From Scene Representation to Robust Autonomy: A seamless Integration

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering by Research

by

Sudarshan S Harithas
2021701008
sudarshan.s@research.iiit.ac.in

International Institute of Information Technology
(Deemed to be University)
Hyderabad - 500 032, INDIA
June 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " **From Scene Representation to Robust Autonomy: A seamless Integration**" by **Sudarshan S Harithas**, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Dr. K. Madhava Krishna

To *My Parents*

Acknowledgments

I would like to thank my advisor Prof. K. Madhava Krishna for is support and guidance throughout my journey at the *Robotics Research Center*. I would like to express my deepest gratitude for *Prof. Arun Kumar Singh* from the *Institute of Technology, University of Tartu* in *Estonia*, for imparting key novel ideas that greatly contributed to my research. I would further like to thank *Prof. Chetan Arora* from *IIT Delhi* for his valuable insights and guidance. This thesis is a consequence of the collective efforts and support of these esteemed professors and I am extremely grateful for their guidance and mentorship.

The experience at *IIIT-H* would not have been the same without Rishabh Dev Yadav, we share a strong friendship from our internship days and throughout our master's research. I am grateful for his unwavering support and invaluable contributions as a team member. I am deeply grateful to Swati Dantu for her tireless efforts in editing my research videos. These videos received high praise from reviewers and greatly contributed to the success of the project. In addition, I extend my sincere gratitude to Jhanvi Shingala for her thorough review of my papers prior to submission.

My team mates have consistently brought in fresh perspectives which have greatly helped to speed up research and improve the output. I would like to thank Gurkirat Singh, Bitla Bhanu Teja, Aneesh Chavan, Deepak Singh, Suraj Patni for all their contributions in this wonderful journey.

My seniors at RRC have been extremely generous in imparting their knowledge, sharing experience and patient in answering all my questions. I am extremely grateful to Karnik Ram, VVSST Ayyappa Swamy and Sarthak Sharma for their mentorship and guidance.

Most importantly I would like to thank my family for their constant love and support without which this work would not be possible today.

Abstract

The objective of this study is to examine the interdependence between scene representation and robust autonomy of mobile robots such as quadrotors and cars. It addresses crucial issues with the motion planning and SLAM components in the navigation system.

Voxel Grids built from range sensors have been a popular choice of world representation for motion planning. The planners query the map for distance to collision and gradients, the planner uses these measurements to generate a smooth and safe trajectory through trajectory optimization. However, these methods assume the presence of zero sensor noise, such an assumption is severely violated when the maps are built through RGBD cameras, the noise in the point clouds are transferred to the occupancy maps through a non-linear transform and is further propagated to the distance and gradient field which results in a non-parametric noise. The performance of the deterministic planners that rely on such inaccurate gradients was found to decrease. As an antidote we propose **CCO-VOXEL** the very first chance-constrained optimization (CCO) algorithm that can compute trajectory plans with probabilistic safety guarantees in real-time directly on the voxel-grid representation of the world. We leverage the notion of Hilbert Space embedding of distributions and Maximum Mean Discrepancy (MMD) and gradient free *Cross-Entropy Method* for obtaining its minimum. We also show how a combination of low-dimensional feature embedding and pre-caching of Kernel Matrices of MMD allows us to achieve real-time performance in simulations as well as in implementations on on-board commodity hardware that controls the quadrotor flight.

Voxel maps are not robust to outliers and fail to capture the world geometry when built from a highly noisy and sparse point cloud, such as those generated from triangulation through monocular *visual-inertial SLAM* (*VI-SLAM*). Inaccurate maps like these can significantly decrease the performance of the motion planner. We propose **UrbanFly** , a quadrotor planning framework for navigation in a planar urban high-rise environment. The core importance of *UrbanFly* is its ability to handle sparse and noisy point clouds built from *VI-SLAM*. Based on the insight that the sky-scrappers are planar, we derived an *Uncertainty-Integrated Cuboidal (UIC)* representation of the world and developed an uncertainty-aware planner that can interpret and leverage the benefits of *UIC* to generate safe trajectories in real time. UrbanFly demonstrates a 3.5 times improvement in safety metrics compared to state-of- the-art algorithms.

Points clouds generated from LiDAR need a lot of memory to store and transfer and lack structure, making it hard to extract spatial features. This makes it challenging to deploy lidar based *Loop Detection and Closure* (LDC) for distributed collaborative SLAM. To address this, we present **FinderNet**, a 6-DOF

distributed LDC system. We develop a novel DEM representation to reveal the underlying geometric structure of the point clouds, and an auto-encoder decoder structure to compress the point clouds. LDC is performed in the compressed space, allowing deployment in a multi-agent setting. Our approach provides view-invariance through a process of canonicalization and differentiable alignment, which is different from existing methods that rely on feature aggregation or overlap estimation. **FinderNet** has been evaluated on real-world datasets such as Kitti, GPR, and Oxford Robot Car.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile robots are increasingly being utilized in a diverse range of applications, including manufacturing, logistics, and rescue operations. For these robots to effectively carry out tasks and contribute to our daily lives, it is crucial that they have a robust autonomous navigation system. This system, or navigation stack, consists of three main components. First, the robot is equipped with sensors such as `LiDAR` and `RGBD` cameras that allow it to perceive the external world. This data is then processed by the *scene representation* layer, where algorithms are developed to create a computational model of the world and understand the 3D environment in which the robot operates. Finally, the *action module* interprets the world model to perform tasks such as state estimation (`SLAM`) and trajectory planning.

The navigation stack relies on a model of the world, rather than raw sensor data, to make decisions. However, until now *3D scene representation* and *navigation* have been treated as independent operations. Such an organization leads to sub-optimal performance for two reasons, **(1)** The navigation stack is *unaware* of the uncertainties and inaccuracies in the scene representation. **(2)** The available scene representations are not *action oriented* and lack rich mathematical properties that could be leveraged by the action module. To improve performance, it is necessary to adopt a more tightly coupled approach that integrates 3D scene representation and navigation. This work studies the interplay between the *3D scene representation* and autonomous navigation, specifically enhancing the robustness of the motion planning and state estimation modules within the navigation system as detailed below.

1.0.1 Motion Planning

Quadrotor motion planning involves generating a collision free and dynamically feasible trajectory from start to goal. The planner relies on the *range sensors* within the perception system to estimate the presence of obstacles in the vicinity of the robot. `RGBD` cameras have become a favored perception modality for quadrotors due to their compact and lightweight design, additionally, *Voxel Grids* have emerged as a widely used world representation due to their simplicity in construction and scalability. However, often the point clouds generated through these `RGBD` cameras are noisy comprising of spurious points, and the voxel grids representation is not agnostic to the underlying noise within the point clouds, as depicted in Fig. 1.1 (a). Noisy occupancy maps would result in inaccurate measurements

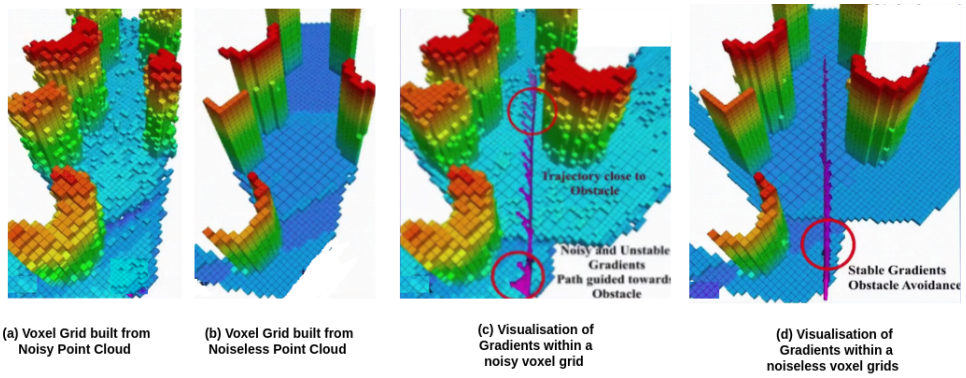


Figure 1.1: The point clouds obtained from RGBD cameras are noisy and voxel grid are not agnostic to the noise. (a) depicts the *Octomap* [29] (voxel grids) observe that the spurious points present in the point clouds are present in the voxel grids without any filtering. (c) The gradients derived from the noisy voxel grids are not consistent as in case of a noiseless map as depicted (d).

of the distance to the nearest obstacle and gradient fields. This, in turn, would adversely impact the performance of deterministic motion planners [82, 83, 81] as they rely on these measurements for optimizing trajectories. As a solution we propose *CCO-VOXEL* [27] (Section 3), an uncertainty aware motion planner that is agnostic to inaccuracies within the voxel grid representations.

