 117, 118] have sought to improve feature representation with 3D convolution networks, which

require expensive computation. Similar to proposed methods for 2D objects [119–123], the proposed methods for 3D objects generate a set of 3D boxes in order to cover most of the objects in 3D space. However, most detectors are limited to specific categories and none of these detectors or proposed methods has adequately addressed rotation, perspective transformations, or domain adoption.

Remark 15. *As mentioned above, there exist several deep-learning-based object detectors trained on large-scale LiDAR datasets [124–126]. These detectors are trained on limited categories in a specific dataset, and if the training and testing data are not consistent, the inference process could fail. They would have to be retrained on new data in order to be viable for our LiDARTag. Another option could be to design our own detector and train on our own datasets rather than using existing detectors. However, there is no guarantee that the resulting detector would work for varied scenes spanning from a cluttered laboratory to spacious outdoor environments. Additionally, the existing detectors rely on powerful GPUs and they are thus not suitable for lightweight mobile robots. On the other hand, the detector proposed in this chapter is robust to general scenarios and achieves satisfactory results in practice. Deep-learning-based methods, however, are interesting future work and are discussed in Sec. 6.8.*

6.3 Tag Design and LiDAR Characteristics

This section describes some essential points to consider when designing and using a LiDAR-based tag system. In particular, this section addresses how the unstructured point cloud from a LiDAR results in different considerations in the selection of a marker versus those used with a camera.

6.3.1 LiDAR Point Clouds vs Camera Images

Pixel arrays (i.e., an image) from standard RGB-cameras of different resolutions are very structured, with the pixels arranged in a uniform (planar) grid, and each image having a fixed number of data points. A LiDAR returns (x, y, z) coordinates of 3D points lying on the surface of objects as well as the intensity, which relies on the reflectivity/material of the object. The reflectivity is measured at the wavelength used by the LiDAR. Some LiDARs also provide beam numbers. The resulting 3D point clouds are typically referred to as "unstructured" because:

- The number of returned points varies for each scan and for each beam. In particular, LiDAR returns are not uniformly distributed in angle or distance[1].

[1]Some LiDARs have different ring density at different elevation angles. For example, *32-Beam Velodyne ULTRA Puck LiDAR* has dense ring density between -5° and 3°, and has sparse ring density from -25° to -5° and from 3° to 15° [127]. Therefore, in a sparse region, a target may be only partially illuminated/observed.

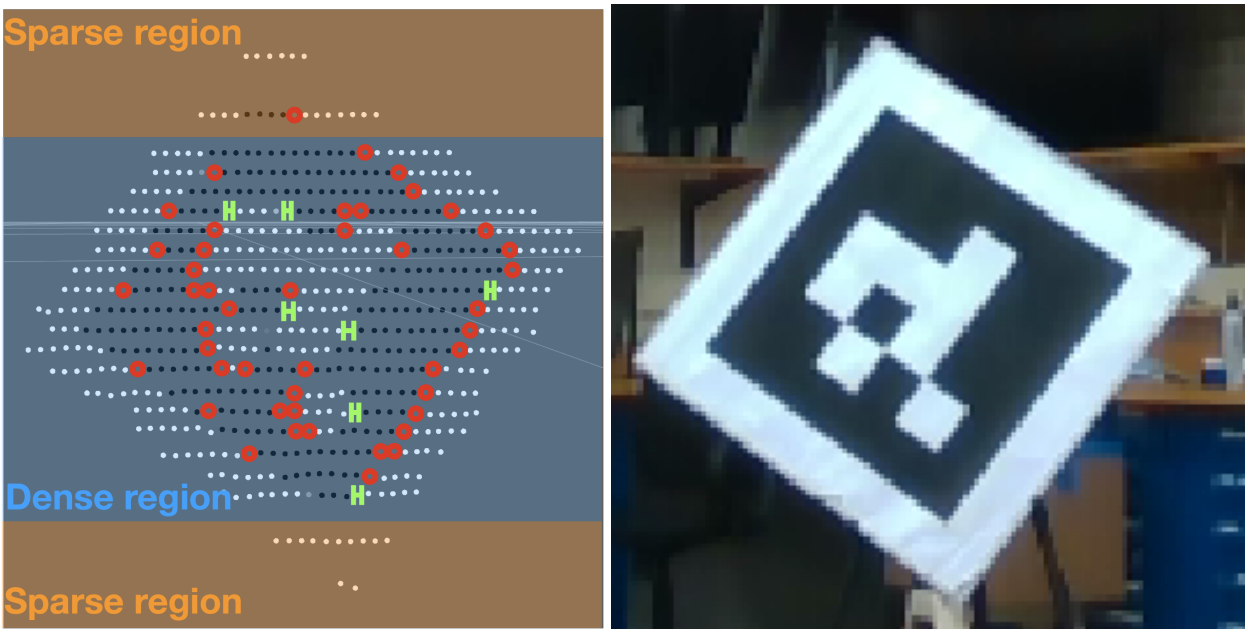


Figure 6.2: This figure illustrates the unstructured nature of a LiDAR-point-cloud return (left) for a planar surface with black and white squares (right). On the left, the black and white dots are lower reflectivity and higher reflectivity, respectively. The sparse region of the LiDAR is indicated in light yellow and the dense region is marked in light blue. The returns are more irregular at the black-white transitions. Red circles indicate missing returns and green bars highlight larger gaps between returned points.

- As shown in Fig. 6.2, high contrast between adjacent regions of a target's surface can result in missing returns and in varying spaces between returns.
- When used outdoors, the number of returned points is also influenced by environmental factors such as weather, especially temperature.

Consequently, as opposed to an image, there is no fixed geometric relationship between the index numbers of returns from two different beams in a multi-beam LiDAR. A further difference is the density of the collected data; currently, basic cameras provide many more data points for a given surface size at a given distance than even high-end LiDARs. These summarized differences have an impact on how one approaches the design of a LiDAR-based tag system vs. a camera-based tag system.

6.3.2 Tag Placement and Design

As mentioned in Sec. 6.3.1, a return from a typical LiDAR consists of an (x, y, z) measurement, an intensity value i, and a beam number (often called ring number, r). In this chapter, we propose exploiting the relative accuracy of a LiDAR's distance measurement to determine "features" when seeking to isolate a LiDARTag in a 3D point cloud (see Sec. 6.4) and associating the isolated LiDARTag points with a continuous function to decode the marker's payload (see Sec. 6.6).

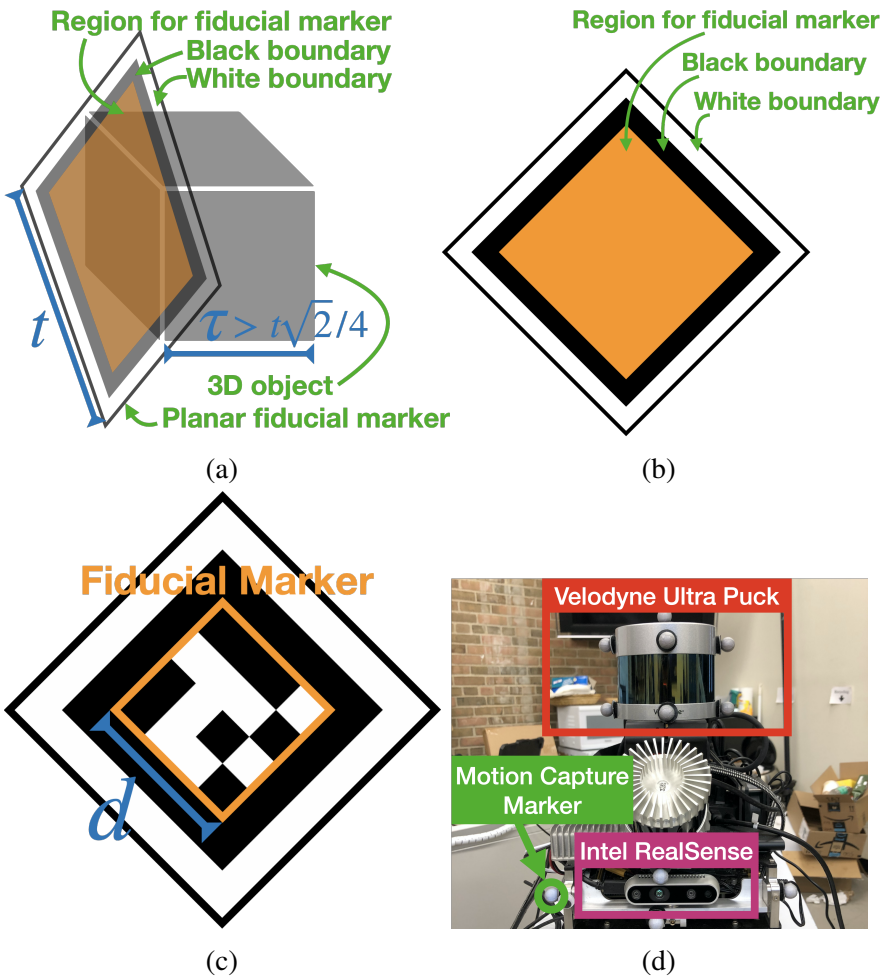


Figure 6.3: (a) illustrates a LiDARTag consisting of two parts: a 3D object with a rigidly attached, planar fiducial marker where t and h are the marker size and height of the object respectively. (b) shows the marker should be placed inside the yellow region and (c) illustrates an example of AprilTag being used as a LiDARTag. (d) is the sensor setup consisting of a LiDAR, a camera and several motion capture markers.

6.3.2.1 Tag design

A LiDARTag is assumed to consist of a planar fiducial marker rigidly attached to a 3D object, as shown in Fig. 6.3(a). In particular, the marker shows different intensity values when illuminated by a LiDAR. As indicated in Sec. 6.3.1, intensity relies on how a LiDAR measures the reflectivity of an object. Most types of fiducial markers for camera-based systems could be adapted for use in LiDAR-based systems as long as the payload is composed of differing reflectivities and is placed inside the region highlighted in yellow in Fig. 6.3(b). Figure 6.3(c) shows an example of an AprilTag used as a LiDARTag. In our experiments, we print the tags from a regular printer and a poster machine. Because fiducial markers are usually printed from a printer, however, markers with two colors such as AprilTag 1-3 [30, 31, 34], ARTag [33, 37], InterSense [128], CyberCode [129], or CALTag [36] can be most easily adapted for use in our LiDARTag system, while Cho et.al [130]

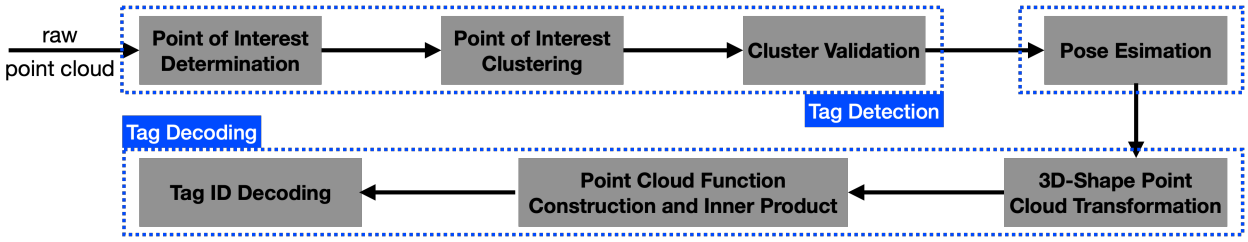


Figure 6.4: The system contains three parts: tag detection, pose estimation, and tag decoding. The detection step takes an entire LiDAR scan (up to 120,000 points from a *32-Beam Velodyne ULTRA Puck LiDAR*) and outputs collections of likely payload points of the LiDARTags. Next, a tag's optimal pose minimizes the L_1-inspired cost in (6.8), though the rotation of the tag about a normal vector to the tag may be off by $\pm 90^\circ$ or 180° and will be resolved in the decoding process. The tag's ID is decoded with a function library inspired by [105, 131]. The decoded tag removes the rotation ambiguity about the normal.

with multi-color cannot.

For this initial study, we employ AprilTag3 as our fiducial markers. Furthermore, within the AprilTag3 family of markers, we select tag16h6c5, that is, a tag encoding 16 bits (i.e., 16 black or white squares), with a minimum Hamming distance of 6, and a complexity of 5. The Hamming distance measures the minimum number of bit changes (e.g., bit errors) required to transform one string of bits into the other. For example, the length-7 strings "1011111" and "1001011" have a Hamming distance of 2, whereas "1011111" and "1001010" have a Hamming distance of 3. The significance is that a lexicode with a minimum Hamming distance h can detect $h/2$ bit errors and correct up to $\lfloor (h-1)/2 \rfloor$ bit errors [109]. The complexity of an AprilTag is defined as the number of rectangles required to generate the tag's 2D pattern. For example, a solid pattern requires just one rectangle, whereas, a white-black-white stripe would need two rectangles (first draw a large white rectangle; then draw a smaller black rectangle). For further details, we refer the reader to the coding discussion in Olson [30].

Remark 16. *One may argue that other members of the AprilTags3 family, such as tag49h15c15, tag36h11c10, tag36h10c10, and tag25h10c8, would be more appropriate. For example, including more bits tends to increase the number of distinct tags in a family, while a larger Hamming distance reduces false positives. However, for tags of the same physical size and the same distance from the sensor, more bits means fewer returns per square and a higher error rate for individual squares.*

6.3.2.2 Tag placement

In Fig. 6.3(a), let t be the tag size and h be the thickness of the 3D object. The 3D object is assumed to have $t\sqrt{2}/4$ clearance around it, and the first LiDAR ring hitting at a LiDARTag is above $3/4$ of the LiDARTag, see Sec. 6.4.2. In particular, the fiducial marker can be attached to a wall as long as the condition, $h > t\sqrt{2}/4$, in Fig. 6.3(a) is met. Finally, it is not recommended to orient the marker like a square due to the quantization error inherited in LiDAR sensors, see [9, Sec.

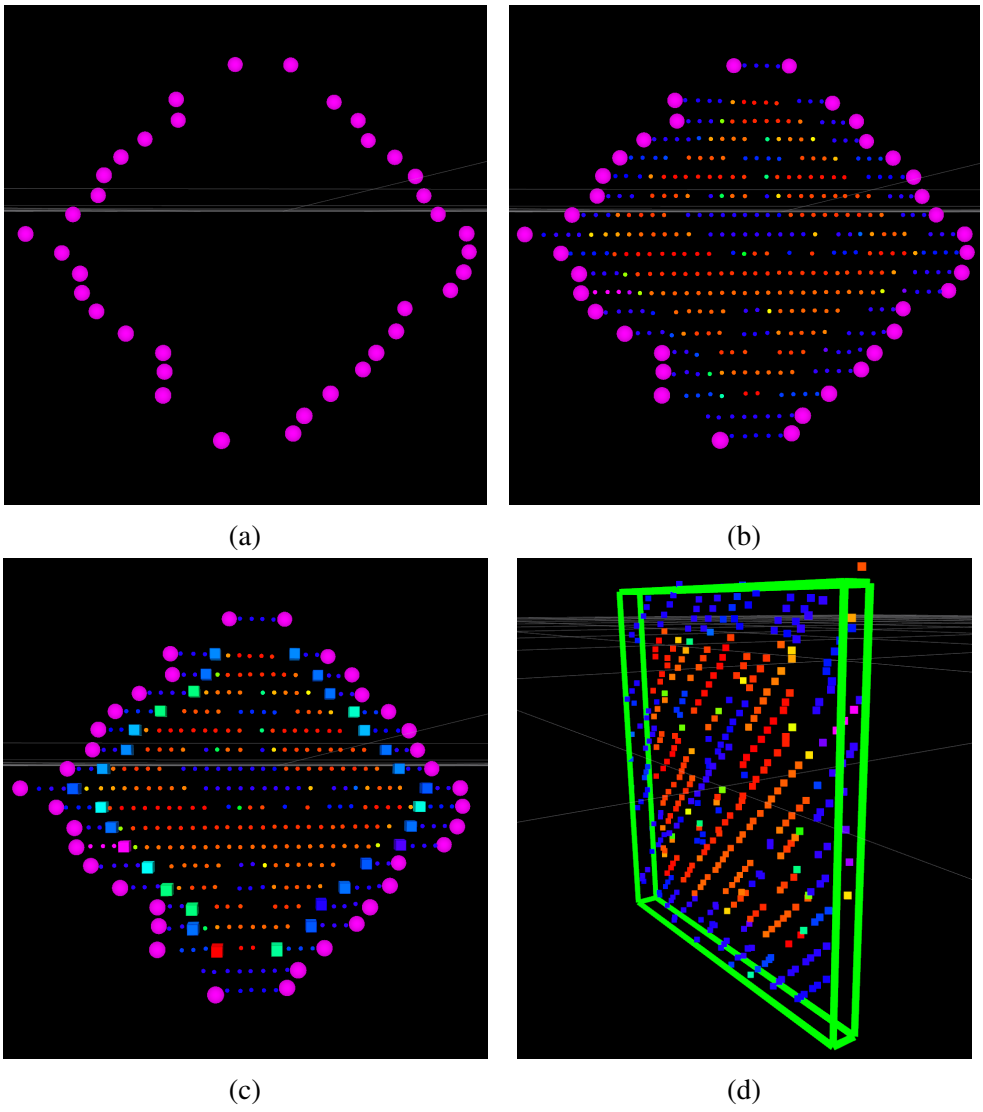


Figure 6.5: Intermediate steps of the LiDARTag system. The system takes a full scan point cloud and applies feature detection as defined in (6.1) and associates the features into clusters (magenta spheres), as shown in (a). Using the features, the clusters are filled from the original scan (different color dots stand for different intensity values) as shown in (b). Later, boundary points (indicated by boxes) are detected. After validating all the clusters, (d) shows the result of the point cloud of a LiDARTag pulled back to the LiDAR origin by H_T^L, which is a rigid-body transformation from the tag to the LiDAR that minimizes an L_1-inspired fitting error (6.8). The green box is the template of the fiducial marker.

II].

6.4 Tag Detection

This section provides the individual steps for detecting potential markers and examining their validity. The overall pipeline is shown in Fig. 6.4. To localize potential LiDARTags within the point cloud, the first step is to find features. The features are then grouped into distinct *clusters*[2]. Most of

[2]Clustering features instead of clustering directly on a LiDAR scan is critical to achieve real-time applications.

these clusters will not contain tags. Therefore, it is essential to validate whether a cluster contains a LiDARTag or not.

6.4.1 Feature Detection

As mentioned in Sec. 6.3.1, images are very structured in that the vertical and horizontal pixel-pixel correspondences are known. Consequently, various kinds of 2D kernels [66] can be applied for edge detection. However, unlike images, raw LiDAR point clouds are unstructured in that even if we have the indices of all points in each beam, we do not know the vertical point-point correspondences.

Therefore, to find a feature in a point cloud, an edge point is defined as discontinuities in distance. Inspired by the point selection method in LeGO-LOAM [132], a point is defined as a feature if it is an edge point, and its consecutive n points are not edge points (similar to plane features in LeGO-LOAM). Given consecutive $n+1$ points, we choose to use a 1D kernel to compute spatial gradients at each point to find edge points. Let $p_{i,m}$ be the i^{th} point in the m^{th} beam so that the gradient of distance $\nabla D(p_{i,m})$ can be defined as

$$\nabla D(p_{i,m}) = \|p_{i+l,m} - p_{i,m}\|_2 - \|p_{i-l,m} - p_{i,m}\|_2, \tag{6.1}$$

where ℓ is a design choice (here, $\ell = 1$). If $\nabla D(p_{i,m})$ at a point exceeds a threshold ζ, we then consider $p_{i,m}$ as a possible edge point. Using distance gradients requires a LiDARTag being ζ away from the background. For speed in real time application, we do not apply further noise smoothing, edge enhancement, nor edge localization. Finally, if there is only one edge point in the consecutive $n+1$ points ($n = 2$ in this chapter), then the edge point is considered as a feature.

6.4.2 Feature Clustering

After determining the features in the current point cloud, we group them into clusters using the single-linkage agglomerative hierarchical clustering algorithm[3] [133, 134]. As indicated in Sec. 6.3.1, LiDAR returns are not uniformly distributed in angles or distance. The linkage criteria, therefore, considers the x-y axes and the z-axis differently: signed Manhattan distance is chosen for the x-y axes, and ring numbers are selected for the z-axis. Similarly, boundaries of a cluster are defined by the four center points of a cuboid's faces and maximum/minimum ring numbers, as shown in Fig. 6.6. The algorithm loops over each feature, either linking it to an existing cluster and updating its boundaries, or creating a new cluster.

[3]We chose this clustering algorithm because the number of LiDARTags is unknown. Therefore, algorithms like K-Means Clustering cannot be used.

Remark 17. *LiDAR rings are determined by the elevation angle of an emitter. Most existing LiDARs provide not only (x, y, z, i) values, but also a ring number of a data point. If ring numbers are not available and there exists only one rotation axis in the LiDAR, a ring number can be simply regressed against elevation angle by taking the LiDAR's ring numbers as a discrete set of corresponding elevation angles, in which the number of elevation angles is the same as the number of beams of the LiDAR. The elevation angle of a data point can be computed as*

$$\arctan\left(\frac{z}{\sqrt{x^2+y^2}}\right).$$

We use this method to regress ring numbers for the Google Cartographer dataset. For more detail, see our implementation on GitHub [101].

The boundaries $(b_1, \cdots, b_4, r_{max}, r_{min})$ of a cluster are the maximum/minimum x, maximum/minimum y and maximum/minimum ring numbers among all features in the cluster. When a new cluster is created from a feature $p^k = (x, y, z, i, r)$, the four center points of the faces are defined as $(x \pm \tau, y \pm \tau)$, with $\tau = t\sqrt{2}/4$ for the $(b_1, \cdots, b_4)$. The r_{max} and r_{min} are defined in terms of the ring numbers r, as shown in Fig. 6.6. The linkage criteria $L(p^k, c_j)$ between a feature p^k and a cluster c_j is

$$\begin{cases} \min(b_i^k - \tau) \leqslant p^k \leqslant \max(b_i^k + \tau),\ \forall i = 1, \cdots, 4 \\ r_{min} - 1 \leqslant r \leqslant r_{max} + 1, \end{cases} \tag{6.2}$$

where the first line is the signed Manhattan distance for the x-y axes and the second is the ring number for the z-axis. If both conditions are met, the feature is linked to the cluster. The corresponding boundaries are updated, if necessary. Due to this linkage criteria, a LiDARTag requires $\tau = t\sqrt{2}/4$ clearance around it to avoid false linkage, and based on preliminary testing, we also impose that the topmost beam on the LiDARTag should be above $3/4$ of the target, as indicated by the red region in Fig. 6.7(b).

Remark 18. *We chose not to use a k-d tree structure [135] because the number of features and the resulting clusters are not large enough to benefit from the data structure. The construction time of the tree could overtake the querying time.*

6.4.3 Cluster Validation

At this point, we have grouped the features into clusters as shown in Fig 6.5(a). In practice, few (possibly none) of the clusters will contain a valid tag and thus it is important to be able to eliminate

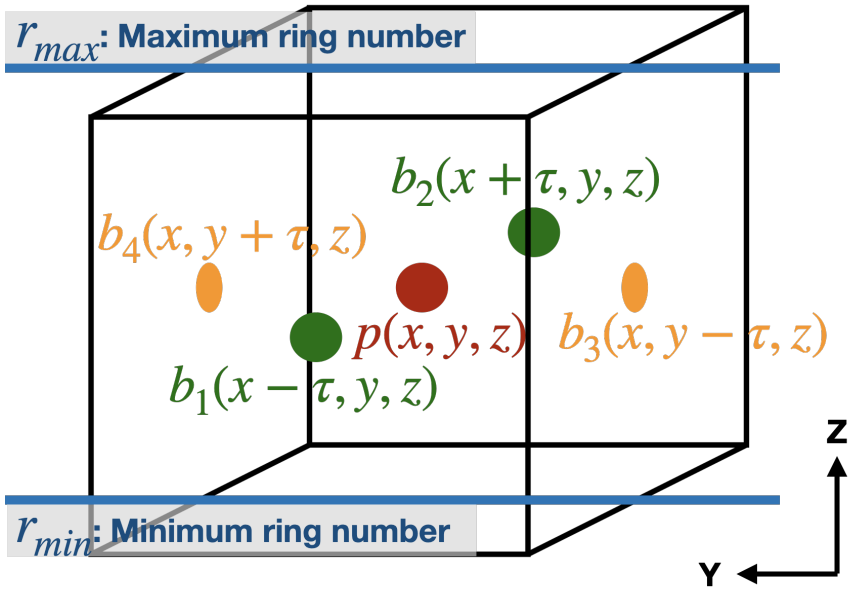


Figure 6.6: This figure shows the initial state of a cluster, in which has only single one feature. A cluster is defined as a cuboid in $\mathbb{R}^3$. When a feature fails at linkage, a cluster will be created and centered at itself (x, y, z) with four boundaries $(x \pm \tau, y \pm \tau, z)$ for the x-y axes, where τ is $t\sqrt{2}/4$, as well as the maximum ring number and the minimum ring number for the z-axis.

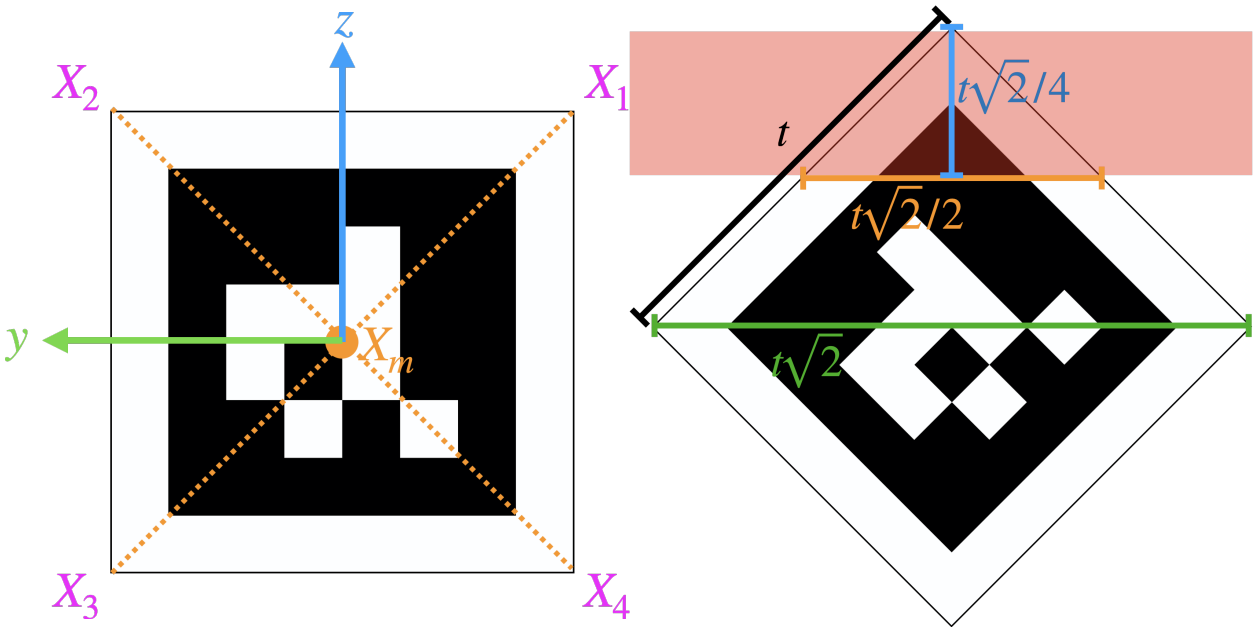
clusters that are clearly invalid. To do so, we first used the point cloud data to fill in LiDAR returns between the features of a cluster, as shown in Fig. 6.5(b).

Inspired by AprilTag [30, 31], tag-family-based heuristics are used to validate a cluster: number of points η and number of features ψ in the cluster. Another geometry-based heuristic is also deployed: the outlier percentage (κ) of a plane fitting process. To save computation time, if any of the above processes fails, the cluster is marked as invalid and does not proceed to the next stage of validation. The first two values are determined by what type of tag family is chosen. Shown in Fig. 6.3(c) is a tag family which contains 16 bits.

A lower bound on the number of points in a cluster is determined by how many bits are contained in the fiducial marker. If the payload is $d \times d$, the LiDARTag including boundaries is $(d+4) \times (d+4)$. If we assume a minimum of five returns for each bit in the tag, then the minimum number points in a valid cluster is

$$\eta \geqslant 5(d+4)^2. \tag{6.3}$$

On the other hand, an upper bound is determined by the distance from the LiDAR to the marker and its size t. Given a tag at distance D, the maximum number of returns on the marker happens



(a) (b)

Figure 6.7: (a) describes the coordinate system of the fiducial marker. (b) indicates that the first beam hitting the LiDARTag should be at least $3/4$ above target, outlined as the red region.

when it directly faces to the LiDAR and can be computed as:

$$M \frac{t\sqrt{2}}{D \sin \theta}, \tag{6.4}$$

where θ is the horizontal resolution of the LiDAR, M is the number of rings hitting on the tag, and $t\sqrt{2}$ is the diagonal length of the tag, as shown in Fig. 6.7(b).

The boundaries of the payload can be detected by an intensity gradient,

$$\nabla I(p_{i,m}) = |p^I_{i+\ell,m} - p^I_{i,m}| - |p^I_{i-\ell,m} - p^I_{i,m}|, \tag{6.5}$$

where ℓ is also a design choice (here, taken as one), and $p^I_{i,m}$ is the intensity value of the i^{th} point on the m^{th} ring. An example of detected boundary points is shown in Fig. 6.5(c). If $\nabla I(p_{i,m})$ exceeds a threshold, then $p_{i,m}$ is a payload edge point. To successfully decode a tag, we will need at least one ring on each row of the payload. Hence, the minimum number of payload edge points is

$$\psi \geqslant 2(d+2). \tag{6.6}$$

Finally, we apply a plane fitting process to the remaining clusters. If the percentage of outliers of the plane fitting is more than κ (chosen as 0.05), the cluster is considered invalid.

The above heuristics allow us to extract a potential fiducial marker from the LiDARTag through

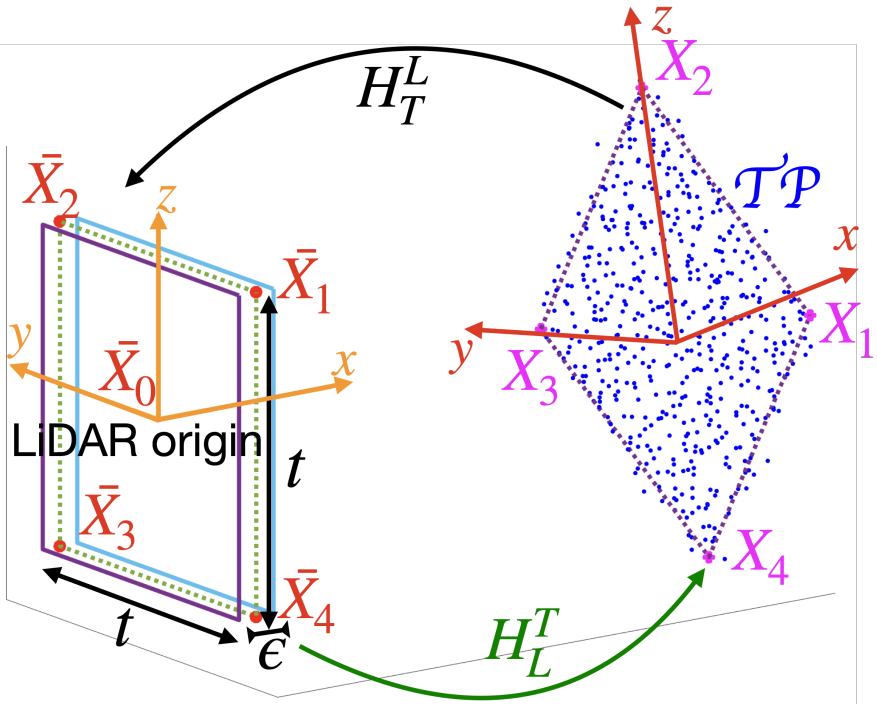


Figure 6.8: This conceptual figure illustrates the proposed method to estimate a LiDARTag's pose. The target's coordinate frame is defined as the mean of the four vertices $(X_1, \cdots, X_4)$ and the template of known geometry is defined by $(\bar{X}_1, \cdots, \bar{X}_4)$ with depth ϵ at the LiDAR origin. The rigid-body transformation H_T^L (black arrow) projects the target's point cloud to the template. The actual pose of the LiDARTag is estimated by (6.9) using the inverse transformation H_T^L (green arrow).

features in both cluttered indoor and spacious outdoor environments. The next step is to estimate the pose of the marker.

6.5 Pose Estimation and Initialization

The pose of a LiDARTag is defined as H_L^T, a rigid-body transformation from the LiDAR frame to the LiDARTag frame, as shown in Fig. 6.8. To estimate the pose, we employ the L_1-inspired method proposed in [9]. The pose estimation is formulated into an optimization problem (6.9) in Sec. 6.5.1. Due to $\mathrm{SE}(3)$ being non-convex and the requirement for a fast estimate, initial guesses to initialize the optimization problem and the gradient of the cost function are necessary, see Sec. 6.5.2.

6.5.1 LiDARTag Pose Estimation

Define the target point cloud $\mathcal{TP} := \{\mathcal{X}_i\}_{i=1}^M$ as the collection of LiDAR returns from a LiDARTag, where M is the number of points. Given the target geometry, we define a template with vertices $\{\bar{X}_i\}_{i=1}^4$ located at the origin of the LiDAR and aligned with the y-z plane as defined in Fig. 6.8. We therefore seek a rigid-body transformation from LiDAR to the tag, $H_L^T \in \mathrm{SE}(3)$, that "best fits" the template onto the LiDAR returns of the target. In practice, it is actually easier to

project the target point cloud $\mathcal{TP}$ back to the origin of the LiDAR through the inverse of the current estimate of transformation $H_T^L := (H_L^T)^{-1}$ and measure the error there. The action of $H \in \mathrm{SE}(3)$ on $\mathbb{R}^3$ is $H \cdot \mathcal{X}_i = R\mathcal{X}_i + p$, where $R \in \mathrm{SO}(3)$ and $p \in \mathbb{R}^3$. For $a \geqslant 0$ and $\lambda \in \mathbb{R}$, an L_1-inspired cost is defined as

$$c(\lambda, a) := \begin{cases} \min\{|\lambda - a|, |\lambda + a|\} & \text{if } |\lambda| > a \\ 0 & \text{otherwise} \end{cases}. \tag{6.7}$$

Let $\{\bar{\mathcal{X}}_i\}_{i=1}^N := H_T^L(\mathcal{TP}) := \{H_T^L \cdot \mathcal{X}_i\}_{i=1}^N$ denote the projected point cloud by H_T^L, and denote a point's (x, y, z)-entries by $(\bar{x}_i, \bar{y}_i, \bar{z}_i)$. The total fitting error of the point cloud is defined as

$$C(H_T^L(\mathcal{TP})) := \sum_{i=1}^{M} c(\bar{x}_i, \epsilon) + c(\bar{y}_i, d/2) + c(\bar{z}_i, d/2), \tag{6.8}$$

where $\epsilon \geqslant 0$ is a parameter to account for uncertainty in the depth measurement of the planar target and the principal axis with the smallest variance is used, see Sec. 6.5.2. The optimization problem becomes

$$H_T^{L*} := \underset{R_T^L, p_T^L}{\arg\min}\, C(H_T^L(\mathcal{TP})). \tag{6.9}$$

Finally, the pose of a LiDARTag is $H_L^T = H_T^{L*}$; see [9] for more details. To solve this optimization problem, we leverage a gradient-based solver in the NLopt library [136] and the closed form of the gradient, which is provided on our GitHub [101].

Figure 6.5(d) shows the projected returns of a LiDARTag being inside the green box (aligned with the y-z plane) at the LiDAR origin. In addition, we further compute the 2D convex hull within the y-z plane of the pullback of point cloud and utilize the surveyor's formula [72] to calculate the area of the convex hull. Our assumption on where first ring hits the marker results in at least 75% of the marker's area being illuminated. Therefore, if the estimated area is less than 75% of the marker size, the cluster is considered invalid.

Remark 19. *Equation* (6.9) *provides an estimated rigid-body transformation from the LiDAR to the tag, and importantly, due to the symmetric of the target, the rotation of the tag about a normal vector to the tag may be off by* $\pm 90^\circ$ *or* 180°. *In particular, the four rotations in Fig. 6.9 are not determined. This ambiguity will be removed after decoding the tag, see Sec. 6.6.*

6.5.2 Optimization Initialization

The corners of the LiDARTags, $(X_1, \cdots, X_4)$, are estimated from $\mathcal{TP}$. The initial guess of a rigid-body transformation is chosen to minimize the distance from the points $(\bar{X}_1, \cdots, \bar{X}_4)$ and $(X_1, \cdots, X_4)$. This will be reduced to a (constrained orthogonal) procrustes problem [137], namely

a problem of the form:

$$\Theta = \underset{\Omega:\ \Omega^T\Omega=I}{\arg\min} ||\Omega A - B||_F, \tag{6.10}$$

where for us, Ω will be a to-be-determined rotation matrix and $\|\cdot\|_F$ is the Frobenius norm.

Without loss of generality, $\bar{X_0}$ is assumed to be the origin, $(0,0,0)$ and X_0 is the mean of $\mathcal{TP}$[4]. The translation p is thus given by

$$\begin{aligned} \bar{X_0} &= RX_0 + p,\ \bar{X_0} = [0,0,0]^\mathsf{T} \\ p &= -RX_0. \end{aligned} \tag{6.11}$$

The rest of the problem can be formulated as:

$$\begin{aligned} \left\|H_T^L X_i - \bar{X_i}\right\|_2^2 &= \left\| \begin{bmatrix} R & p \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X_i \\ 1 \end{bmatrix} - \begin{bmatrix} \bar{X_i} \\ 1 \end{bmatrix} \right\|_2^2 \\ &= \left\|RX_i + p - \bar{X_i}\right\|_2^2 = \left\|RX_i' - \bar{X_i}\right\|_2^2, \end{aligned} \tag{6.12}$$

$$\begin{aligned} \sum_{i=1}^{4} \left\|H_T^L X_i - \bar{X_i}\right\|_2^2 &= \sum_{i=1}^{4} \left\|RX_i' - \bar{X_i}\right\|_2^2 \\ &= \left\| (RX_1' - \bar{X_1}) \vdots \cdots \vdots (RX_4' - \bar{X_4}) \right\|_F^2 \\ &= \left\|R\mathbf{X} - \bar{\mathbf{X}}\right\|_F^2, \end{aligned} \tag{6.13}$$

where $X_i' = X_i - X_0$, $\mathbf{X} = \begin{bmatrix} X_1' & X_2' & X_3' & X_4' \end{bmatrix}$ and $\bar{\mathbf{X}} = \begin{bmatrix} \bar{X_1} & \bar{X_2} & \bar{X_3} & \bar{X_4} \end{bmatrix}$. The problem is then

$$R^* = \underset{R:\ R^TR=I}{\arg\min} \left\|R\mathbf{X} - \bar{\mathbf{X}}\right\|_F^2. \tag{6.14}$$

By the procrustes optimization problem [137], we have a closed form solution:

$$M = \bar{\mathbf{X}}\mathbf{X}^\mathsf{T} = U\Sigma V^\mathsf{T} \tag{6.15}$$

$$R^* = UV^\mathsf{T}. \tag{6.16}$$

Remark 20. *To estimate **X**, we project the target point cloud $\mathcal{TP}$ along a principal axis of a Principal Components Analysis (PCA) [138]. Using the 2D projected point cloud, we use RANSAC to regress lines to determine target edges and solve for the intersections of the lines, to obtain an*

[4] In practice, this produces an good initial guess of translation for (6.9)

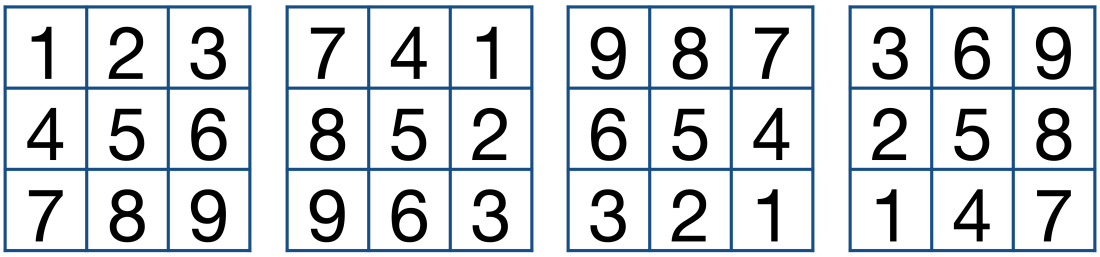


Figure 6.9: Before decoding, the estimated rotation about the normal axis is only known modulo 90°, which means (a) to (d) yield the same normal vector. Accounting for the three possible rotations of $(\pm 90^\circ, 180^\circ)$, results in 4 possible continuous functions in the function dictionary. When computing the inner product to the correct id of a LiDARTag, only one of the four functions 'zis correct. From the correct function, the modulo 90° ambiguity is removed.

initialization of the vertices. The smallest variance of the principal axis is then used for the ϵ in (6.8). *If the number of edge points is less than three, or any of edges fails when regressing a line, the cluster is marked as invalid.*

6.6 Function Construction and Tag Decoding

In Sec. 6.5.1, we defined a template at the LiDAR origin, estimated H_T^L, and we thus have the projected point cloud. Specifically, the projected point cloud is located at the LiDAR origin inside the template on the y-z plane with the thickness being the sensor noise on the x-axis. Due to the sparsity of the point cloud, we construct a continuous function in an inner product space (RKHS) for the projected point cloud [105]. For each LiDARTag in the tag family, we pre-compute four continuous functions to account for four possible rotations, as shown in Fig. 6.9, consequently, resulting in a function dictionary. Each function is constructed by converting each pixel of the tag image to a point in $\mathbb{R}^3$, see [101] for implementation detail. Finally, we compute the inner product of the estimated function and each function in the dictionary. The largest inner product is the ID of the LiDARTag, and the ambiguity of rotation in Sec. 6.5.1 is removed.

Let $\widetilde{\mathcal{X}} := \{(\widetilde{p}_i, \ell(\widetilde{p}_i)) | \widetilde{p}_i \in \mathbb{R}^3 \text{ and } \ell(\widetilde{p}_i) \in \mathcal{I}\}_{i=1}^M$ be a collection of projected points, where M is the number of points. In this work, we use the intensity as our information inner product space as $\mathcal{I} = \mathbb{R}$ and the inner product, $\langle \cdot, \cdot \rangle_{\mathcal{I}}$, is just the scalar product of intensity values. Therefore, the labels are simply the intensity values. The continuous function of $\widetilde{\mathcal{X}}$ is defined as

$$f(\cdot) = \sum_{i=1}^{M} \ell(\widetilde{p}_i) k(\cdot, \widetilde{p}_i), \tag{6.17}$$

where $k : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ is the kernel of an RKHS [105].

Given another continuous function g of point cloud $\widetilde{\mathcal{Z}} := \{(\widetilde{p}_j, \ell(\widetilde{p}_j)) | \widetilde{p}_j \in \mathbb{R}^3 \text{ and } \ell(\widetilde{p}_j) \in \mathcal{I}\}_{j=1}^N$,

where N is the number of points. The inner product of f and g is

$$\langle f, g \rangle = \sum_{i=1}^{M} \sum_{j=1}^{N} \langle \ell(\widetilde{p}_i), \ell(\widetilde{p}_j) \rangle_{\mathcal{I}} k(\widetilde{p}_i, \widetilde{p}_j). \tag{6.18}$$

The kernel k is modeled as the squared exponential kernel [139, Chapter 4]:

$$k(\widetilde{p}_i, \widetilde{p}_j) = \sigma^2 \text{exp}\left(-\frac{1}{2}(\widetilde{p}_i - \widetilde{p}_j)^{\mathsf{T}} \Lambda (\widetilde{p}_i - \widetilde{p}_j)\right), \tag{6.19}$$

where σ^2 is the signal variance (set to $1e5$) and Λ is an isotropic diagonal length-scale matrix with its diagonal entry set to the inverse of squared half of the bit size of a LiDARTag: $1/(t/(2(d+4)))^2$. Let t be the LiDARTag size, and the d-bit tag family is used ($d+4$ bits, including its boundaries). Then the bit size is $t/(d+4)$.

After applying the kernel trick to (6.18), we get [131]

$$\langle f, g \rangle = \sum_{i=1}^{M} \sum_{j=1}^{N} k_{\mathcal{I}}(\ell(\widetilde{p}_i), \ell(\widetilde{p}_j)) \cdot k(\widetilde{p}_i, \widetilde{p}_j), \tag{6.20}$$

where

$$k_{\mathcal{I}}(\widetilde{p}_i, \widetilde{p}_j) = \exp\left(-\frac{(\ell(\widetilde{p}_i) - \ell(\widetilde{p}_j))^2}{2l_{\mathcal{I}}^2}\right), \tag{6.21}$$

and the length-scale $l_{\mathcal{I}}$ is set to 10 ($0 \leqslant \ell(\widetilde{p}_i) \leqslant 255$).

Remark 21. *To fully utilize the projected point cloud of LiDARTag returns, we extend the planar LiDARTag to a 3D LiDARTag based on the intensity value of each point in the point cloud. The linear transformation is defined as:*

$$\widetilde{p}_i = \begin{bmatrix} \widetilde{x}_i \\ \widetilde{y}_i \\ \widetilde{z}_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \frac{t}{2(d+4)I_{max}} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \bar{x}_i \\ \bar{y}_i \\ \bar{z}_i \\ \bar{i}_i \end{bmatrix}, \tag{6.22}$$

where I_{max} is the maximum intensity of the point cloud and $t/(d+4)$ is the bit size.

Remark 22. *If the fitting error* (6.8) *in Sec. 6.5.1 is greater than 10% of the number of points in the cluster or it is not able to decode the potential cluster in Sec. 6.6, this cluster is marked as invalid.*

Remark 23. *Reproducing Kernel Hilbert Spaces (RKHS') have been widely used in the Representer Theorem [140–142] for various regularization problems, such as function estimation, classification and Support Vector Machines (SVMs). These are typically high- or infinite-dimensional problems*

that while mathematically feasible, often appear to be not practically computable. With the help of RKHS and the Representer Theorem, the solutions to these problems can be formulated in lower-dimensional subspaces spanned by the "representers" of the data.

6.7 Experimental Results

We now present experimental evaluations of the proposed LiDARTag. In this work, we choose an easel as our 3D object to support the tag. Additionally, fiducial markers from the tag16h6 family of AprilTag3 are used, with sizes of 1.2, 0.8, 0.61 meters, as shown in Fig. 6.3. We do not compare the proposed LiDARTag system with camera-based tag systems because it is unfair to compare depth estimation from a LiDAR with depth estimation from a monocular camera. All experiments are conducted with a *32-Beam Velodyne ULTRA Puck LiDAR* and an Intel RealSense camera rigidly attached to the torso of a Cassie-series bipedal robot as shown in Fig. 6.3(d). We use the ROS [81] to communicate and synchronize between sensors. The LiDARTag system runs faster than 100 Hz on a laptop equipped with Intel® Core™ i7-9750H CPU @ 2.60 GHz, which is similar to the processor on a robot coming to the market.