The performance of the motion planners are further sensitive to the choice of perception modality. Although RGBD cameras are noisy it outputs a dense point clouds enabling the *Voxel Grids* to reconstruct a continuous environment. However, the point clouds generated through triangulation from monocular *VI-SLAM* are sparse and noisy, making a continuous reconstruction of the world impossible. Through our work in *UrbanFly* we demonstrate that voxel grids are not robust to handling outliers, and further propose an *Uncertainty Integrated Cuboidal* (UIC) a continuous planar representation that is robust to noise. We further develop an uncertainty aware planner that leverages the mathematical properties of the UIC to compute a safe and dynamically feasible trajectory.

1.0.2 State Estimation

The navigation stack's state estimation module is a critical component that determines the robot's pose at every moment and creates a mapped representation of the environment. Although LiDAR measurements are highly precise and have a long-range sensing capability, processing and extracting meaningful information from unstructured point clouds can be challenging. Additionally, the cost of storing and transmitting LiDAR point clouds limits their use in Collaborative SLAM. To address these challenges, we propose the DEM (Section 5.1.1), a representation that exposes the underlying structure of point clouds in 6-DOF motion and can be compressed to improve storage efficiency.

We utilize the advantages of DEM in LiDAR loop detection and closure for distributed collaborative SLAM through *FinderNet* [25]. Our encoder-decoder architecture enables compression-decompression of the DEM and transmission. Loop detection and closure operations are performed in the latent (com-

pressed) space of the point clouds. Unlike current literature that either uses feature aggregation [8, 74, 79, 68] for view invariance or estimates point cloud overlaps for LDC [9, 44], we provide an alternate approach where view invariance is achieved through canonicalization and differentiable alignment. For a detailed explanation of our methodology, refer to *Section* 4.

1.1 Contributions

The core contributions of the thesis can be summarized as given below:

Quadrotor Motion Planning

1. A new trajectory planner has been proposed [27] that ensures probabilistic safety and is independent of the obstacle's geometry and non-parametric noise.
2. The trajectory planning problem has been approached in a unique way by formulating it as a distribution matching problem, where the target distribution can be represented by the dirac function. Additionally, optimization is carried out using a gradient-free method known as *Cross Entropy Minimization*.
3. Extensive evaluation of [27] through simulation and real-world field testing.
4. Formulation of a novel *Uncertainty Integrated Cuboidal* representation for single camera based uncertainty aware quadrotor trajectory planning [26].

LiDAR SLAM

1. Formulation of *Loop Detection and Closure* (LDC) for LiDAR SLAM as a canonicalization and differentiable alignment problem [25].
2. Development of a multi-functional pipeline that performs LDC in the compressed space of point clouds with a 6-DOF displacement.
3. Development of attention based self-supervised modules that enables training with minimal supervision and eliminates data-augmentation requirements.
4. Integration of [25] into *Lio-SAM* [64] and evaluation of real world datasets.

1.2 Thesis Organization

This thesis studies the interactions between perceptual scene representation and robust autonomy, specifically we enhance the performance of the motion planning and state estimation modules with the navigation stack. The necessary background and terminologies used in the work is introduced in

Chapter 2. Chapter 3 is a detailed description of *CCO-VOXEL* [27] a gradient free uncertainty aware planner for navigation in a voxel-grid world. In Chapter 5 we introduce *UrbanFly*, a motion planning algorithm designed to generate safe trajectories for navigating among high-rise buildings. Our work related to *Loop Detection and Closure* (`LDC`) for `LiDAR SLAM` has been presented in chapter 4. Finally, the concluding remarks and suggestions for future research directions is presented in chapter 6.

Chapter 2

Background

2.1 Volumetric Mapping & Quadrotor Motion Planning

Volumetric mapping is used to generate a 3D model of the environment in which the robot operates. Typically, the environment is discretized into 3D grids known as voxels. Volumetric maps can store information such as occupancy, distance fields and derived measurements such as gradients. Motion planning is task of generating a trajectory from a given start to goal location, the planning algorithms use the underlying volumetric maps to detect obstacles and avoid them, furthermore, many methods use the distance fields and gradients to perform trajectory optimization for improved performance. In the context of motion planning this thesis explores the relation between the inaccurate volumetric maps and motion planning.

2.1.1 3D Occupancy maps

Occupancy maps indicate the probability that a given voxel in space is occupied, typically, these methods assume that the robot has an accurate state estimation systems and is equipped with a range measurement sensor such as `LiDAR` or `RGBD` cameras. The approach for integrating the range sensor measurements into the probabilistic map was given by [47], where the probability that a voxel n is occupied given a history of sensor reading $z_{1:t}$ is given by Eq. 2.1, where $P(n)$ is probablity that a given voxel is occupied known as a prior (the initial value of $P(n) = 0.5$).

$$P(n/z_{1:t}) = \left[1 + \frac{1 - P(n/z_t)}{P(n/z_t)} \frac{1 - P(n/z_{1:t-1})}{P(n/z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{2.1}$$

Using the log-adds form of Eq. 2.1, the multiplications can be replaced with additions leading to faster updates, the resulting log-odds formulation is given by Eq. 2.2

$$L^o(n/z_{1:t}) = L^o(n/z_{1:t-1}) + L^o(n/z_t) \tag{2.2}$$

Where

$$L^o = \log \frac{P(n)}{1 - P(n)}$$

After the update of the occupancy, a threshold is set on $P(n/z_{1:t})$ to ultimately decide if the cell is occupied or free. However, Eq. 2.2 has a a major drawbacks, it can be observed that to change the state we must observe the same voxel has many times as it has been observed previously, such a requirement limits the real time adaptability of the method.

For real-time occupancy mapping [77] introduced Eq. 2.3, where the lower and upper limits of the log-adds value is given by l_{min} and l_{max} respectively. Such a formulation leads to better adaptability and enables compact representation [29].

$$L^o = \max\left(\min\left(L^o(n/z_{1:t-1}) + L^o(n/z_t), l_{max}\right), l_{min}\right) \tag{2.3}$$

Building upon the basic formulation of occupancy mapping given above, *Octomap* [29] builds a comapct *Octree* based data-structure for volumetric mapping. [29] checks for occupancy at multiple resolutions, it mode in the octree represents a voxel and each voxel has been subdivided into 8 equal sub-volumes until the least specified resolution is reached. For our work [27] we choose *Octomap* [29] to generate the volumetric map of the environment.