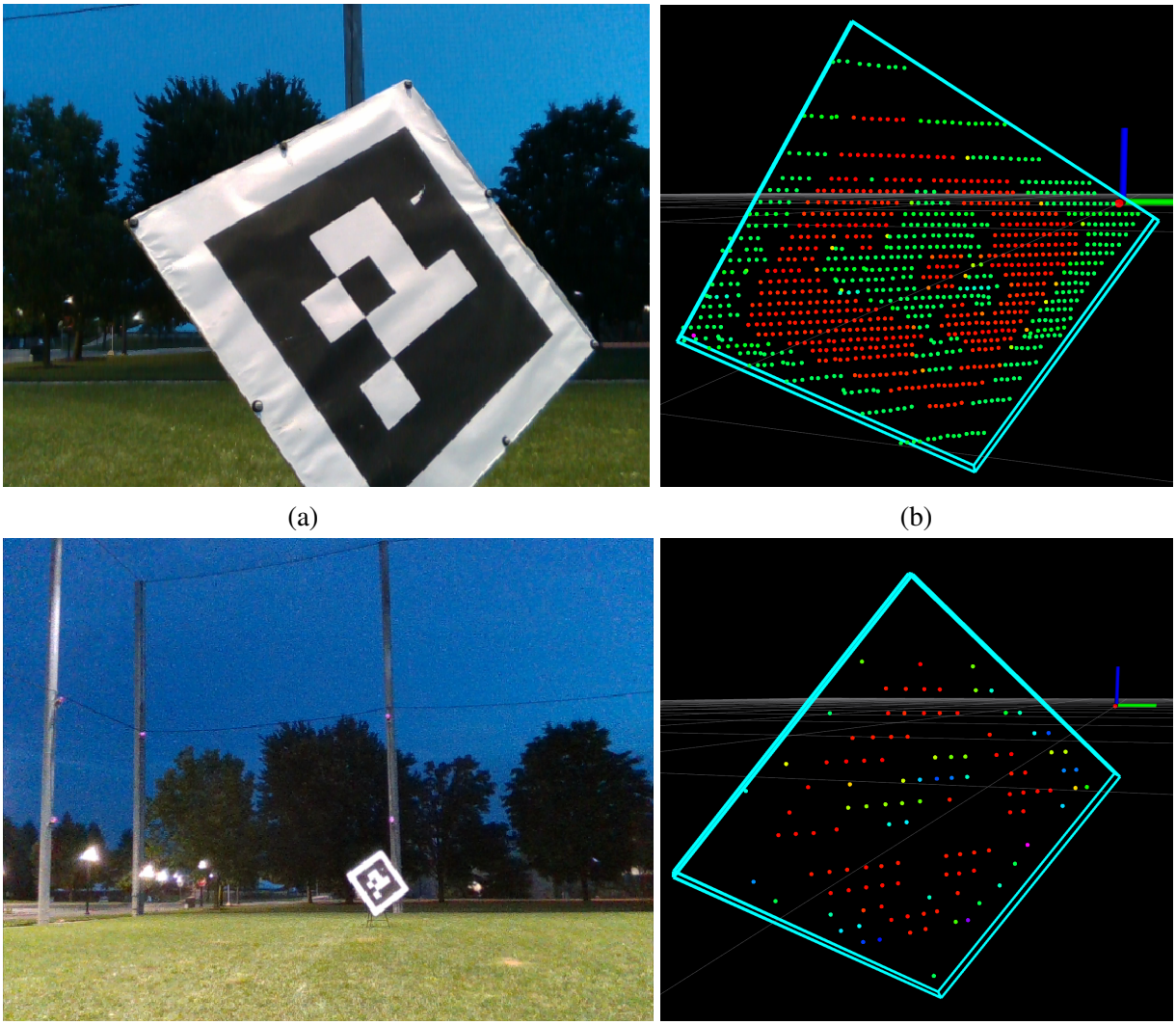
Datasets are collected in a cluttered laboratory to evaluate detection performance and a spacious outdoor facility, M-Air [143], equipped with a motion capture system to validate pose estimation and ID decoding. Additionally, false positives are evaluated on the Google Cartographer indoor dataset [41] and the outdoor Honda H3D datasets [42].

6.7.1 Pose Evaluation and Decoding Accuracy

A motion capture system developed by Qualisys is used as a proxy for ground truth poses. The setup consists of 30 motion capture cameras with markers attached to tags, a LiDAR and a camera, as shown in Fig. 6.3(d). Datasets are collected at various distances and angles. Each of the datasets

Table 6.1: Decoding accuracy of the RKHS method and pose accuracy of the fitting method. The ground truth is provided by a motion capture system with 30 motion capture cameras. The distance is in meters. The translation error is in millimeters and rotation error is the misalignment angle, (6.23), in degrees.

Face-on to LiDAR					**Rotated at 45 degrees**				
Distance	No. Scans	No. Wrong ID	Translation Error	Rotation Error	Distance	No. Scans	No. Wrong ID	Translation Error	Rotation Error
2.15	73	0	14.03	0.44	2.13	74	0	0.27	0.05
4.29	72	0	10.13	0.67	3.95	134	0	0.52	0.34
5.90	81	0	16.23	0.44	5.93	137	0	0.36	0.05
7.97	78	0	1.32	0.21	7.92	126	0	0.26	0.32
10.12	87	0	1.64	0.40	10.38	130	0	4.91	1.03
12.14	69	0	2.07	0.36	12.12	71	0	5.78	0.39
13.87	35	1	2.81	10.48	14.08	49	2	1.98	15.92
Summary	No. Scans	Wrong ID Ratio	Translation Error	Rotation Error	**Summary**	No. Scans	Wrong ID Ratio	Translation Error	Rotation Error
mean	70	0.202 %	6.891	2.149	**mean**	103	0.276 %	1.744	2.586
std	16.88	–	6.418	4.577	**std**	37.25	–	2.076	5.888
median	73	–	2.81	0.44	**median**	126	–	0.52	0.34



(a) (b)

(c) (d)

Figure 6.10: (a) and (c) are images of the tag placed at 2 and 14 meters away from a Cassie-series robot. (b) and (d) describe the results of projecting the template (green box) from the LiDAR origin to the tag's returns by the poses of LiDARTag at 2 and 16 meters, respectively. While (d) shows much sparser LiDAR returns than (b) due to the farther distance, we are still able to accurately estimate the pose and its ID. Compared to ground truth provided by 30 motion capture cameras, the resulting poses are a few millimeters off in translation and a few degrees off in rotation.

contains images (20 Hz) and scans of point clouds (10 Hz). The 1.2-meter target is placed at distances from 2 to 14 meters in 2 meter increments. At each distance, data is collected with a target face-on to the LiDAR and another dataset with the target rotated by 45 degrees. More results and videos are on GitHub, see [101].

The optimization problem in (6.9) is solved with the Method of Moving Asymptotes (MMA) algorithm [144, 145] provided in NLopt library [136]. We use the optimized LiDARTag pose to project the template at the LiDAR origin onto the LiDARTag's returns to show the qualitative results of pose estimation. Figure 6.10(b) and Figure 6.10(d) show the pose of a tag at 2 meter and 16 meter,

Table 6.2: This table averages all the datasets we collected and describes computation time of each step for indoors and outdoors.

Outdoor			
No. Points	No. Features	No. Clusters	Total Computation
51717	2179	271	114.86 Hz
PoI Clustering	Fill In Clusters	Point Check	Plane Fitting
2.63 ms	0.3 ms	0.00 ms	0.27 ms
Line Fitting	PCA	Pose Optimization	Tag Decoding
0.01 ms	0.03 ms	0.48 ms	3.34 ms
Indoor			
No. Point Cloud	No. Features	No. Clusters	Total Computation
54277	1820	225	102.41 Hz
PoI Clustering	Fill In Clusters	Point Check	Plane Fitting
3.28 ms	0.22 ms	0.00 ms	0.15 ms
Line Fitting	PCA	Pose Optimization	Tag Decoding
0.01 ms	0.01 ms	0.42 ms	2.47 ms

respectively. Even though the farther marker has much sparser LiDAR returns, by lifting the returns to an RKHS space, we are capable of correctly identifying its ID. In particular, a 1.2-meter target placed at 16 meters and rotated by 45 degrees is the detection limit of our *32-Beam Velodyne ULTRA Puck LiDAR*. However, for a LiDAR with a different number of beams or points, the detection limit is subject to change. The more beams or points, the greater is the detection range. Using our LiDAR, the detectable angle is a little smaller than camera-based marker due to the sparse point clouds. Table 6.1 compares quantitatively the pose estimation between the proposed LiDARTag and ground truth. The translation error is reported in millimeters, and rotation error ξ is represented as geodesic distance [5] in degrees [146]:

$$\xi = \|\mathrm{Log}(\bar{R}\widetilde{R}^{\mathsf{T}})\|, \tag{6.23}$$

where $\|\cdot\|$ is the Euclidean norm, $\bar{R}$ and $\widetilde{R}$ are the ground truth and estimated rotation matrices, respectively, and $\mathrm{Log}(\cdot)$ is the logarithm map in the Lie group $\mathrm{SO}(3)$.

6.7.2 LiDARTag System and Speed Analysis

The computation time and cluster analysis of each step of the pipeline is shown in Table 6.2. Indoors, we have fewer clusters because detected features are closer to each other resulting in many of them being clustered together. The computation time in an outdoor environment is faster than indoors because more clusters are rejected in the early stage due to the sparsity of the clusters, see Table 6.3. In both environments, the system achieves real-time performance (at least 100 Hz).

[5] The shortest path between two points on the $\mathrm{SO}(3)$ group.

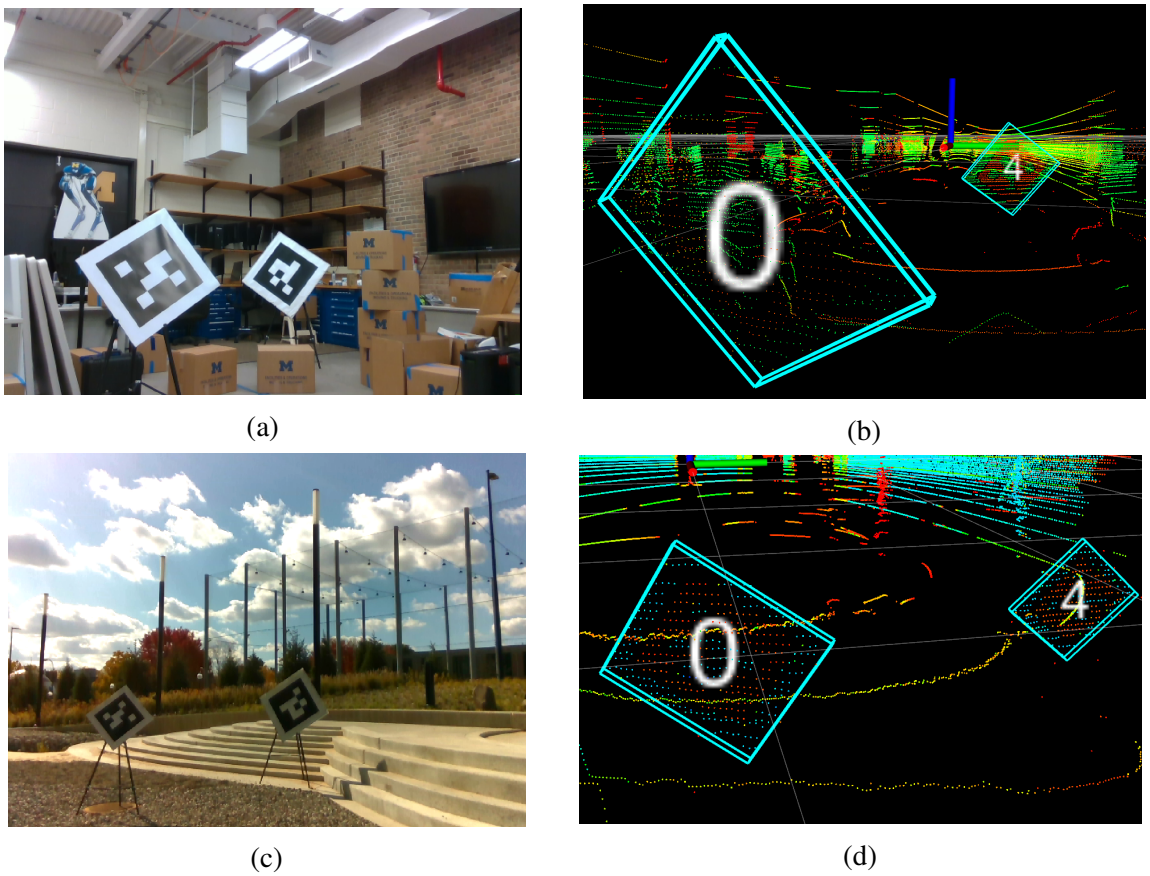
Table 6.3: This table takes into account all the data we collected and shows numbers of rejected clusters in each step in different scenes.

Outdoor		
No. Min. Return	No. Max. Return	No. Plane Fitting
247.72	3.41	14.71
No. Boundary Points	No. Pose Estimation	No. Decoding Failure
4.10	0.48	0.00
Indoor		
No. Min. Return	No. Max. Return	No. Plane Fitting
76.44	1.12	0.00
No. Boundary Points	No. Pose Estimation	No. Decoding Failure
8.14	1.80	1.16
Indoor Cartographer Dataset		
No. Min. Return	No. Max. Return	No. Plane Fitting
65.76	0	1.90
No. Boundary Points	No. Pose Estimation	No. Decoding Failure
0.35	0	0
Outdoor H3D Dataset		
No. Min. Return	No. Max. Return	No. Plane Fitting
713.35	8.72	44.41
No. Boundary Points	No. Pose Estimation	No. Decoding Failure
2.74	0.38	0

Table 6.4: The original double sum in (6.18) is too slow to achieve a real-time application. This table compares different methods to compute the double sum, in which the TBB stands for Threading Building Blocks library from Intel. Additionally, we also apply a k-d tree data structure to speed up the querying process; the k-d tree, however, does not produce fast enough results. The unit in the table is milliseconds.

Original Double Sum	Matrix Form	Vector From
144.18	67.11	8.51
TBB Original Form	TBB Vector Form	TBB k-d tree
35.68	2.40	5.73

The original double sum in (6.18) takes over 140 milliseconds for each decoding process. To speed up the process, the inner sum of the double sum is transformed to a matrix and then to a vector form. For more details, see our implementation [101]. These two modifications boost the speed to 8.5 ms. However, this is still not fast enough because for each remaining cluster, (6.18) needs to be computed with all the tags in the function dictionary. Threading Building Blocks library (TBB) [147] is therefore used to further speed up this process to 2.4 ms. All together, the whole process was sped up by a factor of 60, from 144 ms to 2.4 ms. Furthermore, we also investigated the speed performance of employing a k-d tree data structure [148]. A summary Table 6.4 is presented, showing that use of a k-d tree did not improve performance.



(a) (b)

(c) (d)

Figure 6.11: (a) and (c) image a 0.8 and a 0.6 meter tag placed in a cluttered indoor laboratory and a spacious outdoor environment. (b) and (d) show the algorithm successfully detects the two markers of different sizes indicated by cyan boxes.

6.7.3 False Positive Analysis

We have chosen some public datasets containing no ARTag features (i.e., no LiDARTags) so that there cannot be any false negatives and any detection of a LiDARTagin the datasets is consequently a false positive. To better verify the proposed LiDARTag algorithm, cluttered indoor scenes and crowded outdoor scenes are both necessary. The Google Cartographer indoor dataset [41] and Honda H3D outdoor dataset [42] were therefore used to validate the false positive rate of the proposed system. The Cartographer was collected with two Velodyne VLP-16 LiDARs in the Deutsches Museum. We took the longest three sequences consisting of more than 350 thousand LiDAR scans. Each scan contains about 30,000 points. We used the same algorithm with the same parameters and the full function dictionary that was used to detect the two tags in Fig. 6.11 (true positives were detected), and no false positives (i.e., no LiDARTags) were detected in the three sequences.

The Honda H3D dataset was collected by a 64-beam Velodyne LiDAR and consists of 160 crowded and highly interactive traffic scenes in the San Francisco Bay Area. We evaluated on all

Table 6.5: This table shows the numbers of false positive rejection of the proposed algorithm. We validated the rejection rate on the indoor Google Cartographer dataset and the outdoor Honda H3D datasets. The former has two VLP-16 Velodyne LiDAR and the latter has one 64-beam Velodyne LiDAR.

	Google Cartographer	Honda H3D
Scene	Indoor Museum	Crowed Driving Scenes
Duration	150 minutes	48 minutes
No. Scans	350 thousand	29 thousand
No. False Positives	0	0

sequences, resulting in 29 thousand LiDAR scans. Each scan consists of more than 130 thousand points. Zero targets were extracted by the detector. The results are shown in Table 6.5. Additionally, false positives removed by each step are provided in Table 6.3. Last but not the least, Fig. 6.11 shows that the detector is able to detect markers of different sizes both in a cluttered indoor scene and a spacious outdoor scene.

6.8 Conclusion and Future Work

We presented a novel and flexible fiducial marker system specifically for point clouds. The developed fiducial tag system runs in real-time (faster than 100 Hz) while it can handle a full scan of raw point cloud from the employed *32-Beam Velodyne ULTRA Puck LiDAR* (up to 120,000 points per scan). Each step of the proposed system was extensively analyzed and evaluated in both cluttered indoor as well as spacious outdoor environments. Furthermore, the system can be operated in a completely dark environment.

The LiDARTag pose estimation block deploys an L_1-inspired cost function. It achieved millimeter accuracy in translation and a few degrees of error in rotation compared to ground truth data collected by a motion capture system with 30 motion capture cameras. The sparse LiDAR returns on a LiDARTag are lifted to a continuous function in a reproducing kernel Hilbert space where the inner product is used to determine the marker's ID, and this method achieved 99.7$\%$ accuracy. The rejection of false positives was evaluated on the Google Cartographer indoor dataset and the Honda H3D outdoor dataset. No false positives were detected in over 379 thousand LiDAR scans.

The presented fiducial marker system can also be used with cameras and has been successfully used for LiDAR-camera calibration in [9] and [62]. Additionally, the system is able to detect various marker sizes, whereas camera-based fiducial markers support one marker size at a time. In the future, we shall use the developed LiDARTag within SLAM systems to provide robot state estimation and loop closures. Because of different inherent properties of LiDARs and cameras, it would also be interesting to fuse a camera-based tag system and the proposed LiDARTag system. Currently, the proposed algorithm assumes the point cloud has been motion compensated; how to adopt motion distortion into the algorithm is an interesting direction for future work. Furthermore,

if a dataset has been collected and labeled, a deep-learning architecture can replace the process of LiDARTag detection, thus offering another interesting area for future research.

CHAPTER 7

Optimal Shape Design for LiDAR Pose Estimation and Global Solver

7.1 Introduction

Targets have been widely employed as fiducial markers [30, 32, 33, 35–40] and for target-based sensor calibration [9–17, 19–21]. Fiducial markers (artificial landmarks or targets) help robots estimate their pose by estimating the target pose and are applied to simultaneous localization and mapping (SLAM) systems for robot state estimation and loop closures. Additionally, it can facilitate human-robot interaction, allowing humans to give commands to a robot by showing an appropriate marker. Extrinsic target-based calibration between sensors (cameras, Light Detection and Ranging (LiDAR) sensors, Inertial Measurement Unit (IMU), etc) is crucial for modern autonomous navigation [44]. Particularly in target-based LiDAR-camera calibration, one seeks to estimate a set of corresponding features of a target (e.g., edge lines, normal vectors, vertices, or plane equations) in the LiDAR's point cloud and the camera's image plane.

The targets applied in these critical tasks are typically symmetric (square, diamond, rectangle, or circle). A single symmetric target, such as a square [30, 39], leads to an ambiguous pose. This can be solved by adding an observable pattern to the target or by assuring that several asymmetrically-placed symmetric targets can be observed in a single scene. Furthermore, estimating the pose or features of a target of injudicious design suffers from the quantization uncertainty of a sensor, especially for LiDAR sensors. A high-end LiDAR, such as *32-Beam Velodyne ULTRA Puck LiDAR*, still has roughly six centimeters of quantization error at 10 meters, and 18 centimeters at 30 meters. The quantization uncertainty in the LiDAR point cloud leads to rotation errors greater than 15 degrees for targets farther away than 15 meters.

In this chapter, we propose the concept of optimizing target shape to ameliorate problems caused by quantization uncertainty and sparsity of the LiDAR image of a target. Specifically, we propose that a "good target shape" is one that possesses large gradients at edge points when the target undergoes rotations or translations. Moreover, we present a means that exploits target geometry to extract target vertices while estimating pose. The pose estimation problem is formulated so that an existing Semidefinite Programming (SDP) global solver can be modified to globally and efficiently

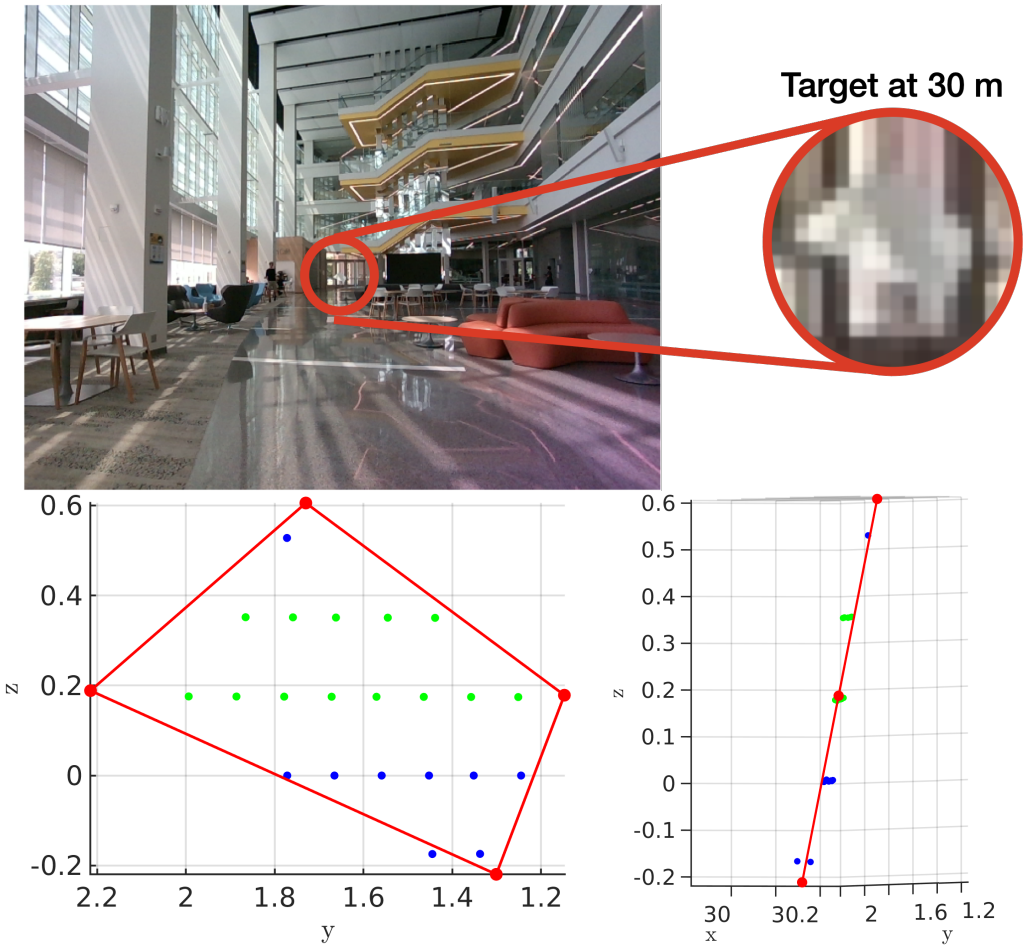


Figure 7.1: Illustration of the vertex and pose estimation using the proposed optimal target shape placed 30 meters away from a LiDAR in the atrium of the Ford Robotics Building at the University of Michigan. The red frame is the proposed optimal shape that induces large gradients at edge points under translation and rotation. The pose and vertices of the target are jointly estimated by a global solver that uses known target geometry. The bottom two figures show the front view and a side view of the fitting results, respectively. The blue and the green dots are the LiDAR returns on the target. Only the blue dots are used for pose estimation while to demonstrate robustness of the approach, the green dots are considered missing. If one were to apply RANSAC — a commonly used method — to regress the target boundaries and subsequently estimate the vertices by line intersections, the method would fail due to the sparsity of inliers.

compute the target's pose. Figure 7.1 shows the obtained pose estimation of a partially illuminated target placed 30 meters away and having only nine returns (blue dots) from a *32-Beam Velodyne ULTRA Puck LiDAR*, and three LiDAR rings on the target after the green dots are considered missing.