2.1.2 Distance Fields

While many sample based classical global planners such as *RRT*, *RRT**, *A** etc. have been developed that operate directly on the occupancy maps, these methods are not real-time. Modern real-time local planning algorithms [82, 67, 81] achieve real-time performance through trajectory optimization, such planning algorithms query the volumetric representation of the environment for distance and gradient measurements. A distance field is a continuous function represented as $f(x) : R^3 \mapsto R$, the function $f(x)$ return the distance from the query point x in R^3, to the closest point on the surface of the function. The motion planners query the continuous function $f(.)$ for the distance to collision measurement and to obtain the gradients to perform trajectory optimization.

Euclidean Signed Distance Field `ESDF` [51] is popular choice of distance fields representation, in an `ESDF` the distance to the closest surface is represented with opposite signs based on the position of query x relative to the surface (either can be inside a surface or outside), if x is on the surface a 0 is returned. Similar to `ESDF` is the *Euclidean Distance Transform* `EDT`, here all distance to all points outside the surface is represented by a positive sign and all points within and on the surface is assigned a 0. `ESDF` and `EDT` can be built given a occupancy map and in our implementation of [27] we use the `EDT` distance field built over an *Octomap* [29] as the underlying world representation.

2.1.3 Motion Planning for Quadrotors

Trajectory planning for quadrotor systems has a long history in the areas of optimal trajectory generation [19, 3, 46] and integrated perception and navigation [59, 69]. Gradient based planning over voxel representations have become popular in recent years [82, 81, 67, 83], these methods obtain a safe-trajectory through optimization. The general structure of such methods include an initial guess trajectory obtained from a sampling based planner that queries the environment only for occupancy (and not distances or gradients), this trajectory is further improved through gradient based trajectory optimization techniques.

Uncertainity aware motion planning: The gradient based trajectory optimizers assume zero sensor noise while creating the map, and in a noisy environment in the absence of consistent gradients the optimization methods fail to generate safe trajectories. Meanwhile, Chance Constrained Optimization with suitable surrogates too have been well studied, however their inadequacy in handling non-parametric uncertainty was detailed in [54, 23]. Specifically there appears no known methods that handle non parametric uncertainty of discretized volumetric voxel grids other than the proposed [27]. While [33] provides for a chance constrained framework for nearest distance to obstacle it formulates over well defined obstacle shapes and not over voxel grids. Moreover, its computation of safe trajectories runs into minutes while the proposed method [27] is in the order of tens of milliseconds.

Monocular SLAM based Planners: Literature dealing with trajectory planning and/or navigation with monocular SLAM is sparse when compared with a much larger volume of literature that deals with planning with SLAM [39]. A large part of planning with SLAM focuses on actively moving to places where the state uncertainty (often measured as the trace of the state co-variance) is low [39], [30]. In contrast, the Monocular SLAM frameworks tend to rely on heuristics [49] due to the difficulty of estimating accurate co-variance in a monocular setting. In [13] photometric co-variance is considered with state co-variance propagated over the tangent manifold. Nonetheless, this technique relies on Gaussian noise models, which we relax in our current work. Our approach also goes beyond [1] that shows navigation with a camera but does not consider the underlying uncertainty during planning.

Only recently, the notion of CCO was extended to settings where the world is described through a voxel-grid representation of the world derived from point clouds [27]. Our work *UrbanFly* [26] extends this line of work further to a more challenging setting where the point clouds generated by the perception system are very sparse and noisy.

2.2 Point Cloud based Loop Detection & Closure

2.2.1 Simultaneous localization and Mapping

SLAM is a problem of jointly estimating the map of the environment and the pose of the mobile robot, In this work we focus on LiDAR based SLAM systems. The architecture of SLAM can be divided into two parts namely, the *front-end* and *back-end*. The *front-end*, of SLAM is mainly concerned with the task

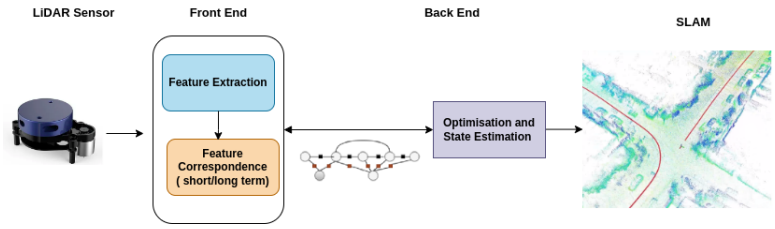


Figure 2.1: The architecture of a typical SLAM system. The backend can further provide supportive information to the front end to help it detect loop candidates.

of sensor data collection and creation of graphical models. The back-end leverages on the abstractions created by the front end and performs inference for state estimation. An overview of SLAM architecture is given in Fig. 2.1.

The front-end of SLAM extracts features and handles both short and long-term correspondences. It matches 2D/3D features between consecutive keyframes for short-term correspondence, and handles loop detection and closure for long-term correspondence. SLAM has been popularly modelled as a *Maximum A Posteriori* estimation problem and factor graph inference techniques are used to perform inference over the graphs. Factor graph optimization for sate estimation has been extensively studied, I refer the reader to [6] [24] for a detailed review.

Loop Detection and Closure (LDC) is central to SLAM that enables a robot to recognize a place which it had previously visited (loop detection/place recognition) and localize (loop closure) itself within a built map, without LDC SLAM would reduce to pure odometry, furtheremore, *loop closures* enable to the robot to better understand the topology of the environment in which it operates.

2.2.2 LiDAR sensor

Light Detection and Ranging (LiDAR) is a popular perception modality present in the autonomous navigation stack of cars and drones. It uses light to measure distances to objects in the environment, laser beams are transmitted by the sensor and are reflected by the objects the *time of flight* between the transmission and reception of beams is measured and the distance to the object is obtained given the time and speed of light. The LiDAR sensor continuously spins emitting beams in all directions and creating a point cloud of the cars surrounding. Point clouds obtained from LiDAR is a geometric set of points, which is essentially a collection of 3D points in *Euclidean Space*. Point clouds are typically represented by their (x, y, z) coordinates in space, and operations on these point clouds are expected to be invariant to the permutation of the members in the set.

2.2.3 Loop Detection and Closure using LiDAR sensors

The SLAM literature has not extensively explored LiDAR based LDC as compared to its image based counterpart.

Handcrafted Feature Descriptors: [35, 43, 70] rely on handcrafted feature descriptors to extract local geometric information and aggregate it to obtain a global descriptor suitable for loop detection and closure. [35, 43] assume presence of a dominant ground plane to detect and close loops. [35] follows a polar representation, where the ground is discretized into bins by splitting in both radial and azimuthal directions, and each bin storing the maximum height present in the vertical volume. [43] discretizes the ground plane into rectangular cells in a Cartesian form, and each cell storing it's point cloud density. Our representation of DEM is a discrete Cartesian representation of the ground plane, where each grid cell stores the maximum height of the points present in it. However, unlike [43, 35], we perform a complete 6-DOF estimation and loop closure.

Learning Based Approaches for Loop Detection: PointNet [55] proposes a neural network model that directly consumes point clouds while maintaining permutation invariance. *PointNetVLAD* [68] uses [55] and NetVLAD[2] to generate global descriptors for place recognition. *PCAN* [79] uses [55] as the backbone architecture to extract local features and the corresponding attention maps along with [2] for feature aggregation. However, both [68, 79] uses PointNet as a backbone architecture, which processes each point separately via a MLP, not capturing local neighbourhood information. Recently, Retriever[72] detects loops directly in the compressed feature representation using Perceiver [32] based mechanism to aggregate the local features. All the above methods use aggregated local features, to compute a global descriptor that is viewpoint invariant. We propose a canonicalization procedure in order to explicitly enforce viewpoint invariance. Furthermore, in contrast to [68, 79, 72] which only perform loop detection, our method performs LDC.

Learning Based Approaches for LDC: LCDNet [8] proposes an end-to-end trainable system, with a *Unbalanced Optimal Transport* algorithm to estimate 6-DOF relative transform between two point clouds. DH3D [15] aggregates local features using hierarchical network to obtain global features for loop detection. Both [8, 15] rely on an expensive 6-DOF data augmentation of the input point cloud in order to achieve orientation invariance. The proposed framework bypasses data augmentation through explicit roll-pich canonicalization followed by yaw alignment. Unlike [8, 15], we operate on highly compressed point cloud representation, making our approach suitable for data transmission in a multi-agent setting. Overlap-based approaches such as OverlapNet [9] and OverlapTransformer [44] are trained using explicit overlap information on range images [4]. OREOS [61] proposes two separate branches: one for loop closure and other for loop detection. Unlike [9, 44, 61] that only estimate the relative yaw between the input point clouds, we estimate the full 6-DOF relative pose.

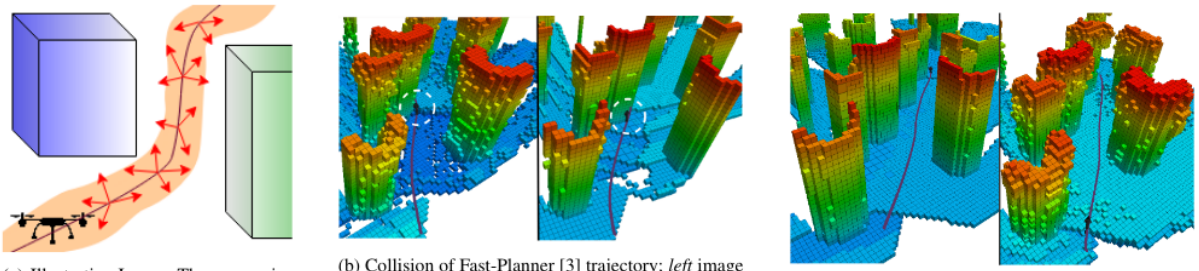
Chapter 3

CCO-VOXEL: Chance Constrained Optimization over Uncertain Voxel-Grid Representation for Safe Trajectory Planning

Many aerial navigation systems use compute trajectories over Octomap [29], or Euclidean Signed-distance field (ESDF) [51] derived from a three-dimensional voxel grid representation of the world. These approaches assume accurate sensor readings during the process of map creation. However, sensor measurements (e.g., point clouds) are noisy and non-parametric in nature. These noisy measurements get translated to incorrect estimates of the distance to the closest obstacle and gradient field. Consequently, deterministic planners [82, 83, 67, 81] who rely on these two pieces of information being accurate are expected to lose their performance and safety guarantees under perception noise. For example, noisy gradients coupled with an incorrect estimate of the distance to the nearest obstacle can result in the UAV moving towards the obstacle and colliding. This work proposes a chance-constrained optimization (CCO) over VOXEL grids as an antidote to this duress. CCO-VOXEL acknowledges that the distance to the closest occupied voxel is noisy and consequently takes this uncertainty into account while computing optimal trajectory plans. As a result, CCO-VOXEL avoids collisions where gradient-based deterministic planner fails (Fig. 3.1).

Previous methods leveraging CCO for handling sensor noise had to rely on the assumption that the underlying random variables that constitute the chance constraint function are parametric or Gaussian [3, 22, 21, 85]. However, as seen in Fig. (3.2) the distribution over the nearest voxel occupancy is non-parametric. Moreover, existing works on CCO such as [85] assume that the obstacles have a pre-defined shape which is crucial for obtaining tractable reformulations of the chance constraints. In this work, we relax both the assumptions of parametric uncertainty and the analytical collision model. Yet, we maintain real-time performance in both simulation and hardware experiments with on-board computation. The core innovations behind our approach are summarized below.

Algorithmic Contribution: For the first time, we present a CCO-based probabilistically safe trajectory planning that is agnostic to both the nature of the underlying uncertainty and the obstacle geometry. Inspired by our prior works [54, 23], we use the notion of distribution embedding in Reproducing Kernel Hilbert Space (RKHS) and Maximum Mean Discrepancy (MMD) to reformulate CCO as a distribution matching problem. However, we overcome one of the key limitations of [54, 23]: estimating the so-



(a) Illustrative Image: The arrows indicate the direction of gradients and the patch around represents the distribution in the collision distance

(b) Collision of Fast-Planner [3] trajectory; *left* image is the collision in the noisy map and the *right* is for visualization purpose that indicates the path has collided with an actual obstacle (noiseless map)

(c) CCO-VOXEL implementation which avoids the obstacle, in presence of noise (*right*) and its visualization in a noiseless map (*left*)

Figure 3.1: **Overview**: Accurate gradient and distance to closest obstacles cannot be extracted from uncertain voxel grid representation built from noisy sensors. Fig.(a) illustrates this effect, wherein the gradient at a given point can be in any one of the directions shown by arrows. Similarly, the distribution of distance is shown as a confidence interval band. The noisy gradients and distance estimation tend to guide the trajectory towards the obstacle leading to a collision as shown in (b) (the collision is highlighted by a white circle) . We propose CCO-VOXEL, a method that accounts for the uncertainty in the distance measurements and obtains an optimal solution through a gradient free *Cross Entropy Minimization* technique resulting in a safe and optimal trajectories as shown in (c).

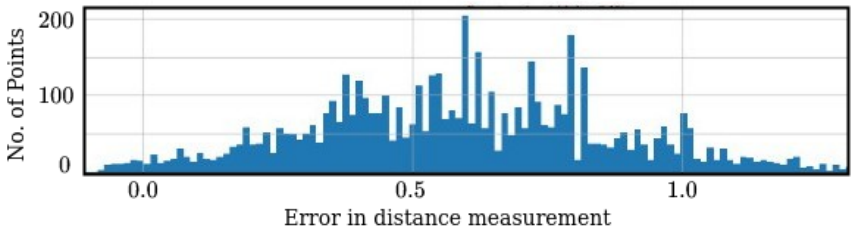


Figure 3.2: The error (difference between the true collision distance and the measured value) distribution histogram that is obtained from a noisy voxel grid. The OptiTrack Motion Capture system was used to determine the ground truth distance. Octomap with Euclidean Distance Transform (dynamicEDT3D) is used for collision distance measurements.

called desired (or target) distribution for computing the distribution matching cost. We show that by embedding the distribution of constraint violation in RKHS instead of collision avoidance constraint themselves, as done in [54, 23], we can by-pass the need to estimate the desired distribution. Furthermore, distribution of constraint violation induces efficient structure in the algebraic form for MMD, allowing for pre-computation and caching of most computationally expensive parts. We also improve the computational performance of our approach by projecting the samples drawn from the distribution of constraint violations into some latent dimension before computing the MMD. We learn this projection through auto-encoder-based supervised learning. Finally, we minimize a combination of MMD and kinodynamic costs through the Cross-Entropy Method (CEM) to compute a smooth and probabilistically safe trajectory.

State-of-the-art Performance: We show tangible performance gain over deterministic gradient planners [82] and methods that handle uncertainty by growing the obstacle map by the co-variance of the

distribution [12, 65]. The benefits are seen in the form of almost negligible collisions under the duress of varying levels of sensor noise even as the previous methods reports high collision rates.

3.1 Problem Formulation

Assumptions: We assume that the robot has precise motion capabilities but noisy perception. As a result, the built ESDF has imprecise information about the gradient field and distance to the closest obstacle. The distribution over the distance has the form $d = d_{measured} + \epsilon_i$, where ϵ_i are samples drawn from some black-box distribution whose parametric form is not known.