7.1.1 Contributions

In particular, this work presents the following contributions:

1. We propose the concept of target shape optimization for estimating pose and vertices from

LiDAR point clouds. Specifically, we design a target so that its edge points induced by LiDAR rings are "highly" sensitive to translation and rotation. This attenuates the effects of quantization uncertainty and sparsity of a target's LiDAR image. The resulting shape is asymmetric to remove pose ambiguity.

2. We present a means that uses target shape to jointly estimate target vertices and pose. Because the cost function of the proposed method can be formulated as an SDP, the target's pose and vertices can be globally estimated with an open-source solver [84].

3. We utilize an open-source LiDAR simulator [87] to provide ground truth of the poses and vertices. In the simulation, we validate that the optimal shape with the global solver achieves centimeter error in translation and a few degrees of error in rotation when the targets are at a distance of 30 meters and partially illuminated. In addition, we conduct experimental evaluations where the ground truth data are provided by a motion capture system, and achieve results similar to the simulation.

4. We open-source all the related software for this work, including the generation of the optimal shape, our means for pose estimation, and the simulated/experimental datasets; see [149, 150].

7.2 Related Work

To the best of our knowledge, there is no existing work on target shape design for LiDAR point clouds. The closet publication on target shape design is [151], which evaluated the relative range error of dense terrestrial laser scanners using a plate, a sphere, and a novel dual-sphere-plate target. We therefore review instead some techniques to improve the pose estimation of fiducial markers and to assist in extracting features of calibration targets.

7.2.1 Fiducial Markers

Fiducial markers for cameras were originally developed and used for augmented reality applications [32,33] and have been widely used for object detection and tracking, and pose estimation [106]. Due to their uniqueness and fast detection rate, they are also often used to improve Simultaneous Localization And Mapping (SLAM) systems [104]. CCTag [111] adopts a set of rings (circular target) to enhance pose estimation from blurry images. ChromaTag [35] proposes color gradients on a squared target to speed up the detection process and obtain more accurate pose estimation. More recently, LFTag [38] has taken advantage of topological markers, a kind of uncommon topological pattern, on a squared target to improve pose estimation at a longer distance. However, all the mentioned fiducial markers only work on cameras.

In our prior work on LiDAR [39], we proposed the first fiducial marker for LiDAR point clouds, which can be perceived by both LiDARs and cameras. We achieved millimeter error in translation and a few degrees in rotation. However, due to the quantization error of the LiDAR, the performance of the pose estimation (especially in-plane rotation) was noticeably degraded when the target was farther than 12 meters. Thus, this work proposes the concept of target shape design to specifically address the quantization uncertainty present in LiDAR returns and push the range of pose estimation to more than 30 meters. In passing, we note that symmetric targets, such as a square or hexagon, suffer from rotational ambiguity. Hence, our designed target will not be symmetric.

7.2.2 Target-Based LiDAR-Camera Calibration

LiDAR-camera calibration [9–17, 19, 20] requires feature correspondences from the image pixels and the LiDAR point cloud. However, the representations and inherent properties of camera images and LiDAR point clouds are distinct. An image (pixel arrays) is dense and very structured, with the pixels arranged in a uniform (planar) grid, and each image has a fixed number of data points. On the other hand, each scan of a LiDAR returns a 3D point cloud consisting of a sparse set of (x, y, z) coordinates with associated intensities. In particular, LiDAR returns are not uniformly distributed in angle or distance [40, III-A]. Target-based LiDAR-camera calibration utilizes targets to identify and estimate the corresponding features, such as vertices, 2D/3D edge lines, normal vectors, or the plane equations of the targets. References [9–11, 21] have noted that placing the targets so that the rings of the LiDAR ran parallel to its edges led to ambiguity in the vertical position due to the spacing of the rings and thus was detrimental to vertex or feature estimation. References [14, 19] utilize RANSAC [152] and plane fitting to remove the outliers of the LiDAR returns, while [11] proposes a "denoising process" for LiDAR returns around the target boundaries before applying RANSAC to extract features. When estimating the target vertices, the later references separate the edge points into groups and then apply the RANSAC algorithm. However, regressing the line equation of edge points will fail when there are not enough edge points or inliers, as shown in Fig. 7.1. Additionally, no target geometry information is used while estimating the features.

The remainder of this chapter is organized as follows. Section 7.3 formulates the design of an optimal target shape for LiDAR point clouds. The extraction of the target vertices while globally estimating the pose is discussed in Sec. 7.5. The simulation and experimental results are presented in Sec. 7.6 and Sec. 7.7. Finally, Sec. 7.8 concludes the chapter and provides suggestions for further work.

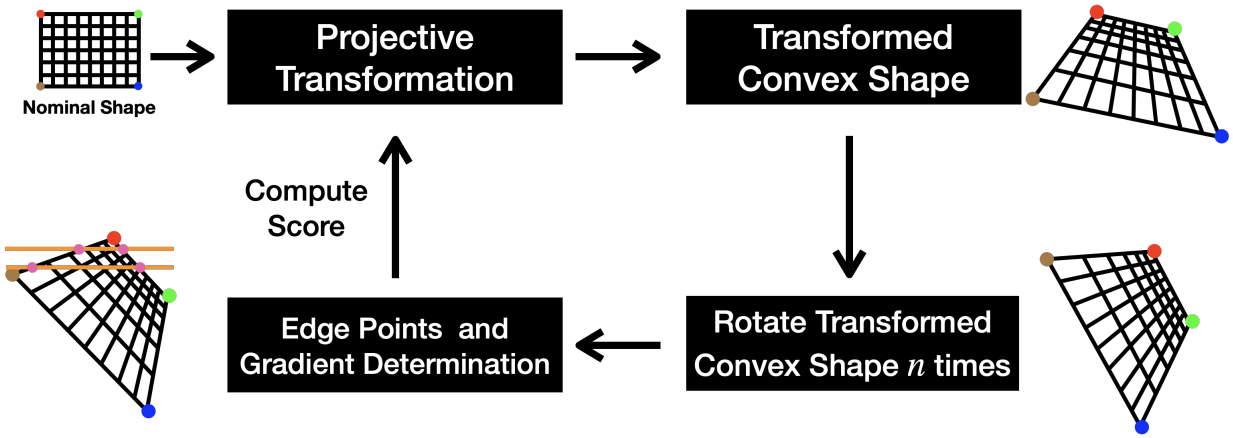


Figure 7.2: Optimization process for determining an optimal target shape. Projective transformations applied to a nominal quadrilateral generate candidate convex quadrilaterals (targets). Edge points are intersections of LiDAR rings with the target boundaries. The objective is to maximize the gradient of edge points under actions of $\mathrm{SE}(2)$ applied to the target. To enhance robustness, the gradients are computed for n-discrete rotations of the quadrilateral under partial illumination, and the score is the worst case.

7.3 Optimal Shape For Sparse LiDAR Point Clouds

In this section, we propose a mathematical formulation of target shape design. The main idea is for target translation and rotation to result in large gradients at edge points defined by the LiDAR returns. Fig. 7.2 summarizes a high-level optimization process.

7.3.1 Convex Shape Generation

We apply projective transformations on a nominal convex quadrilateral in 3D to generate planar candidate targets. We will see that applying a projective transformation rather than working with a collection of vertices makes it easier to ensure convexity of the target and to generate a cost function that is invariant under scaling and translations.

Let $\mathcal{V} := \{X_i | X_i := (x_i, y_i, 1) | x_i, y_i \in \mathbb{R}\}_{i=1}^4$ denote the 3D vertices X_i of a nominal convex quadrilateral, such as a square. Given $\mathbf{P}$, a projective transformation defined by a non-singular 3×3 matrix [69, p.33], let $\widetilde{X}_i$ denote the new vertices transformed by $\mathbf{P}$:

$$\widetilde{X}_i = \begin{bmatrix} x_i' \\ y_i' \\ \lambda_i' \end{bmatrix} = \mathbf{P} X_i = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & v \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}. \tag{7.1}$$

The resulting vertices $\widetilde{\mathcal{V}} := \{\widetilde{X}_i\}_{i=1}^4$ lie in the projective space $\mathbb{P}^2$ [69, p.26]. Let $\mathcal{V}'$ be the corresponding transformed vertices in the Cartesian space $(\mathbb{R}^2)$ [69, p.27] and $\Pi : \mathbb{P}^2 \to \mathbb{R}^2$ be the

mapping function

$$\Pi(\widetilde{\mathcal{V}}) := \Pi(\mathbf{P}(\mathcal{V}))) := \left\{ X_i' \middle| X_i' := \left(\frac{x_i'}{\lambda_i'}, \frac{y_i'}{\lambda_i'} \right) \right\}_{i=1}^{4}. \tag{7.2}$$

To summarize, given a nominal convex quadrilateral, $\mathcal{V}$, and a projective transformation, $\mathbf{P}$, we construct a new quadrilateral via

$$\mathcal{V} \longmapsto \mathbf{P}\mathcal{V} =: \widetilde{\mathcal{V}} \text{ and } \mathcal{V}' := \Pi(\widetilde{\mathcal{V}}). \tag{7.3}$$

Remark 24. *From here on, we will abuse notation and pass from Cartesian coordinates to homogeneous coordinates without noting the distinction.*

It is important to note that for any desired quadrilateral, there exists a projective transformation yielding $\mathcal{V}'$ from $\mathcal{V}$. Hence, our procedure for generating candidate targets is without loss of generality.

Theorem 2 ([153, p.274])**.** *There exists a unique projective transformation that maps any four points, no three of which are collinear, to any four points, no three of which are collinear.*

While Theorem 2 can be used to construct an arbitrary quadrilateral, convexity need not be conserved as shown in Fig. 7.3. We need the following condition to ensure that the resulting polygon is convex:

Theorem 3 ([154, p.39])**.** *Let* $\Omega : \mathbb{R}^3 \to \mathbb{R}^2$ *by* $\Omega(x, y, \lambda) = (x/\lambda, y/\lambda)$*. If* $\mathrm{dom}\Omega = \mathbb{R}^2 \times \mathbb{R}_{++}$*, where* $\mathbb{R}_{++} = \{x \in \mathbb{R} | x > 0\}$ *and the set* $C \subseteq \mathrm{dom}\Omega$ *is convex, then its image*

$$\Omega(C) = \{\Omega(x) | x \in C\} \tag{7.4}$$

is also convex.

For the domain of our projective transformation to be $\mathbb{R}^2 \times \mathbb{R}_{++}$, and hence for the candidate target to be automatically convex, the following linear inequality should be imposed in (7.1),

$$p_{31}x_i + p_{32}y_i + \upsilon > 0. \tag{7.5}$$

Because we consider two candidate targets to be equivalent if one can be obtained from the other by translation and scaling, we are led to decompose the projective transformation as follows

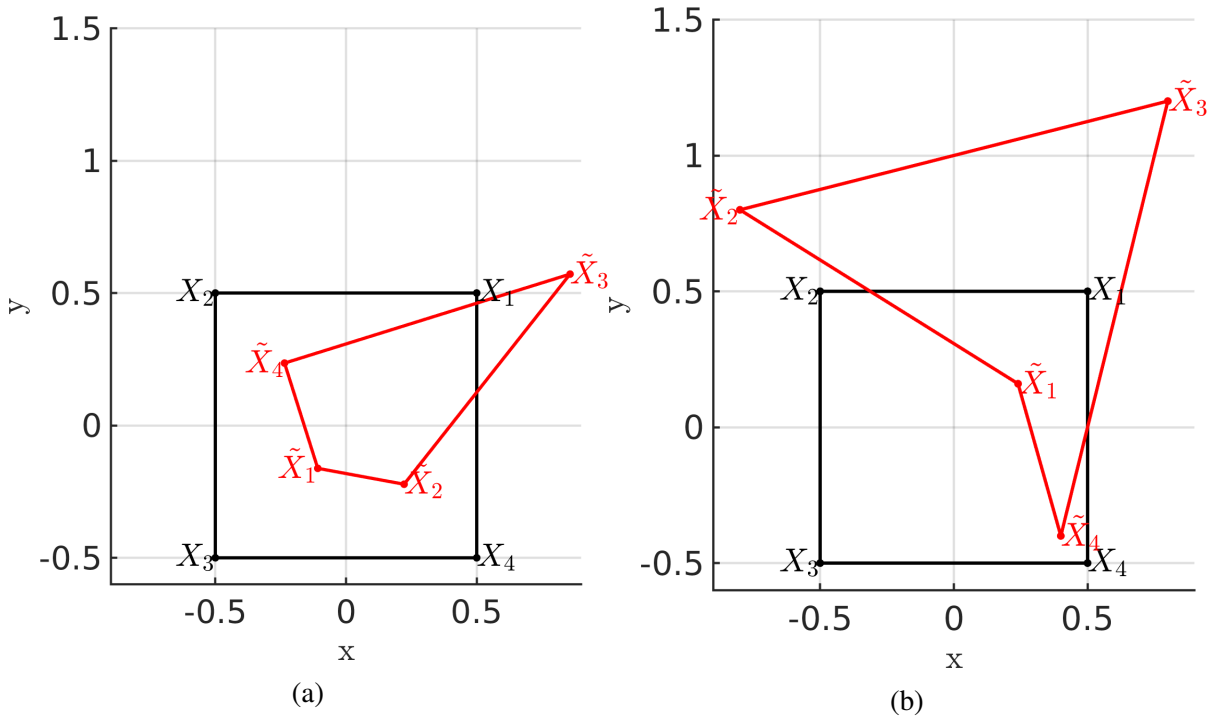


Figure 7.3: Equation (7.5) versus convexity. The red indicates the resulting shapes transformed by projective transformations from the same nominal shape (black). The left shows a case where (7.5) is satisfied and thus the transformed shape is convex; otherwise, it is non-convex, as shown on the right, where $p_{31}x_3 + p_{32}y_3 + v = -0.5$.

[155, 156],

$$\begin{aligned}\mathbf{P} &= \mathbf{H}_S\mathbf{H}_{SH}\mathbf{H}_{SC}\mathbf{H}_E \\ &= \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & \mathbf{1} \end{bmatrix} \begin{bmatrix} 1 & k & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \lambda & 0 & 0 \\ 0 & 1/\lambda & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{v}^\top & v \end{bmatrix},\end{aligned} \tag{7.6}$$

where $\mathbf{H}_S, \mathbf{H}_{SH}, \mathbf{H}_{SC}, \mathbf{H}_E$ are similarity, shearing, scaling and elation transformations, respectively; see [155] for more details. By setting $s = 1, \mathbf{t} = (0, 0)$ in (7.6), the DoF of the projective transformation drops from eight to five. Our family of candidate targets is now given by (7.2) with $\mathbf{P}$ satisfying (7.5) and (7.6).

In summary, we can describe candidate convex target shapes via projective transformations while reducing the number of degrees of freedom to five.

Remark 25. *The same design process could be run with an N-gon, for $N \geqslant 3$. If N is too large, the target will have at least one very short edge which will be impossible to discern in a point cloud. In between, it's a tradeoff between having adequate area to collect LiDAR returns, non-parallel edges to minimize pose ambiguity, and long enough edges to pick them out of a point cloud. We used a 4-gon as a reasonable starting point. Investigating N equals 3 and 5 would be interesting as well.*

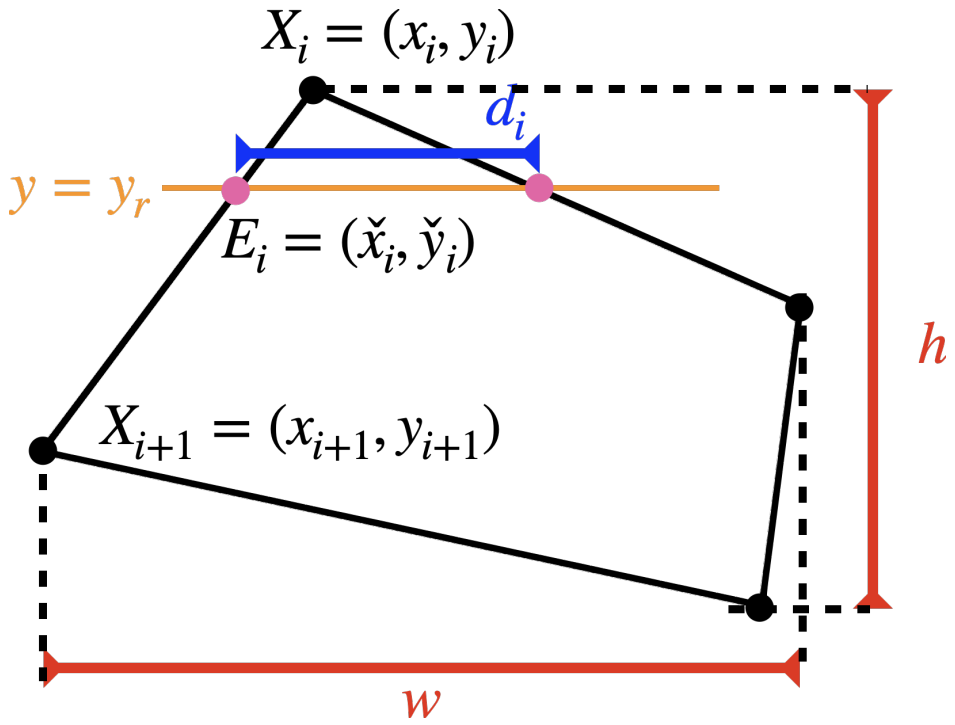


Figure 7.4: The pink dots are the edge points determined by a LiDAR ring (orange line) intersecting with the edge line connecting two vertices (X_i and X_{i+1}). The distance between the two edge points on the same LiDAR ring is d_i. The height and width of the shape is w and h, respectively.

7.3.2 Edge Points Determination

As mentioned in (7.2), $\mathcal{V}'$ are the 2D vertices of a candidate target. An edge point E_i is defined by the intersection point of a LiDAR ring and the line connecting two vertices of the polygon as shown in Fig. 7.4. If we let $\mathcal{S}$ be the boundary of the quadrilateral with vertices $\mathcal{V}'$, the collection of edge points detected by the LiDAR is the set $\mathcal{EP} := \{E_i\}_{i=1}^{M}$, where M is the number of edge points and is given by the intersection of $\mathcal{S}$ with the LiDAR rings $\mathcal{LR}$, i.e.

$$\mathcal{S} = \partial\text{conv}(\mathcal{V}'), \quad \{E_i\}_{i=1}^{M} = \mathcal{S} \cap \mathcal{LR}. \tag{7.7}$$

When the LiDAR rings are horizontal ($y = y_r$), an edge point $E_i = (\breve{x}_i, \breve{y}_i)$ can always be computed in closed form,

$$\breve{x}_i = x_i + \frac{x_{i+1} - x_i}{y_{i+1} - y_i}(y_r - y_i) \;\text{ and }\; \breve{y}_i = y_r. \tag{7.8}$$

7.3.3 Shape Sensitivity

From experience gained in LiDARTag [39], we observed that the pose estimation suffers the most from in-plane rotation. Therefore, we compute the shape sensitivity in $\mathrm{SE}(2)$. The sensitivity of a polygon is defined as the gradient of the edge points with respect to rigid-body transformations

of the polygon, with the LiDAR rings held constant, as shown in Fig. 7.5. Hence, the sensitivity captures the horizontal movement of an edge point after the shape is rotated and translated.

For a transformation in the Special Euclidean group $\mathbf{H} \in \mathrm{SE}(2)$, let E_i', denote the transformed edge point by

$$E_i' := \mathbf{H} \circ E_i = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \breve{x}_i \\ \breve{y}_i \\ 1 \end{bmatrix}, \tag{7.9}$$

where θ, t_x, t_y are the rotation angle, the translation on x-axis, and the translation on y-axis, respectively. Denote $\mathrm{Exp}(\kappa, \omega, u, v)$ as the exponential map that maps from the Lie algebra $\mathfrak{se}(2)$ to the continuous Lie group $\mathrm{SE}(2)$,

$$\begin{aligned} \mathrm{Exp}(\kappa, \omega, u, v) &= \mathrm{expm}\left(\kappa \begin{bmatrix} 0 & -\omega & u \\ \omega & 0 & v \\ 0 & 0 & 0 \end{bmatrix} \right) \\ &= \begin{bmatrix} \cos(\omega\kappa) & -\sin(\omega\kappa) & \frac{1}{\omega}(v\cos(\omega\kappa) + u\sin(\omega\kappa) - v) \\ \sin(\omega\kappa) & \cos(\omega\kappa) & \frac{1}{\omega}(v\sin(\omega\kappa) - u\cos(\omega\kappa) + u) \\ 0 & 0 & 1 \end{bmatrix}, \end{aligned} \tag{7.10}$$

where (ω, u, v) parameterize the unit sphere $S^2 \subset \mathbb{R}^3$, expm is the usual matrix exponential, and κ is a dummy variable for differentiation. Comparing $\mathbf{H}$ in (7.9) and (7.10) leads to

$$\begin{cases} \theta = \omega\kappa \\ t_x = \frac{1}{\omega}(v\cos(\omega\kappa) + u\sin(\omega\kappa) - v) \\ t_y = \frac{1}{\omega}(v\sin(\omega\kappa) - u\cos(\omega\kappa) + u). \end{cases} \tag{7.11}$$

For each triple of values (ω, u, v), the action of (7.10) on a candidate target quadrilateral results a path $p_i(\kappa) := p_i(\mathrm{Exp}(\kappa\omega, \kappa u, \kappa v))$ being traced out by the edge points along a LiDAR ring. Using (7.11) to differentiate the path p_i at the identity of $SE(2)$ produces an action of $\mathfrak{se}_2$,

$$v_i(\omega, u, v) := \left.\frac{d}{dt}\right|_{\kappa=0} p_i(\mathrm{Exp}(\kappa\omega, \kappa u, \kappa v)),\ (\omega, u, v) \in \mathfrak{se}_2. \tag{7.12}$$

From (7.8), (7.10), (7.11), and (7.12), the gradient of the edge point with respect to the LiDAR ring is

$$\begin{aligned} v_{i_x} =& \omega \left(\frac{(x_i - x_{i+1})(x_i y_{i+1} - y_i x_{i+1} + x_{i+1} y_r - x_i y_r)}{(y_i - y_{i+1})^2} - y_r \right) \\ &+ u + \left(\frac{x_{i+1} - x_i}{y_{i+1} - y_i} \right) v. \end{aligned} \tag{7.13}$$

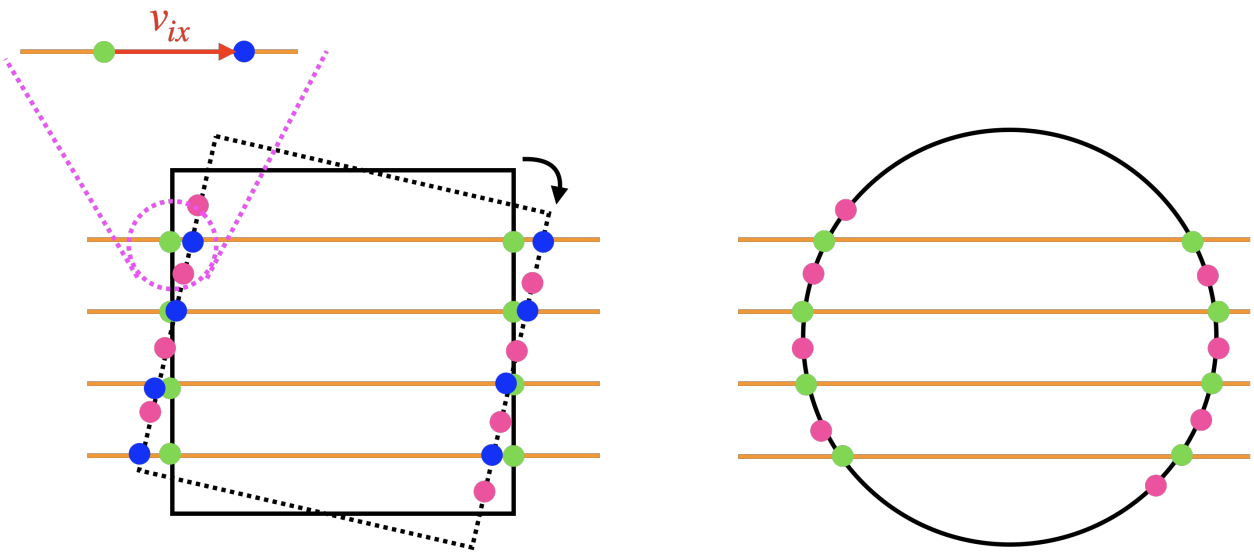


Figure 7.5: Shape sensitivity under rotation. The sensitivity of a shape is defined by the gradient (v_{ix}) of each edge point as it moves along the ring lines (orange) under rotations and translations of the shape. The green dots are the original edge points and the pink dots are the edge points on the rotated shape. The same process can be done for translation. The left shows a gradient v_{ix} of a square where the dotted square has been rotated from the nominal positioned square. The right shows that the gradient of an edge point on a circle is zero under rotation about its center. The gradient would be non-zero for translations.