Trajectory Parametrization: We parametrize quadrotor trajectory in the following manner

$$\begin{bmatrix} x(t_1) \\ x(t_2) \\ \dots \\ x(t_n) \end{bmatrix} = \mathbf{P}\mathbf{c}_x, \begin{bmatrix} \dot{x}(t_1) \\ \dot{x}(t_2) \\ \dots \\ \dot{x}(t_n) \end{bmatrix} = \dot{\mathbf{P}}\mathbf{c}_x, \begin{bmatrix} \ddot{x}(t_1) \\ \ddot{x}(t_2) \\ \dots \\ \ddot{x}(t_n) \end{bmatrix} = \ddot{\mathbf{P}}\mathbf{c}_x. \tag{3.1}$$

where, $\mathbf{P}, \dot{\mathbf{P}}, \ddot{\mathbf{P}}$ are matrices formed with time-dependent basis functions (e.g polynomials) and $\mathbf{c}_x$ are the coefficients associated with the basis functions. Similar expressions can be written for $y(t), z(t)$ as well in terms of coefficients $\mathbf{c}_y, \mathbf{c}_z$, respectively.

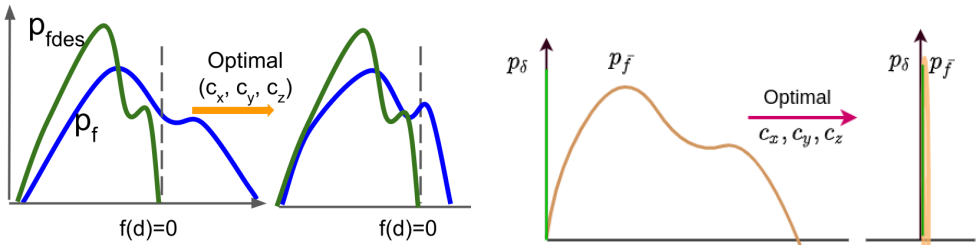
Safe Motion Planning: Using the parametrization discussed above, we formulate our probabilistically safe motion planning as the following optimization problem, wherein $P(\cdot)$ represents probability.

$$\min l(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z) \tag{3.2a}$$

$$P(f(d) \leq 0) \geq \eta, \forall t \tag{3.2b}$$

$$f(d) = r_{safe} - d(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z, t) \tag{3.3}$$

Our cost function (3.2a) encourages smoothness in the trajectory by mapping the trajectory coefficients to penalties on higher-order motion derivatives such as jerk and penalising the violation of acceleration and velocity limit. The algebraic form is similar to that used in [82]. The function $d(\cdot)$ in (3.3) maps the coefficients to distance to the closest obstacle at a given time t. Consequently, the inequality (3.2b) ensures that the probability of the closest obstacle distance being less than safe value r_{safe} is below some specified threshold η. Constraints of the form (3.2b) are known as chance constraints [7]. Generally, chance-constrained optimization (CCO) are considered computationally intractable. Thus, existing works focuses on deriving tractable approximations of (3.2b) under two predominant assumption. Firstly, the obstacles are assumed to have defined geometric shapes (e.g ellipsoid) and as a result it becomes possible to derive an analytical expression for $d(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z, t)$. Second, the perception uncertainty are often assumed to be Gaussian. When both the assumptions are made simultaneously it even becomes possible to formulate a convex approximation of (3.2b) [85], [5].



(a) The setting from [54] where the goal is to bring the distribution p_f to the left of $f(d) = 0$. This is achieved by constructing a desired distribution $p_{f_{des}}$ and minimizing the distance of p_f from it in RKHS. Importantly, estimation of $p_{f_{des}}$ required solving an optimization problem

(b) We work with the distribution of the collision constraint violation rather than the constraint themselves. As a result, the desired distribution in our case is simply the Dirac Delta distribution p_δ. The distribution $p_{\bar{f}}$ approaches the desired Dirac function p_δ for an optimal $\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z$

Figure 3.3: An intuitive understanding of the RKHS embedding of [54] and our proposed method

Our work relaxes both the above mentioned assumptions and thus substantially expands the applicability of chance-constrained optimization based motion planning in real-world environment. We handle chance-constraints on collision avoidance by treating $d(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z, t)$ as a black-box model that can only be queried from a voxel-grid representation of the world. Secondly, we make no assumption on the underlying perception uncertainty.

Reformulation from [54], [23] and Limitations: Let p_f represent the probability distribution of $f(d)$, i.e $p_f = P(f(d))$. Our prior work [54], [23] imagined CCO as the problem of choosing an appropriate $(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z)$ such that the p_f at any given time t takes on an appropriate shape. This naturally gives rise to the notion of desired distribution $p_{f_{des}}$, i.e a distribution that p_f should resemble as closely as possible (Fig. 3.3 (a)). The entire mass of the $p_{f_{des}}$ lies to the left of $f(d) = 0$. As p_f becomes similar to $p_{f_{des}}$, its mass starts getting shifted to the left and the probability of $P(f(d) \leq 0)$ goes up. This intuition led to the following reformulation of optimization (3.2a)-(3.2b)

$$\min l(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z) + w l_{dist}(p_f, p_{f_{des}}), \tag{3.4}$$

where, l_{dist} is a cost that quantifies the similarity between two distributions. The constant w controls the trade-off between minimizing the distribution similarity (or probability of collision avoidance) with the primary cost. We discuss suitable form for l_{dist} in the next section.

The **main limitation** of [54], [23] is that it requires one to solve an optimization problem to estimate $p_{f_{des}}$. While tractable, in a reactive one-step setting, this estimation process becomes highly challenging for multi-step trajectory planning such as the one considered in this work. In the next section, we introduce our main algorithmic results: a novel RKHS reformulation of the CCO (3.3)-(3.2b) that does not require us to estimate $p_{f_{des}}$.

3.2 Main Algorithmic Results

3.2.1 Distribution over Constraint Violations

Our main idea is to work with the distribution of violations of the collision avoidance constraints rather than the constraints themselves. More precisely, let $\overline{f}$ be the constraint violation function defined as

$$\overline{f} = \max(0, f(d(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z, t))) \tag{3.5}$$

A distribution over d can be mapped to $\overline{f}$ and we represent it as $p_{\overline{f}}$. Although, computing the analytical form for $p_{\overline{f}}$ is intractable, we can make the following remark regarding the best possible shape that it can take in the context of collision avoidance.

Remark 1. *The best possible shape of $p_{\overline{f}}$ is given by a Dirac Delta distribution p_δ . In other words, p_δ acts as the desired distribution for $p_{\overline{f}}$*

Remark 2. *As $p_{\overline{f}}$ becomes more and more similar to p_δ, the probability of collision avoidance goes up.*

Remark 1 and 2 elucidates our motivation behind shifting from the distribution of constraints, as done in [54], [23], to distribution of constraint violations. The desired distribution in our case is always fixed and most importantly, exactly known. Remark 2 forms the basis of the following proposed reformulation of CCO.

$$\min_{\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z} l(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z) + w l_{dist}(p_{\overline{f}}, p_\delta) \tag{3.6}$$

Note that $p_{\overline{f}}$ is in fact a function of d which in turn is a function of the trajectory coefficients $(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z)$

3.2.2 Maximum Mean Discrepancy and Sample Approximation

A common method to quantify similarity between two distributions is through Kullback Liebler Divergence (KLD). However, it requires us to know the analytical form for the distributions under consideration and thus is not suitable to quantify the similarity between $p_{\overline{f}}$ and p_δ. One possible workaround is provided by the notion of Maximum Mean Discrepancy (MMD) that quantifies the similarity of two distributions in Reproducing Kernel Hilbert Space (RKHS). Importantly, both the embedding into RKHS and MMD can be obtained by just sample level information of distributions. To this end, let $\mu_{p_{\overline{f}}}$ and μ_{p_δ} represent the RKHS embedding of $p_{\overline{f}}$ and p_δ computed in the following manner.

$$\mu_{p_{\overline{f}}} = \sum_{i=0}^{N} \alpha_i k(\overline{f}(d_i), \cdot) \tag{3.7}$$

$$\mu_{p_\delta} = \sum_{i=0}^{N} \beta_i k(0, \cdot) \tag{3.8}$$

where, d_i refers to the i^{th} sample drawn from the distribution of d for a given $(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z, t)$. Please note that (3.8) follows from the fact the samples from a Dirac Delta distribution are all zeroes. Using (3.7) and (3.8), we can formulate l_{dist} in the following manner.

$$l_{dist} = \left\| \mu_{p_{\overline{f}}} - \mu_{p_\delta} \right\|_2^2 \tag{3.9}$$

It is worth reiterating that MMD (3.9) is a function of $(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z, t)$. Each possible choice leads to a different distribution of d and consequently MMD. Thus the goal is to come up with the right trajectory coefficients which minimizes both MMD and the primary cost function. We achieved this through Cross-Entropy Method [48] [53] and is discussed in Sec. 3.2.4.

3.2.3 Pre-Computation and Improving Sample Complexity

Computing MMD through Kernel Matrices: Let $d_0, d_1, d_2, \ldots, d_n$ be the samples drawn from the distribution of the closest obstacle at any given time t for a given $\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z$. The distance samples can then be mapped to samples from $p_{\overline{f}}$ through (3.5). In this part, we derive the matrix representation of the MMD (3.8) to show how various parts of it can be pre-computed and cached. To this end, we expand (3.9) in the following manner.

$$\begin{aligned}\|\mu_{p_{\overline{f}}} - \mu_{p_\delta}\|^2 = &\langle \mu_{p_{\overline{f}}}(d), \mu_{p_{\overline{f}}}(d) \rangle - 2\langle \mu_{p_{\overline{f}}}, \mu_{p_\delta} \rangle + \\ &\langle \mu_{p_\delta}, \mu_{p_\delta} \rangle \end{aligned} \tag{3.10}$$

Substituting kernel mean functions (3.7) and (3.8) in (3.10)

$$\langle \mu_{p_{\overline{f}}}(d), \mu_{p_{\overline{f}}}(d) \rangle = \langle \sum_{i=0}^{N} \alpha_i k(\overline{f}(d_i), \cdot), \sum_{j=0}^{N} \alpha_j k(\overline{f}(d_j), \cdot) \rangle \tag{3.11a}$$

$$\langle \mu_{p_{\overline{f}}}(d), \mu_{p_\delta} \rangle = \langle \sum_{i=0}^{N} \alpha_i k(\overline{f}(d_i), \cdot), \sum_{j=0}^{N} \beta_j k(0, \cdot) \rangle \tag{3.11b}$$

$$\langle \mu_{p_\delta}, \mu_{p_\delta} \rangle = \langle \sum_{i=0}^{N} \beta_i k(0, \cdot), \sum_{j=0}^{N} \beta_j k(0, \cdot) \rangle \tag{3.11c}$$

Using the kernel trick Eq. (3.11) can be reduced to a matrix form and can be expressed as

$$\|\mu_{p_{\overline{f}}} - \mu_{p_\delta}\|^2 = \mathbf{C}_\alpha \mathbf{K}_{\overline{\mathbf{f}}\overline{\mathbf{f}}} \mathbf{C}_\alpha^{\mathbf{T}} - \mathbf{2}\mathbf{C}_\alpha \mathbf{K}_{\overline{\mathbf{f}}\mathbf{0}} \mathbf{C}_\beta^{\mathbf{T}} + \mathbf{C}_\beta \mathbf{K}_{\mathbf{00}} \mathbf{C}_\beta^{\mathbf{T}} \tag{3.12}$$

where C_α and C_β are the weight vectors given by

$$C_\alpha = \Big[\alpha_0, \alpha_1, ..., \alpha_n\Big]; C_\beta = \Big[\beta_0, \beta_1, ..., \beta_n\Big]$$

and $K_{\overline{f}\overline{f}}$, $K_{\overline{f}\delta}$ and $K_{\delta\delta}$ are matrices defined as

$$\mathbf{K}_{\overline{\mathbf{f}}\overline{\mathbf{f}}} = \begin{bmatrix} k(\overline{f}(d_0), \overline{f}(d_0)) & \ldots & k(\overline{f}(d_0), \overline{f}(d_n)) \\ k(\overline{f}(d_1), \overline{f}(d_0)) & \ldots & k(\overline{f}(d_1), \overline{f}(d_n)) \\ \vdots & \ddots & \vdots \\ k(\overline{f}(d_n), \overline{f}(d_0)) & \ldots & k(\overline{f}(d_n), \overline{f}(d_n)) \end{bmatrix}$$

$$\mathbf{K}_{\mathbf{f}\delta} = \begin{bmatrix} k(\overline{f}(d_0), 0) & \dots & k(\overline{f}(d_0), 0) \\ k(\overline{f}(d_1), 0) & \dots & k(\overline{f}(d_1), 0) \\ \vdots & \ddots & \vdots \\ k(\overline{f}(d_n), 0) & \dots & k(\overline{f}(d_n), 0) \end{bmatrix}$$

$$\mathbf{K}_{\delta\delta} = \mathbf{1}_{n \times n}$$

We use Radial Basis Functions (RBF) during implementation. The use of the Dirac-Delta function as the desired function leverages us with two key properties which would further decrease the computation time.

1. Pre computing matrices: The matrix $K_{\delta\delta}$ is essentially a set of ones and it does not need explicit computation at run time. Furthermore, if a polynomial kernel is used the matrix $K_{\overline{f}\delta}$ would also reduce to set of ones and only one matrix needs computation at run time.

2. Symmetric matrix: It can be observed the kernel matrices are square and symmetric, this implies that we can compute values only for one triangle of the matrix and copy the values into the other triangle.

The use of these techniques in addition to feature space embedding, and vectorized matrix-algebra enables the computation of MMD in real-time.

Feature space embedding of distance measurements: We need a total of n samples of $\overline{f}$ to compute its embedding in RKHS and consequently MMD. A large n (≈ 100) gives a good estimate of MMD but at the same time increases the computational burden (the size of Kernel matrix in (3.12) increases). In this part, we use an auto-encoder to improve the sample and computation complexity. More concretely, we stack the n samples of $\overline{f}$ at a given point on the trajectory and project it into a low dimensional latent space. Our auto-encoder based embedding can be seen as the more sophisticated version of reduced-set method used in [54].

We learn the low-dimensional mapping through supervised learning. Formally, we seek to determine an encoder W_1 and decoder W_2 pair such that they perform an identity operation. The training is performed by optimizing the loss function given in (3.13) using *Stochastic Gradient Descent*.The training data consists of n sets of $\overline{f}$, each with m samples, stacked to form the data matrix $\mathbf{D}$ of dimension $\boldsymbol{R}^{\boldsymbol{n \times m}}$.

$$\min_{\mathbf{W}_1, \mathbf{W}_2} \|\mathbf{D}\mathbf{W}_1\mathbf{W}_2 - \mathbf{D}\|^2 \tag{3.13}$$

The final latent space dimension where MMD would be computed is given by $\mathbf{D}\mathbf{W}_1$

3.2.4 Trajectory Optimization

We minimize (3.6) in two steps. At the first step, we use graph-search technique to find a trajectory that approximately minimizes (3.6). This resulting trajectory is used to initialize the Cross Entropy Method (CEM) optimizer for computing an optimal solution of (3.6).