Notice that $v_{i_y} = 0$ because the y-coordinates of $\mathbf{H}(\mathcal{V}') \cap \mathcal{LR}$ remain fixed.

Finally, we define the sensitivity $\mathcal{M}$ of the polygon

$$\mathcal{M}(\mathcal{V}, \mathcal{LR}, \omega, u, v) := \frac{1}{h} \sum_{i=1}^{M} v_{i_x}^2, \tag{7.14}$$

where M is the number of edge points, and h, defined in Fig. 7.4 is included because gradients in (7.13) scale with vertical height.

7.3.4 Initial Cost Function

The candidate target's sensitivity defined in (7.14) does not take into account the discrete nature of the LiDAR returns on a given ring. Let d_i denote the distance between two edge points on the i-th LiDAR ring. We want to encourage targets that have d_i larger than the spatial quantization of the LiDAR returns. We can do this in two ways, by scaling (7.14) by w and by including a term in the cost of the form $\sum_{i=1}^{K} d_i$ (d_i is larger for wider targets). The resulting score of shape (Ψ) becomes

$$\Psi = w\mathcal{M} + \mu \sum_{i=1}^{K} d_i, \tag{7.15}$$

where w is the width of the polygon, K is the number of rings illuminating the polygon, and μ is a weight trading off the two terms in the cost function.

7.4 Robust Shape for Real LiDAR Sensors

In previous section (Sec. 7.3.4), we added an extra term to (7.15) to account for the discrete measurements in azimuth direction of the LiDAR. Additionally, LiDARs have different ring densities at different elevation angles. For example, *32-Beam Velodyne ULTRA Puck LiDAR* has dense ring density between -5° and 3°, and has sparse ring density from -25° to -5° and from 3° to 15° [127]. A target could be partially illuminated in the sparse region, as shown in Fig. 7.6(a). Therefore, assuming edge points are uniformly distributed is not practical and using a distribution of edge points that is similar to reality is critical while maximizing (7.15). Additionally, LiDAR rings from mobile robots are not always parallel to the ground plane. We account for non-horizontal LiDAR rings by rotating the candidate target.

7.4.1 Partial Illumination of Target

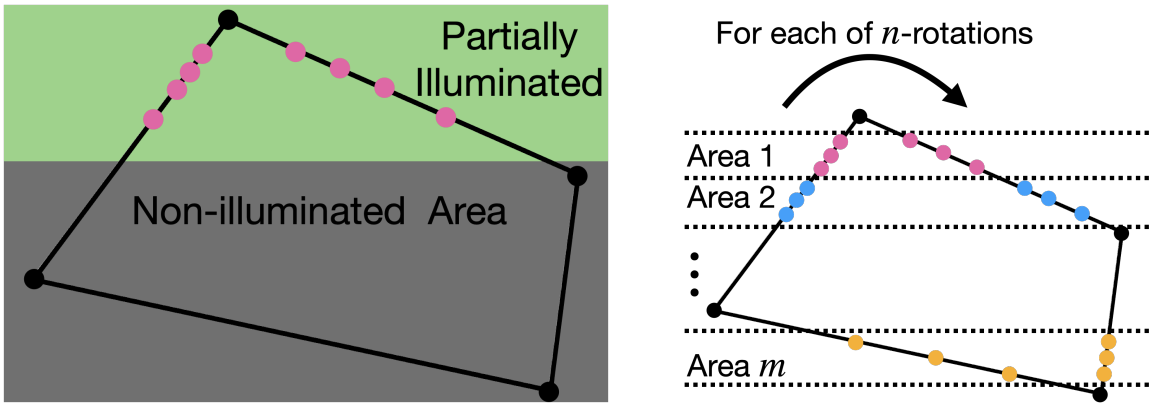
To have the shape being robust to illuminated area and the angle of the rings with respect to the target, we first rotate the generated polygon n times, and then divide the rotated polygon into m areas. Only one area is illuminated by LiDAR rings at a time to determine edge points and to compute the score (7.15), Ψ_{ij}, for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. Figure 7.6(b) shows the edge points being determined for a partially illuminated target. Equation (7.15) is consequently evaluated $n \times m$ times for illumination of the target and the lowest among the $n \times m$ scores is the final score of the candidate target shape.

7.4.2 Optimization for the Optimal Shape

To summarize, the resulting optimization problem depends on the projective transformation parameters that are used to generate a convex polygon, edge points illuminated by horizontal LiDAR rings lied on the rotated quadrilateral, the transformation of the edge points in $\mathfrak{se}_2$, and distances between two edge points on the corresponding LiDAR rings. Thus, the optimization problem is defined as:

$$\mathbf{P}^* = \arg\min_{\mathbf{P}} \min_{\omega,u,v} \max_{i,j} \{-\Psi_{ij}\}. \tag{7.16}$$

The optimization problem (7.16) was (locally) solved by `fmincon` in MATLAB, after the optimization parameters were randomly initialized. We rotated the generated polygon six times. Each rotated polygon was divided into five areas, and four LiDAR rings were used to illuminate one area at a time. The unit sphere of unit vectors in $\mathfrak{se}(2)$, mentioned in Sec. 7.3.3, was discretized



(a) (b)

Figure 7.6: The left shows a partially illuminated candidate shape. Because we also rotate the target when computing a score, we can without loss of generality use horizontal strips to partially occlude the target. The right shows that the target is divided into m-areas of partial illumination, and that for each of n-rotations of the candidate target, a score is assigned to each subarea based on (7.15). The final score of the shape is the lowest among the $n \times m$ scores.

into 25×25 faces (by normalizing the vectors to have unit length, we can reduce the dimension from three to two). Once the generated polygon was rotated and illuminated, the sensitivity of the resulting edges points were evaluated at each face on the unit sphere. The resulting optimal shape is shown in Fig. 7.7. One can observe that the resulting shape satisfies: **1)** it has sufficient area so as to collect LiDAR returns; **2)** the length of the shortest side is still long enough to be identified through edge points; **3)** its asymmetric shape avoids the issue of pose ambiguity.

Remark 26. *The main point of this chapter is that target shape can be used to enhance the estimation of target vertices and relative pose between a target and a LiDAR. We have proposed one algorithmic means to produce an "optimal target shape". Different notions of cost will result in different shapes.*

7.5 Global Pose and Feature Estimation

In this section, we propose a means to use known target geometry to extract target vertices while globally estimating relative pose between target and LiDAR. For a collection of LiDAR returns $\mathcal{TP} := \{\mathcal{X}_i\}_{i=i}^{N}$, let $\mathcal{EP} := \{E_i\}_{i=1}^{M} \in \mathcal{TP}$ be the M target edge points. Given the target geometry, we define a template with vertices $\{\bar{X}_i\}_{i=1}^{4}$ located at the origin of the LiDAR and aligned with the y-z plane as defined in Fig. 7.8. We also denote $\bar{\mathcal{L}} := \{\bar{\ell}_i\}_{i=1}^{4}$ as the line equations of the adjacent vertices of the template. We seek a rigid-body transformation from the template to the target, $\mathbf{H}_L^T \in \mathrm{SE}(3)$, that "best fits" the template onto the edge points. In practice, it is actually easier to project the edge points $\mathcal{EP}$ back to the origin of the LiDAR through the inverse of the current estimate of transformation $\mathbf{H}_T^L := \left(\mathbf{H}_L^T\right)^{-1}$ and measure the error there. The action of $\mathbf{H} \in \mathrm{SE}(3)$ on $\mathbb{R}^3$ is $\mathbf{H} \cdot \mathcal{X}_i = \mathbf{R}\mathcal{X}_i + \mathbf{t}$, where $\mathbf{R} \in \mathrm{SO}(3)$ and $\mathbf{t} \in \mathbb{R}^3$.

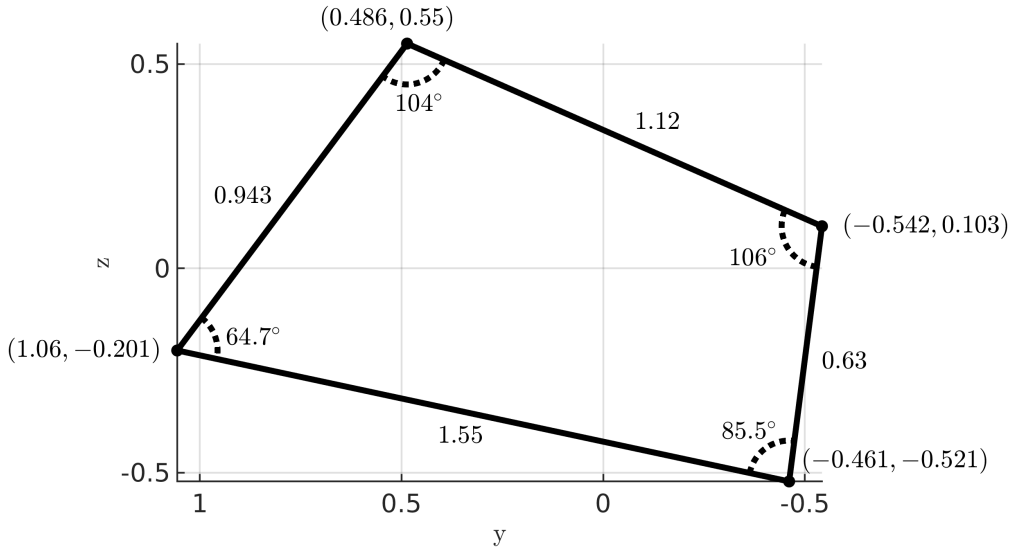


Figure 7.7: The resulting optimal shape from (7.16) in arbitrary units.

The cost j_i of edge point $E_i \in \mathcal{EP}$ is defined as the point-to-line distance,

$$j_i(E_i; \overline{\mathcal{L}}) = \min_{\bar{\ell}_i \in \overline{\mathcal{L}}} \|E_i - \bar{\ell}_i\|_2^2 \tag{7.17}$$

where $\overline{\mathcal{L}}$ is the set of line equations for the target. Let $\{\bar{E}_i\}_{i=1}^M := H_T^L(\mathcal{EP}) = \{H_T^L \cdot E_i\}_{i=1}^M$ denote the projected points by H_T^L. The total fitting error is defined as

$$J(H_T^L(\mathcal{EP})) := \sum_{i=1}^{M} j_i(\bar{E}_i) \tag{7.18}$$

To minimize (7.18), we adopt techniques that were used to globally solve 3D registration or 3D SLAM [1, 82–84] where the problem is formulated as a QCQP, and the Lagrangian dual relaxation is used. The relaxed problem becomes a SDP and convex. The problem can thus be solved globally and efficiently by off-the-shelf specialized solvers [85]. As shown in [84], the dual relaxation is empirically always tight (the duality gap is zero).

Once we (globally) obtain $\mathbf{H}_T^L$, the pose of the target is $\mathbf{H}_L^T = (\mathbf{H}_T^L)^{-1}$, and the estimated vertices are $\{\widetilde{X}_i\}_{i=i}^4 := \{\mathbf{H}_L^T \cdot \bar{X}_i\}_{i=1}^4$. The edge-line equations, the normal vector, and the plane equation of the target can be readily obtained from the vertices.

7.6 Simulation Results

Before carrying out experiments with the new target shape, we used a MATLAB-based LiDAR simulator introduced in [21] to extensively evaluate the pose and vertex estimation of the optimal shape. Both quantitative and qualitative results are provided. We do not compare against

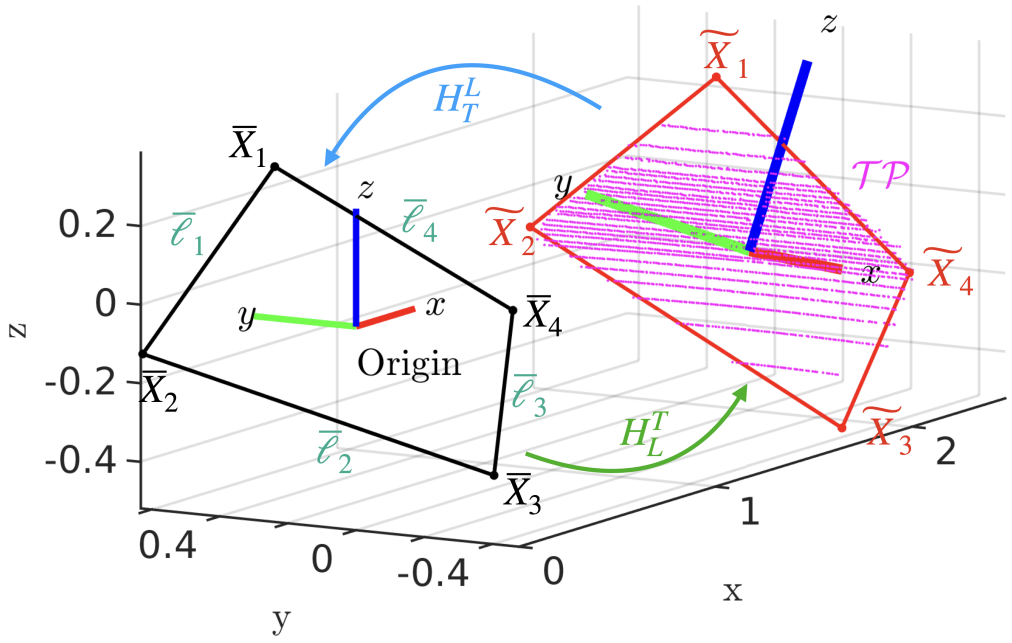


Figure 7.8: Pose definition and illustration of template fitting. A coordinate frame for the template (target shown in black) is defined by aligning the plane of the template with the y-z plane of the LiDAR frame and also aligning the mean of its vertices with the origin of the LiDAR. Let H_T^L (blue arrow) be an estimate of the rigid-body transformation from target to LiDAR, projecting the edge points of the target back to the template. The estimated pose of the target is then given by the inverse transformation, H_L^T (green arrow). The optimal H_T^L is obtained by minimizing (7.18) (based on point-to-line distance). This figure also shows a fitting result of a target at 2 meters in the Ford Robotics Building. The red frame is the template re-projected onto the LiDAR point cloud by H_L^T.

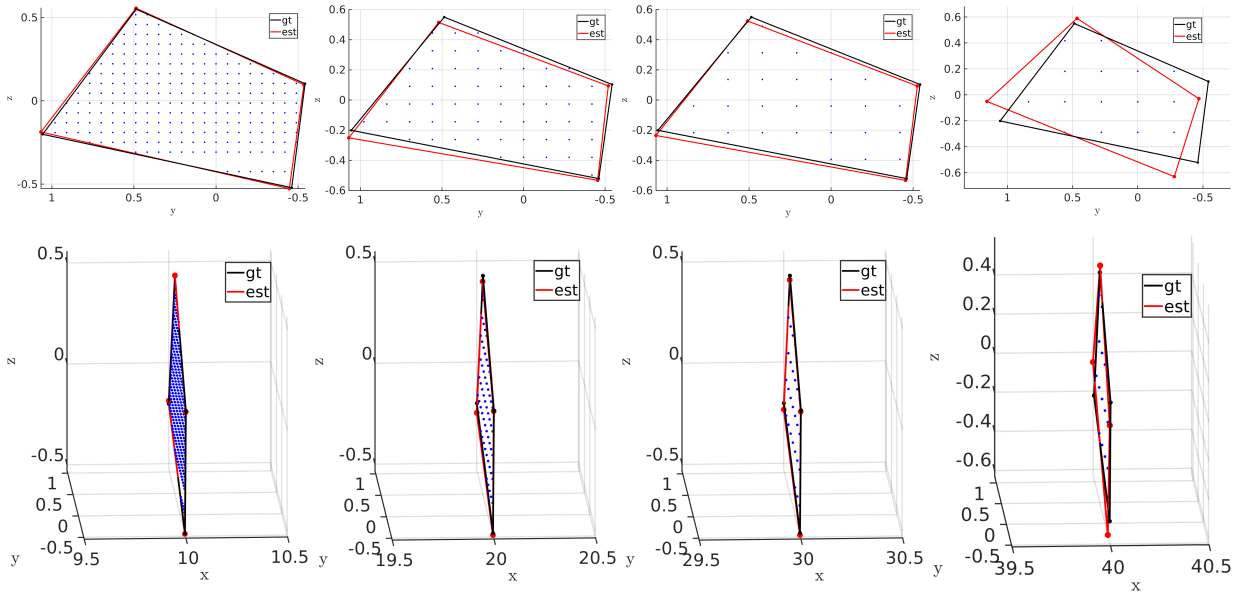


Figure 7.9: Simulation results of the noise-free dataset of the pose estimation at various distances (10, 20, 30, 40 m). LiDAR returns (blue dots) on the target are provided by the LiDAR simulator. Black indicates the ground truth pose from the simulator, and red is the estimated pose and vertices. The top and bottom show the front view and a side view of the fitting results, respectively.

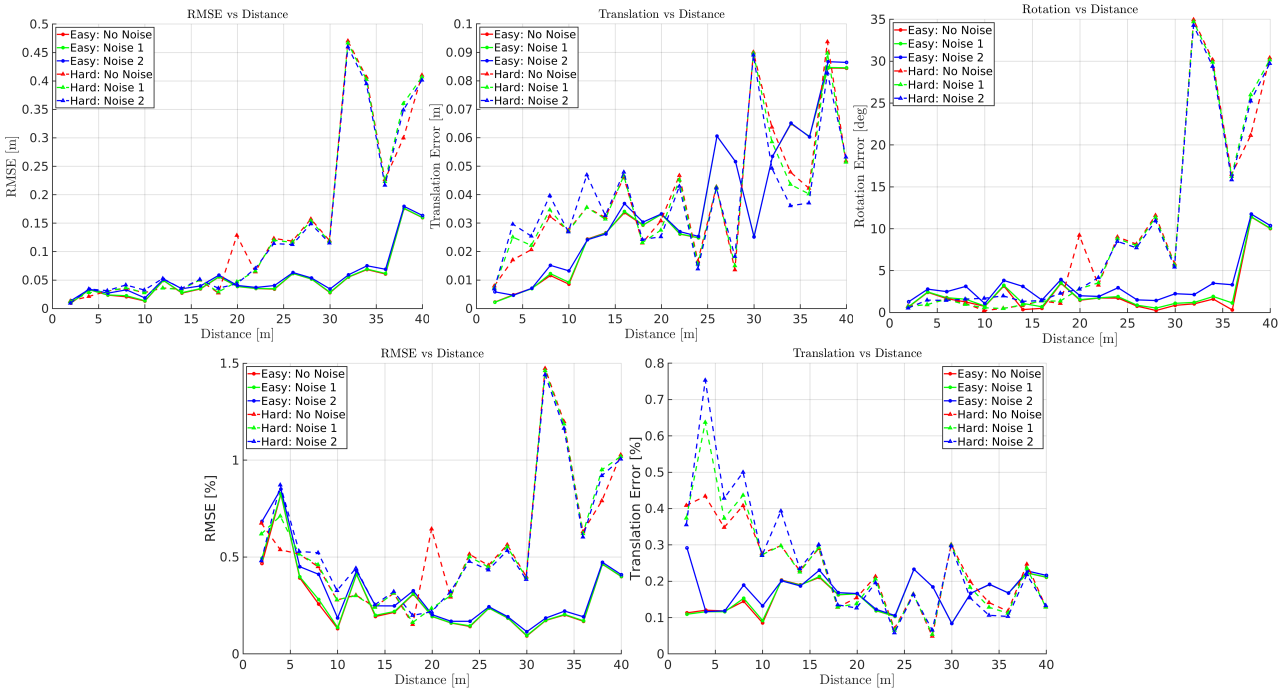


Figure 7.10: Simulation results with a target placed at distances from 2 to 40 meters in 2 m increments in the LiDAR simulator. At each distance, the simulation data are collected with the target face-on to the LiDAR as an easy case (solid line), and for the other, the target is rotated by the Euler angle (roll = 20°, pitch = 30°, yaw = 30°) as a challenging case (dashed line). In addition, we induce two different levels of noises to each dataset, as indicated by the different colors.

standard targets, such as unpatterned rectangles, diamonds, or circles, because their symmetry properties result in pose ambiguity. At large distances, a pattern would not be discernible.

We simulate a *32-Beam Velodyne ULTRA Puck LiDAR*, whose data sheet can be found at [127]. A target is placed at distances from 2 to 40 meters in 2 m increments. At each distance, simulation data is collected with a target face-on to the LiDAR as an easy case, and another dataset with the target rotated by the Euler angles (roll = 20°, pitch = 30°, yaw = 30°) under the XYZ convention as a challenging case. In addition, we induce two different levels of noise to each dataset to examine the robustness of the algorithm.

The results of vertex estimation are reported as the Root-Mean-Square-Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{4}\sum_{i=1}^{4} \|\widetilde{X}_i - X_i\|_2^2}, \tag{7.19}$$

where $\widetilde{X}_i$ is the estimated vertex and X_i is the ground truth vertex from the simulator. The pose on $\mathrm{SE}(3)$ is evaluated on translation e_t in $\mathbb{R}^3$ and rotation e_r on $\mathrm{SO}(3)$, separately. In particular, e_t and e_r are computed by

$$e_t := \|\mathbf{t} - \widetilde{\mathbf{t}}\| \;\text{ and }\; e_r := \|\mathrm{Log}(\mathbf{R}\widetilde{\mathbf{R}}^{\mathsf{T}})\|, \tag{7.20}$$

where $\|\cdot\|$ is the Euclidean norm, $\tilde{\cdot}$ is the estimated quantity, $\mathbf{R}$ and $\mathbf{t}$ are the ground truth rotation and translation, respectively, and $\mathrm{Log}(\cdot)$ is the logarithm map in the Lie group $\mathrm{SO}(3)$. Additionally, we also report the percentage of the RMSE and translation error, which are computed by each quantity divided by the centroid of the target. The quantization error e_q is the distance between two adjacent points on the same ring and can be approximated by the azimuth resolution (0.4°) of the LiDAR times the target distance.

Qualitative results are shown in Figure 7.9. The complete quantitative results of the distances and noise levels are shown as line charts in Fig. 7.10. Table 7.1 shows a subset of quantitative results of the pose and vertex estimation using the noise-free dataset. For vertex estimation, we achieve less than 1% error in most cases. The translation errors are less than the quantization error e_q. We also achieve a few degrees of rotation errors. It can be seen that the estimation limit of the optimal target of width 0.96 meter with our *32-Beam Velodyne ULTRA Puck LiDAR* optimal is 30 meters. However, for a LiDAR with a different number of beams or points, the estimation limit may be different. Based on these results, we were motivated to build the target and run physical experiments.