Initial Trajectory Search: Our initial trajectory search builds on Kinodynamic A* proposed in [82] but with a crucial difference that we use MMD as a part of the *Edge Cost* that connects two nodes of the graph. The typical process is as follows. Several motion primitives are generated by sampling different control inputs $\mathbf{u}_t$ (acceleration). These primitives are then assigned an edge-cost (3.14), wherein ΔM (3.15) is the difference between MMD at the current, and the next node and τ is the time for which the control input is applied. The constant ρ trades-off control costs with time. Apart from the edge cost, there is also a heuristic cost that measures the goal-reaching aspect of each primitive. The primitive with the best edge and heuristic cost combination is chosen for expansion in the next iteration.

In contrast to our approach, [82] uses the distance to the closest obstacle (or simply occupancy in an inflated map) to filter out unsafe primitives while the edge-cost only has the first two terms from (3.14). As mentioned earlier, if this distance information is noisy, the A* process itself can lead to unsafe trajectories (see Table 3.3).

$$e_c = (\|u_t\|^2 + \rho)\tau + \Delta M \qquad (3.14) \qquad\qquad \Delta M = M_{i+1} - M_i \qquad (3.15)$$

CEM Based Refinement: The trajectory produced by the graph search can be sub-optimal and might have a higher arc length. We use Cross Entropy Minimization (CEM) to further refine the trajectory and improve its smoothness while maintaining clearance from the obstacle. The various steps of CEM are summarized in Algorithm 1. The input to the CEM is the sub-optimal trajectory obtained from graph-search. The initial trajectory is converted into an initial estimate for trajectory coefficients $(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z)$ through simple curve-fitting (line 4). The first step in Algorithm 1 is drawing N samples of trajectory coefficients, leading to N possible trajectories (line 8). At each point of every sampled trajectory, we computed the cost function (3.6) (line 10). Note that as shown section 3.2.3, the MMD evaluation itself requires drawing n samples from the collision distance distribution at each point of all the sampled N trajectories. We select the top q trajectory coefficient samples (line 13) that led to lowest cost and use them to compute the mean and variance for the next iteration of CEM (line 15). The output of the CEM is the coefficients of the optimal trajectory.

In a standard CEM implementation after every iteration of the inner loop (line 8 to 13) all the elite samples are discarded. This decreases the efficiency and increases the convergence time. Authors in [53] propose *CEM with memory* where a small subset of the elite set is stored and added to the pool of samples for the next iteration. Our CEM uses this insight in line 14 of Algorithm 1.

Algorithm 1: CEM based Trajectory Refinement

Input: Sub optimal trajectory $(x(t), y(t), z(t))$ from initial path search and trajectory execution time t_{exe}
N_{CEM} = Number of CEM interations
$\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z$ = Trajectory coefficients
$\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z \leftarrow$ **FitPolynomial**(x(t), y(t), z(t))
$\boldsymbol{\mu}_0 = (\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z)$
for $i = 0, i \leq N_{CEM}, i++$ **do**
    Initialize $CostList$ = []
    $Pool$ = N samples from $\mathcal{N}(\mu_i, diag(\sigma_i^2))$
    for $j = 0, j \leq size(Pool), j++$ **do**
        $cost \leftarrow$ **DetermineCost**(S_j)
        append $cost$ to $CostList$
    end
    $EliteSet \leftarrow$ **SelectElites**$(CostList, Pool)$
    $Pool \leftarrow$ **UpdateSamples**$(EliteSet)$
    $(\mu_{i+1}, \sigma_i) \leftarrow$ fit Gaussian distribution $EliteSet$
    $\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z = \boldsymbol{\mu}_i$
end
Optimized Trajectory $\leftarrow \mathbf{Pc_x}, \mathbf{Pc}_y, \mathbf{Pc}_z$

3.3 Experiments and Validation

In this section, we demonstrate the advantages of CCO-VOXEL over deterministic methods and other conservative approaches to handle uncertainty through qualitative and quantitative comparisons. We further show that with the convergence of CEM to an optimal solution the collision constraint violation distribution will also converge to the dirac delta function. Ablations confirm the real-time competency of CCO-VOXEL, while the actual experimental runs indicate its robustness and efficacy for on-board implementations.

Benchmark Comparisons: We benchmark our CCO-VOXEL against fast planner [82] and bounding volume [12, 65] algorithms. The comparison is done on two highly cluttered simulation environments named *Box Cylinder world* and *Wall Grid World* (Fig. 3.7) of dimensions $30 \times 30 \times 7$ and $7 \times 30 \times 7$ respectively. Over 200 trials were conducted and the results presented in Table 3.1 are the cumulative statistics of these trials in both the testing environments. Maximum velocity and acceleration were set to $2m/s$ and $3m/s^2$ respectively.

We sample 50 random trajectories at every iteration of CEM, CCO-VOXEL is implemented in C++11, and used the PX4 SITL (Software in the Loop) for simulations. During simulation, a Gaussian noise of zero mean and $0.2m$ variance is added into the point cloud for testing purposes. **Note** that although the point cloud noise is Gaussian, its mapping to the distribution of the closest obstacle will be non-parametric.

The Fast-Planner [82] is a state-of-the-art algorithm known to generate optimal and safe trajectories. However, in a cluttered and noisy environment the distance to obstacle measurements are no longer reliable. Furthermore, the gradients are noisy and can push the trajectory towards the obstacle leading

Figure 3.4: The image to the left is the custom built quadrotor used for flight testing. The center image is taken during the flight, the computed and tracked trajectory is shown in the right.

to a collision (Fig. 3.1(a) and 3.1(b)) and a drop in the success rate and increase in the smoothness cost of the trajectory. On other hand, CCO-VOXEL is gradient free, and the MMD formulation takes into account the noise uncertainty in the collision distance (Fig. 3.1(c)). As a result, it achieves a higher success rate and a lower smoothness cost.

A possible approach to account for uncertainty is to construct a bounding volume around the obstacle, which is obtained by increasing the robot's footprint by the size of the covariance of the distance distribution. This technique has been used in several approaches such as [12]. Due to the increase in a safety margin, the bounding volume approaches have a higher accuracy in comparison to the fast planner [82], but they tend to generate conservative trajectories. Furthermore, this method prevents the planner from navigating in narrow spaces where it is difficult to maintain an appropriate safe margin after the covariance-based inflation.

As shown in Table 3.1 the proposed method out performs both the benchmarks by a fair margin. The results of our experiments demonstrate a **33.17%** and **44.94%** decrease in the overall trajectory smoothness cost in comparison to Fast-Planner and bounding volume based approaches respectively. Furthermore, the success rate of CCO-VOXEL is approximately **3** times higher in comparison to the benchmarked approaches. But our method has a slightly higher computation time, due to the relatively high computation requirement of the evaluating the MMD cost in CEM.

Real World Autonomous Flight We have conducted a fully autonomous flight test in unknown and cluttered environment. The video demonstration results can be found here[1]. We used a customized self developed quadrotor platform , which is equipped with an Intel Realsense D415 depth camera [58], a Pixhawk flight controller is used along with the PX4 firmware, all the computation modules which include mapping, state estimation and motion planning is conducted online on a Intel NUC 10 board [50] with Core i7-10710U Processor,1.1 GHz – 4.7 GHz Turbo, 6 core, the system is equipped with a 16GB RAM and 1TB SSD. Fig. 3.4 depicts the quadrotor and images from the hardware flight testing.