7.7 Experimental Results

We now present experimental evaluations of the pose and vertex estimation of the optimal shape. All the experiments are conducted with a *32-Beam Velodyne ULTRA Puck LiDAR* and an Intel RealSense camera rigidly attached to the torso of a Cassie-series bipedal robot, as shown in Fig. 7.11. We use the ROS [81] to communicate and synchronize between the sensors. Datasets are collected in the atrium of the Ford Robotics Building at the University of Michigan, and a spacious outdoor facility, M-Air [143], equipped with a motion capture system.

7.7.1 Quantitative Experimental Results in M-Air

The Qualisys motion capture system in M-Air is used as a proxy for ground truth poses and vertices. The setup consists of 33 motion capture cameras with passive markers attached to the target, the LiDAR and the camera, as shown in Fig. 7.11 and Fig. 7.12. Datasets are collected at various distances and angles. Each of the datasets contains images (20 Hz) and scans of point clouds (10 Hz). Similar to the simulation environment, the optimal-shape target is placed at distances from 2 to 16 meters (maximum possible in M-Air) in 2 meter increments. At each distance, data is collected with a target face-on to the LiDAR and another dataset with the target roughly rotated by the Euler angles (roll = 20°, pitch = 30°, yaw = 30°) as a challenging case. The results are shown in Table 7.2. As expected, the results are slightly worse (approximately one degree) than the simulator's due to the white noise of the LiDAR and many missing returns on the targets, as shown

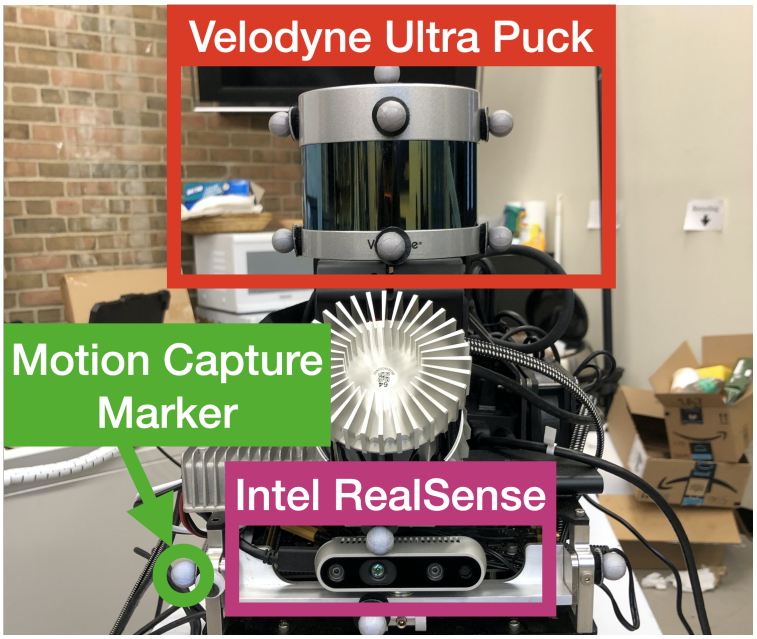


Figure 7.11: The sensor suite consists of a LiDAR, a camera and several motion capture markers.

in Fig 7.13.

Remark 27. *A Velodyne LiDAR return consists of the point's Cartesian coordinates, intensity, and ring number. For each ring, the first and the last point are the edge points of the ring. Since we define a template at the LiDAR origin, we first center the target points by subtracting its centroid. Each centered edge point is associated with the closest edge. Given the current association, we estimate the pose and then redo edge-point-to-edge-line association. Therefore, the optimization process is an alternating process. The optimization is terminated if* $\|\mathrm{Log}(H_{k-1}H_k)\|$ *is smaller than* $1e^{-5}$*, where* $\mathrm{Log}(\cdot)$ *is the logarithm map.*

7.7.2 Qualitative Experimental Results and Target Partial Illumination

For distances beyond 16 meters (the distance limit in M-Air), we present qualitative results from the atrium in Fig. 7.13 to support the simulation-based analysis. The blue dots are LiDAR measurements, and the red frame is the fitting result. Figure 7.14 illustrates a partially illuminated target (the green dots are assumed missing and only blue dots are used for pose estimation); the resulting fitting results are the red frame.

7.8 Conclusion

We presented the concept of optimizing target shape to enhance pose estimation for LiDAR point clouds. We formulated the problem in terms of choosing a target shape that induces large gradients

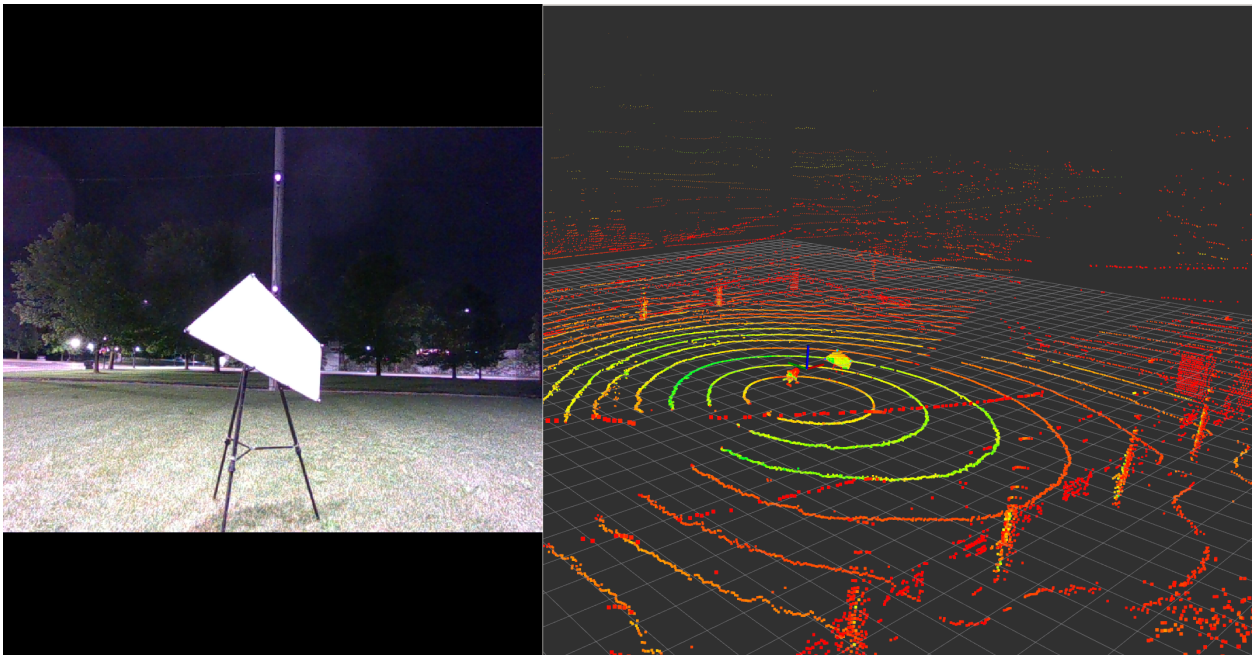


Figure 7.12: The Experimental setup. The left shows passive markers are attached to the four corners of the optimized target shape and the right shows a LiDAR scan in M-Air.

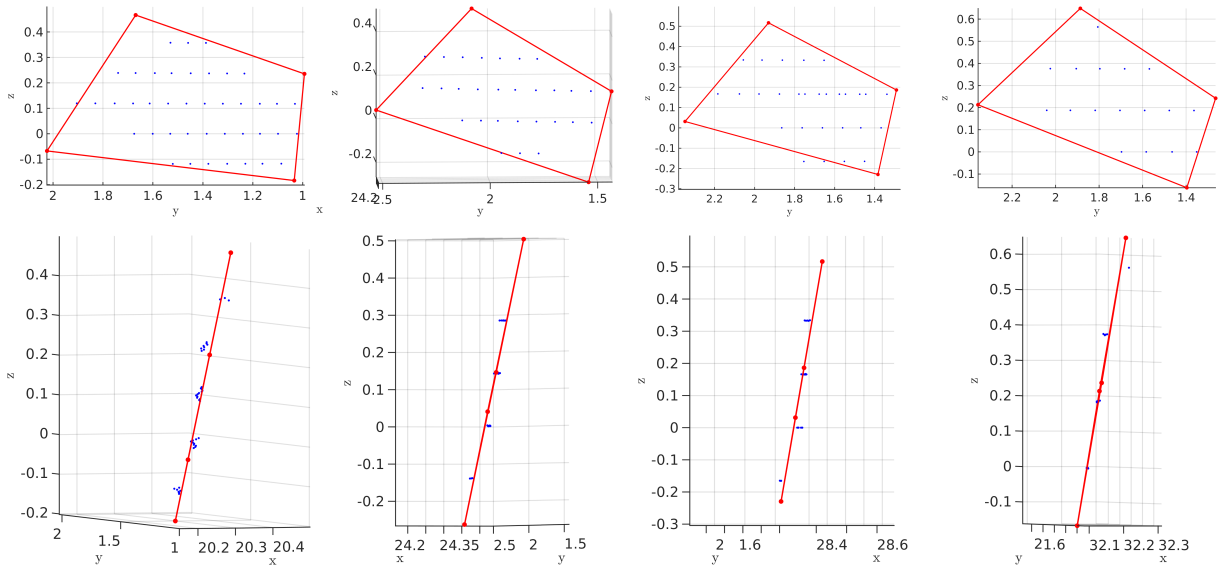


Figure 7.13: Fitting results of the optimal shape at various distances (20, 24, 28, 32 meters) in the atrium of the Ford Robotics Building at the University of Michigan. The blue dots are the LiDAR returns on the target and the red frame are the fitting results. The top and bottom show the front view and a side view of the results, respectively.

at edge points under translation and rotation so as to mitigate the effects of quantization uncertainty associated with point cloud sparseness. For additional robustness, the cost function or score for a candidate target was defined to be the minimum score under a set of relative rotations of the edge points; this had the effect of breaking symmetry in the candidate target, which also removes pose ambiguity.

Table 7.1: Pose and vertex accuracy of the simulation results at various distances. The vertex estimation is computed by (7.19). The quantization error e_q is the distance between two adjacent points on the same ring. The translation error e_t in meters and rotation error e_r in degrees are computed by (7.20).

Face-on LiDAR							
Distance [m]	No. Points	e_q [m]	RMSE [m]	RMSE [%]	e_t [m]	e_t [%]	e_r [deg]
2.00	2530	0.01	0.01	0.49	0.002	0.11	0.71
4.00	1015	0.03	0.03	0.83	0.005	0.12	2.45
6.00	548	0.04	0.02	0.40	0.01	0.12	1.77
8.00	338	0.06	0.02	0.28	0.01	0.15	1.58
16.00	96	0.11	0.04	0.22	0.03	0.21	0.68
30.00	28	0.21	0.03	0.10	0.03	0.08	1.09
32.00	26	0.22	0.06	0.17	0.05	0.17	1.20
Extreme Angle (20, 30, 30)							
Distance [m]	No. Points	e_q [m]	RMSE [m]	RMSE [%]	e_t [m]	e_t [%]	e_r [deg]
1.94	1734	0.01	0.01	0.62	0.01	0.37	0.71
3.93	716	0.03	0.03	0.71	0.03	0.64	0.95
7.93	258	0.06	0.04	0.46	0.03	0.44	0.96
15.93	70	0.11	0.05	0.31	0.05	0.29	1.37
29.93	21	0.21	0.12	0.39	0.09	0.30	5.65
31.93	17	0.22	0.47	1.46	0.06	0.18	34.72

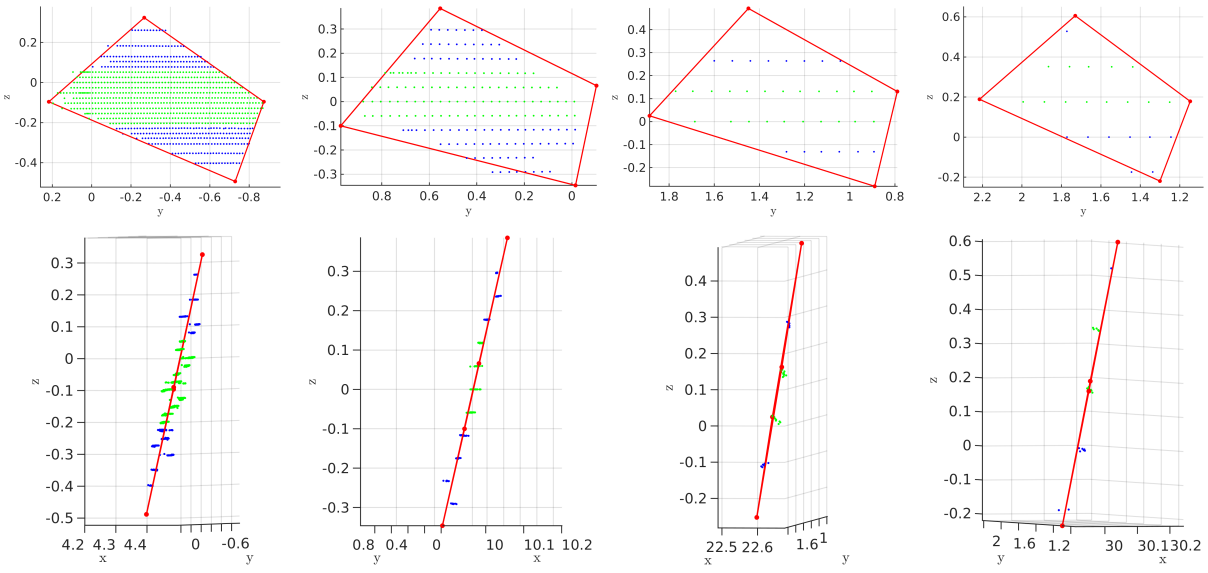


Figure 7.14: Fitting results of the partially illuminated target at various distances (4, 10, 22, 30 meters) in the atrium of the Ford Robotics Building at the University of Michigan. The selected distances are different from Fig. 7.13 to show more results. The red frames are the fitting results. The blue dots are the LiDAR returns on the targets while the green dots are considered missing. The top and bottom show the front view and a side view of the fitting results, respectively.

For a given target, we used the target's geometry to jointly estimate the target's pose and its vertices. The estimation problem was formulated so that an existing semi-definite programming global solver could be modified to globally and efficiently compute the pose and vertices of the

Table 7.2: Pose and vertex accuracy of the experimental results. The ground truth is provided by a motion capture system with 33 cameras. The quantization error e_q is the distance between two adjacent points on the same ring. The translation error e_t in meters and rotation error e_r in degrees are computed by (7.20).

Face-on LiDAR							
Distance [m]	No. Points	e_q [m]	RMSE [m]	RMSE [%]	e_t [m]	e_t [%]	e_r [deg]
2.1487	3009	0.015	0.026	1.193	0.024	1.133	0.619
4.0363	1120	0.028	0.029	0.718	0.024	0.603	1.906
6.0877	527	0.042	0.033	0.544	0.03	0.5	1.426
7.9145	589	0.055	0.033	0.413	0.03	0.384	1.765
10.093	197	0.07	0.034	0.334	0.03	0.294	1.846
12.016	136	0.084	0.035	0.288	0.03	0.247	2.081
13.987	192	0.098	0.03	0.216	0.028	0.201	1.737
15.981	155	0.112	0.03	0.187	0.028	0.176	1.237
17.971	119	0.125	0.031	0.173	0.028	0.156	1.387
Extreme Angle (20, 30, 30)							
Distance [m]	No. Points	e_q [m]	RMSE [m]	RMSE [%]	e_t [m]	e_t [%]	e_r [deg]
2.0448	2708	0.014	0.048	2.364	0.027	1.304	4.577
4.1225	898	0.029	0.04	0.971	0.034	0.835	2.641
5.966	504	0.042	0.033	0.549	0.029	0.486	1.826
7.9452	276	0.055	0.052	0.656	0.031	0.395	4.633
9.9484	155	0.069	0.041	0.412	0.034	0.345	3.52
12.006	105	0.084	0.06	0.498	0.036	0.299	6.064
14.106	75	0.098	0.051	0.364	0.045	0.318	2.714
16.05	54	0.112	0.057	0.355	0.041	0.253	4.634
18.087	48	0.126	0.055	0.305	0.044	0.241	3.593

target. A LiDAR simulator generated synthetic ground truth of the target pose and vertices. We validated that the combination of the optimal shape with the global solver achieved centimeter error in vertex estimation, centimeter error in translation, and a few degrees off in rotation in pose estimation when a partially illuminated target was placed 30 meters from the LiDAR . In experiments, when compared to ground truth data collected by a motion capture system with 33 cameras, we achieved results similar to those of the simulations.

In the future, we shall establish a system to automatically detect the shape in both images and LiDAR point clouds. If a dataset has been collected and labeled, automatic detection using deep-learning architectures is also an exciting future direction. Currently, the proposed algorithm assumes the point cloud has been motion compensated; how to adopt motion distortion into the algorithm is another direction for future work. Applying it as a fiducial marker system or as an automatic calibration system also offers another interesting area for further research. Furthermore, applying the proposed algorithm to 3D target shape fitting and generating shapes with more sides provide interesting research directions.

Part III

Motion Planning System for Legged Robots

CHAPTER 8

Informable Multi-Objective and Multi-Directional RRT* System for Robot Path Planning

8.1 Introduction and Contributions

Multi-objective or multi-destination path planning is a key enabler of applications such as data collection [157–161], traditional TSP [162, 163], and electric vehicle charging for long trips. More recently, autonomous "Mobility as a Service" (e.g., autonomous shuttles or other local transport between user-selected points) has become another important application of multi-objective planning such as car-pooling [164–172] Therefore, being able to efficiently find paths connecting multiple destinations and to determine the visiting order of the destinations (essentially, a relaxed TSP) is critical for modern navigation systems deployed by autonomous vehicles and robots. This chapter seeks to solve these two problems by developing a system composed of a sampling-based anytime path planning algorithm and a relaxed-TSP solver.

Another application of multi-objective path planning is robotics inspection, where a list of inspection waypoints is often provided so that the robot can preferentially examine certain equipment or areas of interest or avoid certain areas in a factory, for example. The pre-defined waypoints can be considered as prior knowledge for the overall inspection path the robot needs to construct for task completion. We show that the proposed system can inherently incorporate such knowledge.

To represent multiple destinations, graphs composed of nodes and edges stand out for their sparse representations. In particular, graphs are a popular representation of topological landscape features, such as terrain contour, lane markers, or intersections [173]. Topological features do not change often; thus, they are maintainable and suitable for long-term support compared to High-

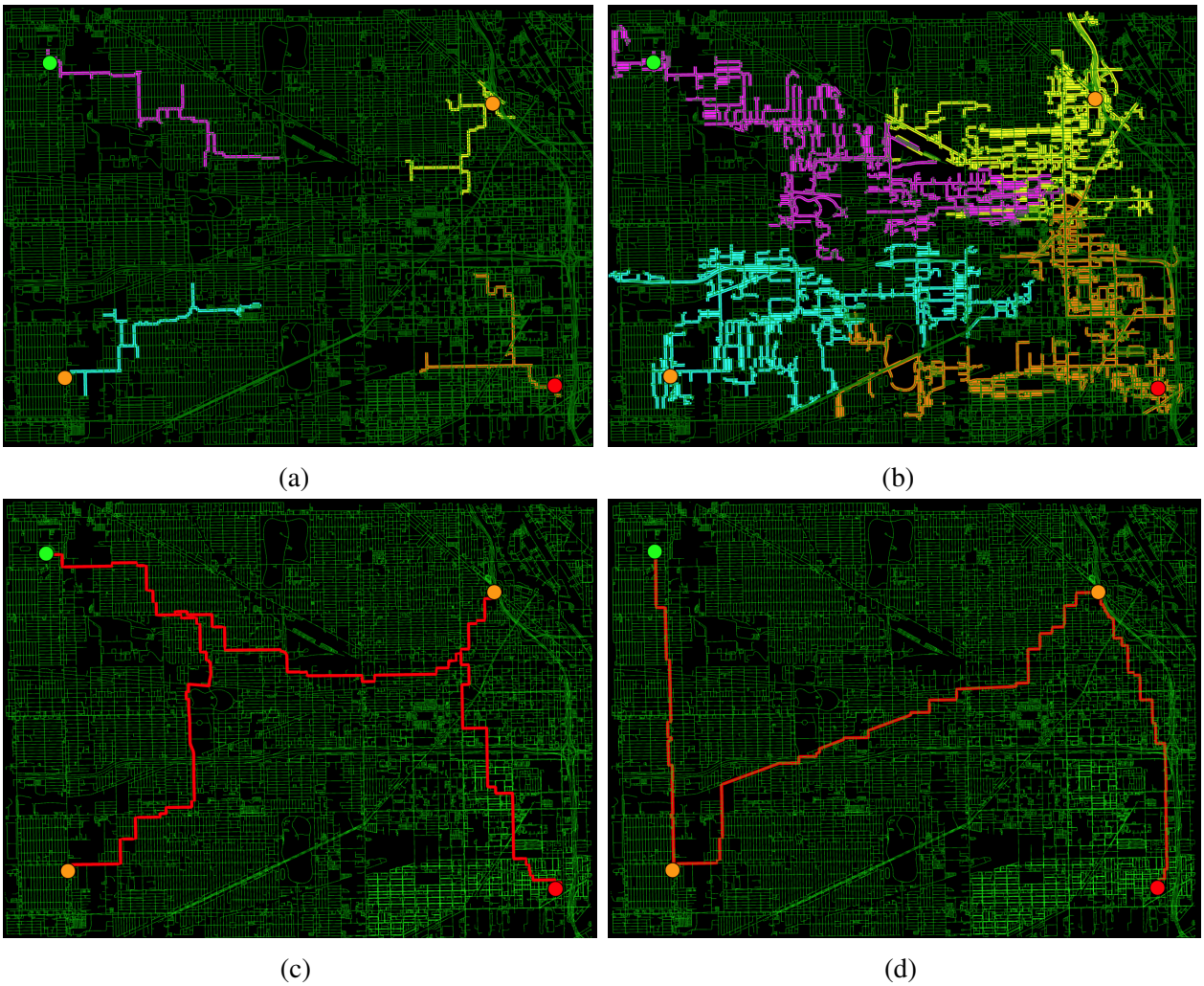


Figure 8.1: Illustration of the proposed informable multi-objective and multi-directional RRT* (IMOMD-RRT*) system evaluated on OpenStreetMap of Chicago, containing $866,089$ nodes and $1,038,414$ edges. The green, red, and orange dots are the source, target, and objectives, respectively. (a) shows the initial stage of the tree expansion of each destination. (b) shows the trees from each destination form a connected graph. (c) shows the first path and visiting order from the IMOMD-RRT*. Given more computation time, (d) shows that IMOMD-RRT* returns a better path and order.

Definition (HD) maps. Graph-based maps such as OpenStreetMap [173] have been developed over the past two decades to describe topological features and are readily available worldwide. Therefore, we concentrate on developing the proposed informable multi-objective and multi-directional Rapidly Exploring Random Trees (RRTs) (RRT*) system for path planning on large and complex graphs.

A multi-objective path planning system is charged with two tasks: **1)** find weighted paths (i.e., paths and traversal costs), if they exist, that connect the various destinations. This operation results in an undirected and weighted graph where nodes and edges correspond to destinations and paths connecting destinations, respectively. **2)** determine the visiting order of destinations that minimizes total travel cost. The second task, called relaxed TSP, differs from standard TSP in that we are allowed to (or sometimes have to) visit a node multiple times; see Sec. 8.2.3 for a detailed discussion.

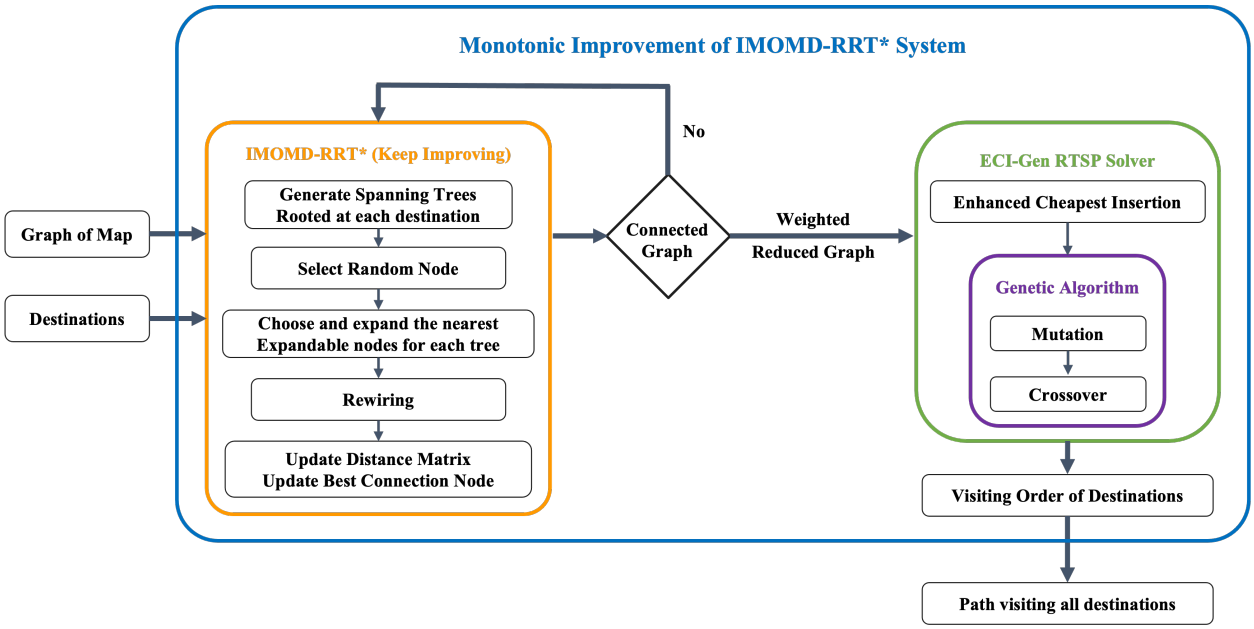


Figure 8.2: Illustration of the proposed IMOMD-RRT* system. It consists of an anytime informable multi-objective and multi-directional RRT* (IMOMD-RRT*) algorithm to construct a connected weighted-undirected graph, and a polynomial-time solver to solve the relaxed TSP. The solver consists of an enhanced version of the cheapest insertion algorithm [178] and a genetic algorithm [179–181], called ECI-Gen solver. The full system (the blue box) will continue to run to further improve the solution over time.

Several approaches [174–177] have been developed to solve each of these tasks separately, assuming either the visiting order of the destinations is given, or a cyclic/complete graph is constructed and the weights of edges are provided. However, in real-time applications, the connectivity of the destinations and the weights of the paths between destinations (task 1) as well as the visiting order of the destinations (task 2) are often unknown. It is crucial that we are able to solve these two tasks concurrently in an anytime manner, meaning that the system can provide suboptimal but monotonically improving solutions at any time throughout the path cost minimization process.

In this chapter, we seek to develop an anytime iterative system to provide paths between multiple objectives and to determine the visiting order of destinations; moreover, the system should be informable meaning it can accommodate prior knowledge of intermediate nodes, if available. The proposed system consists of two components: **1)** an anytime informable multi-objective and multi-directional RRT* (IMOMD-RRT*) algorithm to form a connected weighted-undirected graph, and **2)** a relaxed TSP solver that consists of an enhanced version of the cheapest insertion algorithm [178] and a genetic algorithm [179–182], which together we call ECI-Gen. The proposed system is evaluated on large-complex graphs built for real-world driving applications, such as the OpenStreetMap of Chicago containing $866,089$ nodes and $1,038,414$ edges that is shown in Fig. 8.1.

The overall system is shown in Fig. 8.2 includes the following contributions:

1. An anytime informable multi-objective and multi-directional RRT* (IMOMD-RRT*) system that functions on large-complex graphs. The anytime features means that the system quickly constructs a path on a large-scale undirected weighted graph that meets the existence constraint (the solution path must passes by all the objectives at least once), and the order constraint (with fixed starting point and end point). Therefore, the resulting weighted reduced graph containing the objectives, source, and target is connected.

2. The problem of determining the visiting order of destinations of the connected graph is a relaxed TSP (R-TSP) with fixed start and end nodes, though intermediate nodes can be (or sometimes must be) visited more than once. We introduce the ECI-Gen solver, which is based on an enhancement of the cheapest insertion algorithm [178] and a genetic algorithm [179–182] to solve the R-TSP in polynomial time.

3. We show that prior knowledge (such as a reference path) for robotics inspection of a pipe or factory can be readily and inherently integrated into the IMOMD-RRT*. In addition, providing the prior knowledge to the planner can help navigate through challenging topology.

4. We evaluate the system comprised of IMOMD-RRT* and the ECI-Gen solver on large-scale graphs built for real world driving applications, where the large number of intermediate destinations precludes solving the ordering by brute force. We show that the proposed IMOMD-RRT* outperforms bi-directional A* [183] and ANA* [184] in terms of speed and memory usage in large and complex graphs. We also demonstrate by providing a reference path, the IMOMD-RRT* escapes from bug traps (e.g., single entry neighborhoods) in complex graphs.

5. We open-source the multi-threaded C++ implementation of the system at https://github.com/UMich-BipedLab/IMOMD-RRTStar.

The remainder of this chapter is organized as follows. Section 9.2 summarizes the related work. Section 8.3 explains the proposed anytime informable multi-objective and multi-directional RRT* to construct a simple connected graph. The ECI-Gen solver to determine the ordering of destinations in the connected graph is discussed in Sec. 8.4. Experimental evaluations of the proposed system on large-complex graphs are presented in Sec. 8.5. Finally, Sec. 8.6 concludes the chapter and provides suggestions for future work.

8.2 Related Work

Path planning is an essential component of robot autonomy. In this section, we review several types of path planning algorithms and techniques to improve their efficiency. Furthermore, we

compare the proposed system with existing literature on car-pooling/ride-sharing and the traveling salesman problem.

8.2.1 Common Path Planners

Path planners are algorithms to find the shortest path from a single source to a single target. Graph-based and sampling-based algorithms are the two prominent categories.

Graph-based algorithms [183–192] such as Dijkstra [186] and A* [185] discretize a continuous space to an undirected graph composed of nodes and weighted edges. They are popular for their efficiency on low-dimensional configuration spaces and small graphs. There are many techniques [183, 184, 187–189] to improve their computation efficiency on large graphs. Inflating the heuristic value makes the A* algorithm likely to expand the nodes that are close to the goal and results in sacrificing the quality of solution. Anytime Repairing A* (ARA*) [188] utilizes weighted A* and keeps decreasing the weight parameter at each iteration, and therefore leads to a better solution. Anytime Non-parametric A* (ANA*) [184] is as efficient as ARA* and spends less time between solution improvements. R* [187] is a randomized version of A* to improve performance. Algorithms such as Jumping Point Search [189] to improve exploration efficiency only work for grid maps. However, graph-based algorithms inherently suffer from bug traps, whereas sampling-based methods can overcome bug traps more easily via informed sampling; see Sec. 8.5 for a detailed discussion.

Sampling-based algorithms such as Rapidly Exploring Random Tree (RRT) [193] stand out for their low complexity and high efficiency in exploring higher-dimensional, continuous configuration spaces. Its asymptotically optimal version – RRT* [44, 194, 195] – has also gained much attention and has contributed to the spread of the RRT family. More recently, sampling-based algorithms on discrete spaces such as RRLT and d-RRT* have been applied to multi-robot motion planning [196–199]. We seek to leverage RRT* to construct a simple connected graph that contains multiple destinations from a large-complex map, as well as to accommodate prior knowledge of a reference path.

8.2.2 Car-Pooling and Ride-Sharing

Problems such as car-pooling, ride-sharing, food delivery, or combining public transportation and car-pooling handle different types of constraints such as maximum seats, time window, battery charge, number of served requests along with multiple destinations [164–172, 200–203]. These problems are usually solved by Genetic Algorithms [164], Ant Colony Optimization [167, 203], Dynamic Programming (DP) [172, 200] or reinforcement learning [166, 202]. These methods assume, however, that weighted paths between destinations in the graph are already known [164,

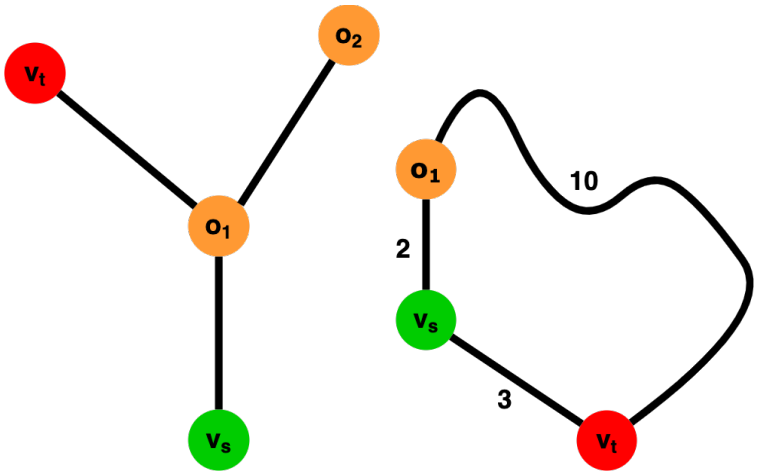


Figure 8.3: (a) shows a *simple connected graph* where the objectives have to be visited twice to visit all destinations. (b) shows the case where revisiting the source v_s allows a shorter path when triangular inequality does not hold.

166–168, 171, 200, 203], while in practice, the connecting paths and their weights are unknown and must be constructed.

8.2.3 Multi-objective Path Planners and Traveling Salesman Problem

Determining the travel order of nodes in an undirected graph with known edge weights is referred to as a Traveling Salesman Problem (TSP). The TSP is a classic NP-hard problem about finding a shortest possible cycle that visits every node exactly once and returns to the start node [162, 163]. Another variant of TSP, called the shortest Hamiltonian path problem [162], is to find the shortest path that visits all nodes exactly once between a fixed starting node (v_s) and a fixed terminating node (v_t). The problem can be solved as a standard TSP problem by assigning sufficiently large negative cost to the edge between v_s and v_t [162, 163].

We are inspired by the work of [174, 176], where the authors propose to solve the problem through a single-phase algorithm in simulated continuous configuration spaces. They first leverage multiple random trees to solve the multi-goal path planning problem and then solve the TSP via an open-source solver. In particular, they assume that the paths between nodes in the continuous spaces can be constructed in single pass and that the resulting graph can always form a cycle in their simulation environment. Thus, the problem can then be solved by a traditional TSP solver. In practice, however, the graph might not form a cycle (e.g., an acyclic graph or a forest) as shown in Fig. 8.3(a), and even if the graph is cyclic or there exists a Hamiltonian path, there is no guarantee the path is the shortest. In Fig. 8.3(b), the Hamiltonian path simply is $v_s \rightarrow o_1 \rightarrow v_t$, and the traversed distance is 12. However, another shorter path exists if we are allowed to traverse a node (v_s in this case) more than once: $v_s \rightarrow o_1 \rightarrow v_s \rightarrow v_t$ and the distance is 7. We therefore propose a

polynomial-time solver for this relaxed TSP problem; see Sec. 8.4 for further discussion.

8.3 Informable Multi-Objecticve and Multi-Directional RRT*

This section introduces an anytime informable multi-objective and multi-directional Rapidly-exploring Random Tree* (IMOMD-RRT*) algorithm as a real-time means to quickly construct from a large-scale map a weighted undirected graph that meets the existence constraint (the solution trajectory must pass by all the objectives at least once), and the order constraint (with fixed starting point and end point). In other words, the IMOMD-RRT* forms a simple[1] connected graph containing the objectives, source, and target.

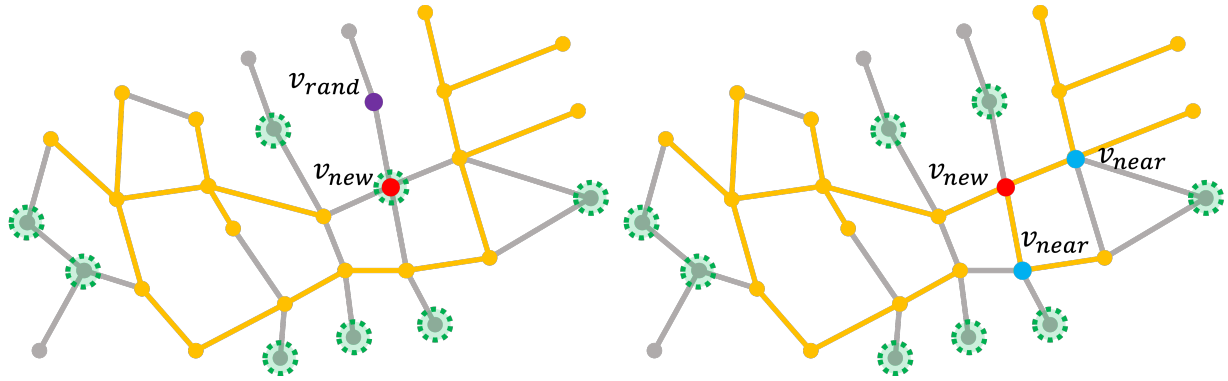
8.3.1 Standard RRT* Algorithm

The original RRT* [194] is a sampling-based planner with guaranteed asymptotic optimality in continuous configuration spaces. In general, RRT* grows a tree where nodes are connected by edges of linear path segments. Additionally, RRT* considers nearby nodes of a newly extended node when choosing the best parent node and when rewiring the graph to find shorter paths for asymptotic optimality.

8.3.2 Multi-objective and Multi-directional RRT* on Graphs

In this chapter, we use *map* to refer to the input graph, which might contain millions of nodes, and use *graph* to refer to the graph composed of only the destinations including the source and target node. The proposed IMOMD-RRT* differs from the original RRT* in six aspects when growing a tree. First, the sampling is performed by picking a random v_{rand} in the map, and not from an underlying continuous space. The goal bias is not only applied to the target but also the source and all the objectives. Second, a steering function directly finds the closest expandable node as v_{new} to the random node v_{rand}, without finding the nearest node in the tree, as shown in Fig. 8.4(a). Note that instead of directly sampling from the set of expandable nodes, sampling from the map ameliorates the bias of sampling on the explored area. Third, the parent node is chosen from the nodes connected with the new node v_{new}, called the neighbor nodes. Among the neighbor nodes, the node that yields the lowest path cost from the root becomes the new node's parent. Fourth, the jumping point search algorithm [189] is also leveraged to speed up tree exploration. Fifth, the IMOMD-RRT* rewires the neighborhood nodes to minimize the accumulated cost from the root of a tree to v_{new}, as shown in Fig. 8.4(b). Lastly, if v_{new} belongs to more than one tree, this node is considered a connection node, which connects the path between destinations, as shown in Fig. 8.5.

[1] A simple graph, also called a strict graph, is an unweighted and undirected graph that contains no graph loops or multiple edges between two nodes [204].



(a) (b)

Figure 8.4: Illustration of tree expansion and connection nodes of a tree. The unexplored paths in the graph and the spanning tree are represented by the gray and yellow lines, respectively. The green-dashed circles are the expandable nodes. (a) shows the tree $\mathcal{T}_i$ extends to the v_{rand} by growing a node v_{new} to the closest expandable node. (b) shows the updated set $\mathcal{X}_i$ of expandable nodes and the tree is rewired around the v_{new}. The rewired nodes v_{near} are represented as the blue dots.

Our proposed graph-based RRT* modification is summarized below with notation that generally follows graph theory [204]. A graph $\mathcal{G}$ is an ordered triple $(\mathcal{V}(\mathcal{G}), \mathcal{E}(\mathcal{G}), \Phi_{\mathcal{G}})$, where $\mathcal{V}(\mathcal{G}) = \{v \in \zeta\}$ is a set of nodes in the robot state space ζ, $\mathcal{E}(\mathcal{G})$ is a set of edges (disjoint from $\mathcal{V}(\mathcal{G})$), and an indication function $\Phi_{\mathcal{G}}$ that associates each edge of $\mathcal{G}$ with an unordered pair (not necessarily distinct) of nodes of $\mathcal{G}$.

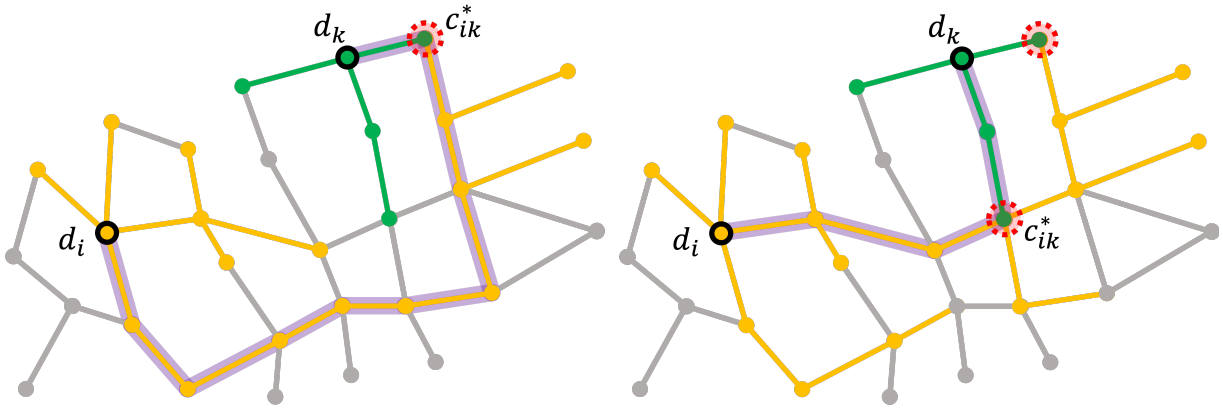
Given a set of destinations $\mathcal{D} = \{d_i | d_i \in \{v_s, v_t\} \cup \mathcal{O}\}_{i=1}^{m+2}$, where $v_s \in \mathcal{V}(\mathcal{G})$ is the source node, $v_t \in \mathcal{V}(\mathcal{G})$ is the target node, and $\mathcal{O} \subseteq \mathcal{V}(\mathcal{G})$ is the set of m objectives, the IMOMD-RRT* solves the multi-objective planning problem by growing a tree $\mathcal{T}_i = (V, E)$, where $V \subseteq \mathcal{V}(\mathcal{G})$ is a set of nodes connected by edges $E \subseteq \mathcal{E}(\mathcal{G})$, at each of the destinations $d_i \in \mathcal{D}$. Thus, it leads to a family of trees $\mathcal{T} = \{\mathcal{T}_1, \cdots, \mathcal{T}_m, \mathcal{T}_{m+1}, \mathcal{T}_{m+2}\}$.

The proposed IMOMD-RRT* explores the graph $\mathcal{G}$ by random sampling from $\mathcal{V}(\mathcal{G})$ and extending nodes to grow each tree. We explain a few important functions of IMOMD-RRT* below.

8.3.2.1 Tree Expansion

Let $\mathcal{N}(v)$ be the set of nodes directly connected with a node, i.e., the neighborhood of a node v [204]. A node is expandable if there exists at least one unvisited node connected and at least one node of the tree connected, shown as the dashed-green circles in Fig 8.4(a). Let $\mathcal{X}_i$ be the set of expandable nodes of the tree $\mathcal{T}_i$. A random node v_{rand} is sampled from the nodes of the graph $\mathcal{V}(\mathcal{G})$. Next, find the nearest node v_{new} in the set of expandable nodes $\mathcal{X}_i$:

$$v_{\text{new}} = \arg\min_{v \in \mathcal{X}_i} \texttt{Dist}(v, v_{\text{rand}}), \tag{8.1}$$



(a) (b)

Figure 8.5: Illustration of better connection nodes resulting in a better path. Two trees rooted at d_i and d_k are marked in yellow and green, respectively. The highlighted purple line shows the shortest path between d_i and d_k. The newly extended node v_{new} (dashed-red circles) is added to a set of connection nodes $\mathcal{C}_{ik}$. The element of the distance matrix, A_{ik}, and the connection node c^*_{ik} that generates the shortest path between the destination d_i and d_k are updated as a shorter path is found.

where $\texttt{Dist}(\cdot,\cdot)$ is the distance between two states. Next, the jumping point search algorithm [189] is utilized to speed up the tree expansion. If the current v_{new} has only one neighbor that is not already in the tree, v_{new} is added to the tree and that one neighbor is selected as the new v_{new}. This process continues until v_{new} has at least two neighbors that are not in tree, or it reaches v_{rand}.

8.3.2.2 Parent Selection

Let the set $\mathcal{N}_i(v_{\text{new}})$ be the neighborhood of the v_{new} in the tree $\mathcal{T}_i$. The node v_{near} in $\mathcal{N}_i(v_{\text{new}})$ that results in the smallest cost-to-come, $\texttt{Cost}(\cdot,\cdot)$, is the parent of the v_{new} and is determined by:

$$v_{\text{parent}} = \underset{v_{\text{near}} \in \mathcal{N}_i(v_{\text{new}})}{\arg\min} \{\texttt{Cost}(\mathcal{T}_i, v_{\text{near}}) + \texttt{Dist}(v_{\text{near}}, v_{\text{new}})\}. \tag{8.2}$$

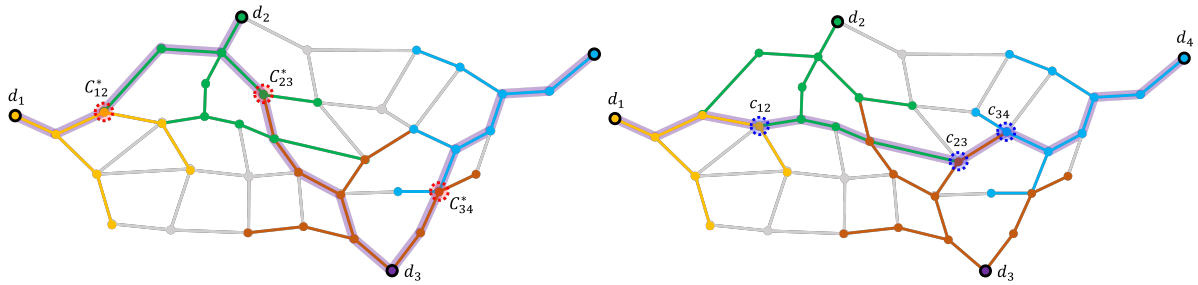
Next, all the unvisted nodes in $\mathcal{N}_i(v_{\text{new}})$ are added to the set of expandable nodes $\mathcal{X}_i$.

8.3.2.3 Tree Rewiring

After the parent node is chosen, the nearby nodes are rewired if a shorter path reaching the node through the v_{new} is found, as shown in Fig. 8.4(b). The rewiring step guarantees asymptotic optimality, as with the classic algorithm.

8.3.2.4 Update of Tree Connection

A node is a connection node if it belongs to more than one tree. Let $\mathcal{C}_{ik}$ be the set of connection nodes between $\mathcal{T}_i$ and $\mathcal{T}_k$, and let c^*_{ik} denote the node that connects $\mathcal{T}_i$ and $\mathcal{T}_k$ with the shortest



(a) (b)

Figure 8.6: Illustration of the rewired path of pseudo-destinations. $d_1 - d_4$ are the destinations and marked in different colors. The dashed-red circles are the connection nodes between two trees. The thick purple line is the final path. (a) shows the resulting path as if $d_1 - d_4$ are "true" destinations, and (b) is the resulting path as if $d_2 - d_3$ are "pseudo" destinations. The real destinations have to be visited as shown in (a), whereas pseudo destinations are artificial destinations to help form a connected graph, and might no longer be visited after rewiring

distance

$$c^*_{ik} = \underset{c \in \mathcal{C}_{ik}}{\arg\min} \{\texttt{Cost}(\mathcal{T}_i, c) + \texttt{Cost}(\mathcal{T}_k, c)\}. \tag{8.3}$$

Let $A_{(m+2)\times(m+2)}$ be a distance matrix that represents pairwise distances between the destinations, where m is the number of objectives. The element $A_{i,k}$ indicates the shortest path between destinations d_i and d_k, as shown in Fig. 8.5(b). $A_{i,k}$ is computed as

$$A_{i,k} = \texttt{Cost}(\mathcal{T}_i, c^*_{ik}) + \texttt{Cost}(\mathcal{T}_k, c^*_{ik}). \tag{8.4}$$

8.3.3 Discussion of Informability

As mentioned in Sec. 8.1, applications such as robotic inspection or vehicle routing might consider prior knowledge of the path, so that the robot can examine certain equipment or area of interests or avoid certain areas in a factory. The prior knowledge can be naturally provided as a number of "pseudo destinations" or samples in the IMOMD-RRT*. A pseudo destination is an artificial destination to help IMOMD-RRT* to form a connected graph. However, unlike *true* destinations that will always be visited, a pseudo destination might not be visited after rewiring, as shown in Fig. 8.6. Prior knowledge through pseudo destinations can also be leveraged to traverse challenging topology, such as bug-traps; see Sec. 8.5.

Remark 28. *One can decide if the order of the pseudo destinations should be fixed or even the pseudo destinations should be objectives (i.e., they will not be removed in the rewiring process.).*

8.4 Enhanced Cheapest Insertion and Genetic Algorithm

This section introduces a polynomial-time solver for the Relaxed Traveling Salesman Problem (R-TSP).

8.4.1 Relaxed Traveling Salesman Problem

The R-TSP differs from standard TSP [162, 163, 205] in two perspectives. First, nodes are allowed to be visited more than once, as mentioned in Sec. 8.2.3. Second, we have a source node where we start and a target node where we end. Therefore, the R-TSP can also be considered a relaxed Hamiltonian path problem [162, 205]. We propose the ECI-Gen solver, which consists of an enhanced version of the cheapest insertion algorithm [178] and a genetic algorithm [179–181] to solve the R-TSP. The complexity of the proposed solver is $O(N^3)$, where N is the cardinality of the destination set $\mathcal{D}$.

8.4.2 Graph Definitions and Connectivity

In graph theory [204, 206], a graph is simple or strict if it has no loops and no two edges join the same pair of nodes. In addition, a path is a sequence of nodes in the graph, where consecutive nodes in the sequence are adjacent, and no node appears more than once in the sequence. A graph is connected if and only if there is a path between each pair of destinations. Once all the destinations form a simple-connected graph, there exists at least one path π that passes all destinations $\mathcal{D}$. We can then consider the problem as an R-TSP (see Sec. 8.4.1), where we have a source and target node as well as several objectives to be visited. Therefore, we impose the graph connectivity and simplicity as sufficient conditions to solve the R-TSP. The disjoint-set data structure [207, 208] is implemented to verify the connectivity of a graph.

8.4.3 Enhanced Cheapest Insertion Algorithm

The regular cheapest insertion algorithm [178] provides an efficient means to find a sub-optimal sequence that guarantees less than twice the optimal sequence cost. However, it does not handle the case where revisiting the same node makes a shorter sequence. Therefore, we propose an enhanced version of the cheapest insertion algorithm, which comprises of a set of actions: **1)** in-sequence insertion, $\lambda_{\text{in-sequence}}$, which is the regular cheapest insertion; **2)** in-place insertion, $\lambda_{\text{in-place}}$, to allow the algorithm to revisit existing nodes; and **3)** swapping insertion, $\lambda_{\text{swapping}}$, which is inspired by genetic algorithms. Finally, sequence refinement is performed at the end of the algorithm.

Let the current sequence be $\mathcal{S}_{\text{current}} = \{v_s, s_1, \cdots, s_i, s_{i+1}, \cdots, s_n, v_t\}$ to indicate the visiting order of destinations, where $s_{\{\cdot\}} \in \mathcal{D}$ and $(n+2)$ is the number of destinations in the sequence.

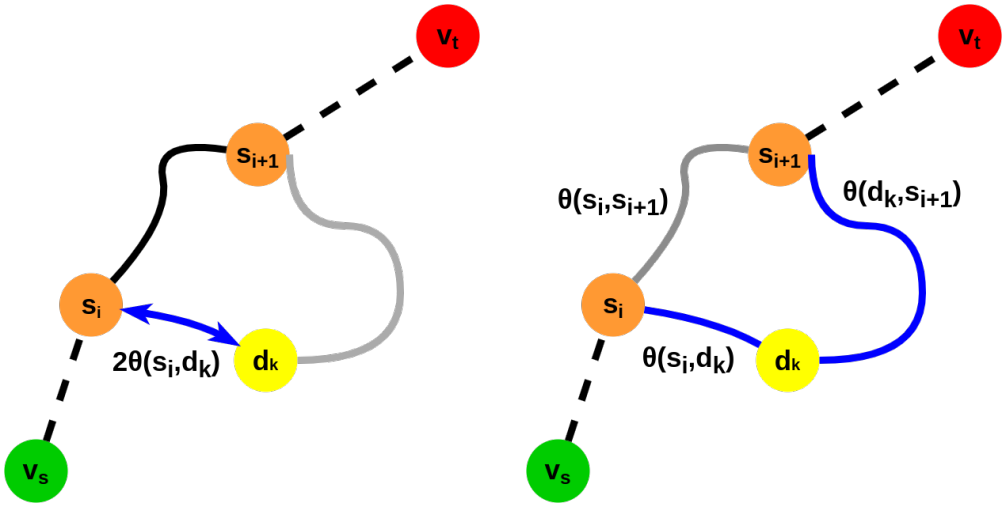


Figure 8.7: Illustration of the insertion cost. (a) shows the in-place insertion and the resulting sequence contains duplicated s_i. (b) shows the in-sequence insertion without a duplicated element.

The travel cost $\theta(\cdot,\cdot)$ is the path distance between two destinations provided by the IMOMD-RRT*. The $\mathcal{S}_{\text{current}}$ is constructed by Dijkstra's algorithm on the graph.

Remark 29. *$\mathcal{S}_{current}$ possibly contains duplicated elements and s_i is not necessary d_i. Therefore, the number of destinations in the sequence n may be larger than the actual number of destinations m. Fig. 8.7(a) shows the duplicated case with $\mathcal{S}_{current} = \{v_s, s_1, \cdots, s_i, d_k, s_i, s_{i+1}, \cdots, s_n, v_t\}$, where s_i is duplicated. Figure 8.7(b) illustrates an unduplicated case where $\mathcal{S}_{current} = \{v_s, s_1, \cdots, s_i, d_k, s_{i+1},$ $\cdots, s_n, v_t\}$.*

Let $\mathcal{K}$ denote the set of destinations to be inserted, and let the to-be-inserted destination be d_k and its ancestor be s_i, where $d_k \in \mathcal{K}$ and $s_i \in \mathcal{S}_{\text{current}}$. Given a current sequence $\mathcal{S}_{\text{current}}$, the location to insert s^*, and the action of insertion λ^* are determined by

$$\lambda^*, s^* = \underset{\substack{\lambda_j \in \Lambda \\ s_i \in \mathcal{S}_{\text{current}}}}{\arg\min}\, \lambda_j(s_i, d_k), \tag{8.5}$$

where $\Lambda = \{\lambda_{\text{in-sequence}}, \lambda_{\text{in-place}}, \lambda_{\text{swapping}}\}$ is the set of the insertion actions.

8.4.3.1 In-place Insertion $\lambda_{\text{in-place}}$

This step detours from s_i to d_k, and the resulting sequence is $\mathcal{S}_{\text{modified}} = \{v_s, s_1 \cdots, s_i, d_k, s_i, s_{i+1}, \cdots, s_n, v_t\}$, as shown in Fig. 8.7(a). The insertion distance is

$$\lambda_{\text{in-place}}(s_i, d_k) = 2\theta(s_i, d_k). \tag{8.6}$$

8.4.3.2 In-sequence Insertion $\lambda_{\text{in-sequence}}$

This step inserts d_k between s_i and s_{i+1} ($\forall s_i \in \{\mathcal{S}_{\text{current}}/(v_t)\}$), and the resulting sequence is $\mathcal{S}_{\text{modified}} = \{v_s, s_1, \cdots, s_i, d_k, s_{i+1}, \cdots, s_n, v_t\}$, as shown in Fig. 8.7(b). The insertion distance is

$$\lambda_{\text{in-sequence}}(s_i, d_k) = \theta(s_i, d_k) + \theta(d_k, s_{i+1}) - \theta(s_i, s_{i+1}). \tag{8.7}$$

8.4.3.3 Swapping Insertion $\lambda_{\text{swapping}}$

The swapping insertion changes the order of nodes right next to the newly inserted node. There are three cases in swapping insertion: swapping left, right, or both. For the case of swapping left, the modified sequence is $\mathcal{S}_{\text{modified}} = \{v_s, s_1 \cdots, s_{i-2}, s_i, s_{i-1}, d_k, s_{i+1}, \cdots, s_n,$
$v_t\}$ by inserting d_k between s_i and s_{i+1} ($\forall s_i \in \{\mathcal{S}_{\text{current}}/(v_s, s_1)\}$), and then swapping s_i and s_{i-1}. The insertion distance of swapping left is

$$\begin{aligned}\lambda_{\text{swapping (left)}}(s_i, d_k) = \theta(d_k, s_{i+1}) - \theta(s_i, s_{i+1}) \\ + \theta(s_{i-1}, d_k) + \theta(s_{i-2}, s_i) \\ - \theta(s_{i-2}, s_{i-1}).\end{aligned} \tag{8.8}$$

The right swap is a similar operation except that it swaps s_{i+1} and s_{i+2} instead. Lastly, the case of swapping both does a left swap (s_i and s_{i+1}) and then a right swap (s_{i+1} and s_{i+2}).

8.4.3.4 Sequence Refinement

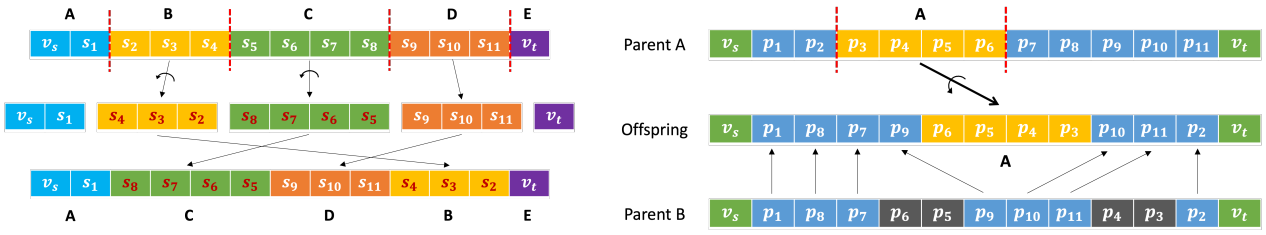
In-place insertion occurs when the graph is not cyclic or the triangular inequality does not hold on the graph. The in-place insertion could generate redundant revisited nodes in the final result and lead to a longer sequence. We further refine the sequence by skipping revisited destination when the previous destination and the next destination are connected. The refined sequence of destinations, $\mathcal{S}_{\text{ECI}}$, with cardinality $r \leqslant n$ is the input to the genetic algorithm.

8.4.4 Genetic Algorithm

We further leverage a genetic algorithm [179–181] to refine the sequence from the enhanced cheapest insertion algorithm. The genetic algorithm selects a parent[2] sequence and then generates a new offspring sequence from it by either a mutation or crossover process.

In brief, we first take the ordered sequence $\mathcal{S}_{\text{ECI}}$ from the enhanced cheapest insertion as our first and only parent for the mutation process, which produces multiple offspring. Only the offspring

[2]Note that the parent in the genetic algorithm is a different concept from the parent node in the RRT* tree, mentioned in Sec. 8.3.2. The terminology is kept so that it follows the literature consistently.



(a) Mutation (b) Crossover

Figure 8.8: Illustration of the mutation and crossover in the Genetic Algorithm. (a) In mutation, the sequence is randomly cut into five segments and each segment is randomly reversed except Segment A and Segment E. In this case, Segment B and C are reversed and D does not. Finally, the modified segments are spliced in random order and resulting an offspring ACDBE. (b) In crossover, two sequences (Parent A and Parent B) are picked from the offspring from the mutation process. A sub-sequence of one of the two sequences is randomly selected (Sub-sequence A in this case). Next, random inversion is performed on Sub-sequence A and the resulting segment is randomly placed inside an empty sequence of the offspring. Lastly, the remaining elements of the offspring are filled by the order of the other sequence (Parent B) except the elements that are already in the offspring sequence.

with a lower cost than the parent are considered the parents for the crossover process.

1) Mutation: There are three steps for each mutation process, as described in Fig. 8.8(a). First, the ordered sequence $\mathcal{S}_{\text{ECI}}$ from the enhanced cheapest insertion is randomly divided into k segments. Second, random inversion[3] is executed for each segment except the first and last segments, which contain the source and the target. Lastly, the segments in the middle are randomly reordered and spliced together. Let the resulting offspring sequence be $\psi = \{v_s, p_1, p_2, \cdots, p_r, v_t\}$, where v_s is the source node, v_t is the target node, $(r+2)$ is the number of destinations in the sequence (the same cardinality of S_{ECI}), and $\{p_i\}_{i=1}^{r}$ is the re-ordered destinations, which could possibly contain the start and end. The cost Θ of the offspring is computed as

$$\Theta = \theta(v_s, p_1) + \sum_{i=1}^{r} \theta(p_i, p_{i+1}) + \theta(p_n, v_t), \tag{8.9}$$

where $\theta(\cdot,\cdot)$ is the path distance between two destinations provided by the IMOMD-RRT*.

We perform the mutation process thousands of times, resulting in thousands of offspring. Note that only the offspring with a lower cost than the cost of the parent are kept for the crossover process.

2) Crossover: Let the set of mutated sequences from the mutation process be $\Psi = \{\psi_i\}_{i=1}^{h}$, where h is the number of offspring kept after the mutation process. For each generation, the crossover process is performed thousands of times and only the offspring with a lower cost than the previous generation are kept. Each crossover process combines sub-sequences of any two sequences $(\psi_i, \psi_j \in \Psi)$ to generate a new offspring, as described in Fig.8.8(b). The probability of a sequence

[3]A random inversion of a segment is reversing the order of destinations in the segment.

ψ_i being picked is defined as:

$$P_\psi(\psi = \psi_i) = \frac{\rho_i}{\sum_{i=1}^{w} \rho_i}, \;\; \rho_i = \frac{1}{\Theta_i}, \tag{8.10}$$

where w is the number of the remaining offspring from each generation after the $(i-1)^{\text{th}}$ generation and ρ_i is the fitness of the sequence ψ_i.

Given the two selected sequences and an empty to-be-filled offspring, a segment of one of the two sequences is randomly selected, and random inversion is performed on the segment. The resulting segment is randomly placed inside the empty sequence of the offspring. Lastly, the remaining elements of the offspring are filled by the order of the other sequence except the elements that are already in the offspring sequence. After a few generations, the offspring with the lowest cost, ψ^*, is the final sequence $\mathcal{S}_{\text{ECI-Gen}}$ of the destinations.

Remark 30. *Whenever the IMOMD-RRT* provides a better path (due to its asymptotic optimality), the ECI-Gen solver will be executed to solve for a better visiting order of the destinations. Therefore, the full system provides paths with monotonically improving path cost in an anytime fashion.*

8.4.5 Discussion of Time Complexity

As mentioned in Sec. 8.4.3, the first sequence $\mathcal{S}_{\text{current}}$ is constructed by Dijkstra's algorithm, which is an $O(N^2)$ process, where N is the cardinality of the destination set $\mathcal{D}$. We then pass the sequence to the enhanced cheapest insertion algorithm, whose time complexity is $O(N^3)$, to generate a set of parents for the genetic algorithm, which is also an $O(N^3)$ process. Therefore, the overall time complexity of the proposed ECI-Gen solver is $O(N^3)$, and indeed a polynomial solver. The benchmark of the proposed solver is presented in Appendix B.1.

8.5 Experimental Results

This section presents extensive evaluations of the IMOMD-RRT* system applied to two complex vehicle routing scenarios. The robot state ζ is defined as latitude and longitude. The distance between robot states $\texttt{Dist}(\cdot, \cdot)$ in (8.1) is defined as the haversine distance [209]. We implemented the bi-directional A* [183] and ANA* [184] as our baselines to compare the speed and memory usage (the number of explored nodes). We evaluate the IMOMD-RRT* system (IMOMD-RRT* and the ECI-Gen solver) on a large and complex map of Seattle, USA. The map contains $1,054,372$ nodes and $1,173,514$ edges, and is downloaded from OSM, which is a public map service built for real applications[4]. We then place 25 destinations in the map. We demonstrate that

[4]Apple Map® actually uses OpenStreetMap (OSM) as their foundation.

Table 8.1: Quantitative results of the proposed IMOMD-RRT* system on two large maps (both graphs contain more than one million nodes and edges) built for real robotics and vehicle applications. The proposed system outperforms bi-A* and ANA*.

		Initial Solution Time [seconds]	Initial Path Cost [kilometers]	Final Memory Usage [# explored nodes]
Seattle	IMOMD-RRT*	**0.44**	**501,342**	**49,768**
	Bi-A*	4.40	808,416	3,240,515
	ANA*	1.70	1,089,873	234,457
San Francisco	IMOMD-RRT*	**1.10**	**156,807**	**61,785**
	Bi-A*	9.93	315,061	3,640,863
	ANA*	Failed	Failed	Failed

the IMOMD-RRT* system is able to concurrently find paths connecting destinations and determine the order of destinations. We also show that the system escapes from a bug trap by inherently receiving prior knowledge. The algorithm runs on a laptop equipped with Intel® Core™ i7-1185G7 CPU @ 3.00 GHz.

To show the performance and ability of multi-objective and determining the visiting order, we randomly set 25 destinations in the Seattle map. There are 25! possible combinations of visiting orders and therefore it is intractable to solve the visiting order by brute force. The results are shown in Fig. 8.9, where IMOMD-RRT* finds the first path faster than both Bi-A* and ANA* with a lower cost and then also spends less time between solution improvements. Additionally, the memory usage of IMOMD-RRT* is less than ANA* and much less than bi-A*. As shown in Table 8.1, the proposed system provides the first solution 10 times faster than bi-A* and four time faster than ANA*. In addition, the proposed system also consumes 65 times less memory than bi-A* and 4.7 times less memory usage than ANA*.

Prior knowledge through pseudo destinations can also be leveraged to traverse challenging topology, such as bug-traps [210]. This problem is commonly seen in man-made environments such as a neighborhood with a single entry or cities separated by a body of water, as in Fig. 8.10. As mentioned in Remark 8.3.3, prior knowledge is provided as a number of pseudo destinations in the IMOMD-RRT* as a prior collision-free path in the graph for robotics inspection or vehicle routing. Next, the prior path is then being rewired by the IMOMD-RRT* to improve the path. We demonstrate this feature by providing the prior knowledge to escape the bug trap in San Francisco, as shown in Fig. 8.10. The map contains $1,277,702$ nodes and $1,437,713$ edges. As shown in Table 8.1, the proposed system escapes from the trap nine times faster than bi-A*, whereas ANA* failed to provide a path within the given time frame. The proposed system also consumes 58.9 times less memory than bi-A*.

In summary, we developed an anytime iterative system to provide paths between multiple objectives by the IMOMD-RRT* and to determine the visiting order of the objectives by the ECI-

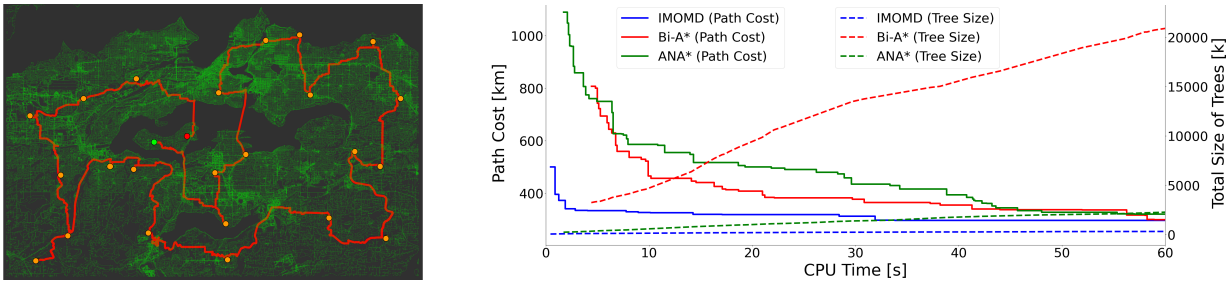


Figure 8.9: Quantitative and qualitative results for an OSM of Seattle, where we have 25 destinations to be visited. The proposed IMOMD-RRT* outperforms Bi-A* and ANA* in term of speed and memory usage (the number of explored nodes).

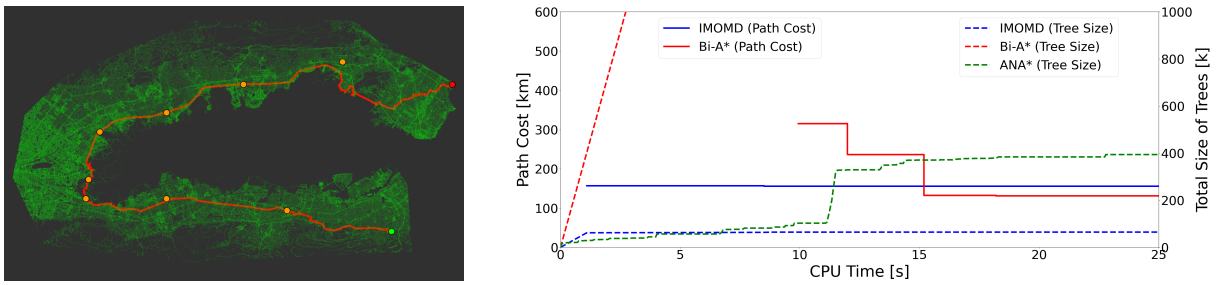


Figure 8.10: Providing prior knowledge to the proposed IMOMD-RRT* system to avoid bug traps. The left and the right are the qualitative and quantitative results for a bug trap in San Francisco, respectively. We have eight pseudo destinations to help escape the challenging topology, where the source and target are separated by a body of water. Note that ANA* failed to provide a solution in the given time.

Gen solver solver in polynomial time. We also demonstrate that the proposed system is able to inherently accommodate prior information to escape from challenging topology.

8.6 Conclusion and Future Work

We presented an anytime iterative system on large-complex graphs to solve the multi-objective path planning problem, to decide the visiting order of the objectives, and to incorporate prior knowledge of the potential trajectory. The system is comprised of an anytime informable multi-objective and multi-directional RRT* to connect the destinations to form a connected graph and the ECI-Gen solver to determine the visiting order (via a relaxed Traveling Salesman Problem) in polynomial time.

The system was extensively evaluated on OpenStreetMap (OSM), built for autonomous vehicles and robots in practice. In particular, the system solved a path planning problem and the visiting order with 25 destinations (25! possible combinations of visiting orders) on an OSM of Seattle, containing more than a million nodes and edges, in 0.44 seconds. In addition, we demonstrated the system is able to leverage a reference path (prior knowledge) to navigate challenging topology for robotics inspection or vehicle routing applications. All the evaluations show that our proposed method outperforms the Bi-A* and ANA* algorithm in terms of speed and memory usage.

In the future, we shall use the developed system within autonomy systems [9, 21, 39, 43, 44, 51, 61, 211–216] on a robot to perform point-to-point tomometric navigation in graph-based maps while locally avoiding obstacles and uneven terrain. It would also be interesting to deploy the system with multi-layered graphs and maps [50, 217, 218] to incorporate different types of information.