[1]https://github.com/sudarshan-s-harithas/CCO-VOXEL/tree/main/Demo

Table 3.1: Benchmark Comparison

Method	Computation Time(s)	Smoothness (m^2/s^5)	Success %
Proposed	mean: 0.0853 std: 0.014	**12.45**	**100**
FastPlanner	mean: 0.012 std: 0.026	18.63	32.37
Bounding Volumes	mean: 0.007 std: 0.001	22.54	36.76

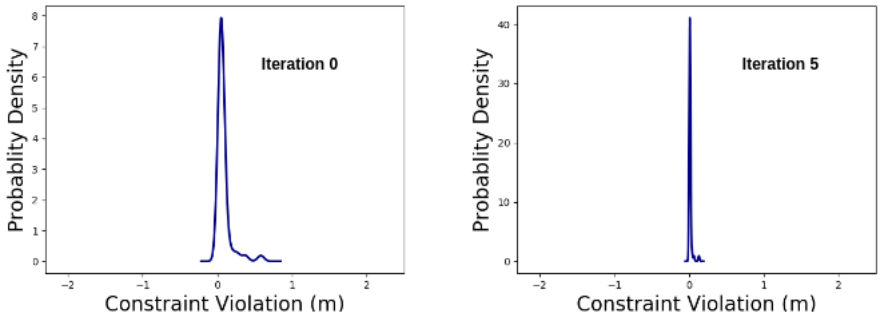


Figure 3.5: With every iteration of our CEM based trajectory optimization we observe that the distribution of the collision-constraint violation approaches the Dirac delta function and therefore the MMD cost is minimized. Note that since plotting is based on approximate kernel density estimation from finite samples, a tiny part of distribution appears to the left of 0 as well.

3.3.1 Convergence analysis

Cross Entropy Minimization: In this section we demonstrate the ability of the CEM to compute the minimum of our unconstrained reformulation of CCO (3.6). The CEM takes the output of the initial trajectory search and further refines it. The convergence of CEM is validated in Fig. 3.6. The figure presents the cost profile of the mean trajectory after every iteration; it can be seen that the optimal solution can be obtained in around four iterations with an average computation time less than $0.015s$. The convergence can be verified by observing the variance updates after every iteration. A monotonic decrease in variance implies that after every iteration it converges to low-cost region and thus it is more useful to sample trajectories close to mean trajectory.

Convergence of the Constraint Violation: We demonstrate that the convergence of CEM (and minimization of MMD) implies a greater match between the distribution of collision constraint violation and the Dirac Delta function. This in turn leads to an increased probability of collision avoidance (recall Remark 4). We observe from Fig. 3.5 that five iterations were enough for the constraint violation dis-

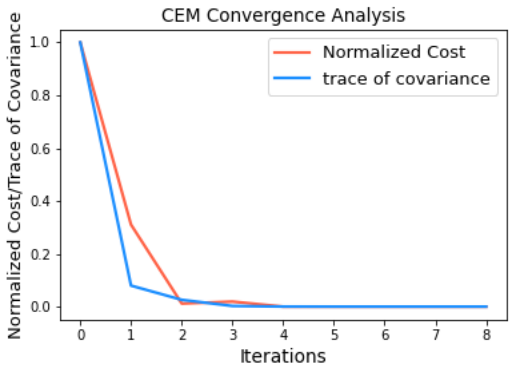


Figure 3.6: CEM Optimization: We observe that the normalized cost and the trace of the covariance has monotonically decreased and the trajectory has converged to an optimal solution

tribution to converge to the Dirac distribution. This convergence results have been empirically verified over a large number of trials.

3.3.2 Ablation Study

To study the impact of each component of our approach, we conducted an ablation study between the possible methods.

MMD Computation: In Sec. 3.2.3 we described the use of an auto-encoder architecture to decrease the sample complexity and consequently time required for an MMD computation. As presented in Table 3.2 we compare this approach against the baseline method where RKHS embedding (Eq. 3.12) are directly computed for a large number of samples without any low dimensional projection. It can be observed that the use of an auto-encoder decreases the computation time by approximately **12** times. Furthermore, we observe from Fig.3.5 that the increase in computation time did not compromise the performance, we were still able to minimize the probability of constraint violations within a very few iterations of CEM.

Table 3.2: Comparison between MMD computation methods

Method	Computation Time(s)
Autoencoder Embedding	**0.00012**
Traditional Method	0.00146

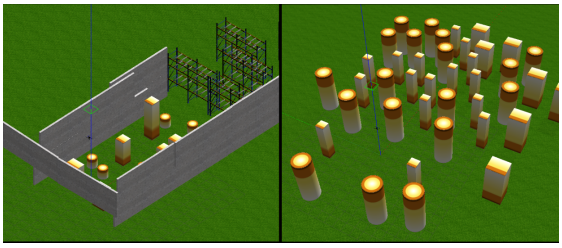


Figure 3.7: Gazebo Environment (a) *Wall Grid World* (b) *Box Cylinder World*

Initial Trajectory Searching Algorithm: Here, we validate the effectiveness of using MMD as the edge-cost in A* based initial trajectory search. To this end, we compare it against the standard kino-dynamic A* used in [82]. As shown in Table 3.3, while our inclusion of MMD as edge cost increases the computation time, it comes up with a massive **2.82** times improvement in success rate. It is worth recalling that our computation time is higher because MMD is computed over the several samples of closest distance to the obstacle drawn from a black-box distribution.

Table 3.3: Initial Trajectory Searching Comparison

Method	Computation Time(s)	Success %
Kinodynamic A* with MMD	0.0798	**100**
Kinodynamic A* [82]	0.017	35.4

3.4 Chapter Summary

Through *CCO-VOXEL* [27] we make a fundamental contribution towards bringing the benefits of CCO-based safe trajectory planning to real-world settings commonly encountered in quadrotor navigation. Our CCO formulation works directly on the voxel grid representation of the world, wherein the collision constraints are not known in analytical form but rather in the form of a black-box query function. Existing approaches that assume known parametric nature in the underlying uncertainty and geometric collision avoidance models are not equipped to work under such minimalist assumptions. We outperformed state-of-the-art deterministic planners in success rate and smoothness metric. In the upcoming chapter, we will continue to take advantage of the benefits provided by our *MMD* and *CEM* formulation to generate trajectories that are both safe and dynamically feasible. However, we will also examine the impact of perception modality on our motion planning abilities, and suggest new perceptual representations specifically designed for quadrotor motion planning.

Chapter 4

UrbanFly: Uncertainty-Aware Planning for Navigation Amongst High-Rises with Monocular Visual-Inertial SLAM maps

As Urban Air Mobility (UAM) becomes more likely in the future, quadrotors will need to navigate through urban areas with tall buildings. Monocular vision, which is a common perception modality for quadrotors due to its lightweight and long-lasting nature, can be used to create maps through a technique called monocular SLAM. However, unlike the `RGBD` cameras as used in [27] these maps can be sparse, noisy and not to metric scale. To achieve accurate scale, we integrate an IMU sensor that provides metric information. When performing SLAM with a sensor setup that includes a monocular camera and an IMU, it's known as *Visual Inertial SLAM (VI-SLAM)* [57]. Despite this, uncertainty and sparsity in the maps remain a problem, making it crucial to account for this uncertainty in trajectory planning as illustrated in Fig. 4.1 (a)(b), it is therefore, crucial to account for the uncertainity during planning. Unfortunately, current algorithms for uncertainty-aware planning are not designed to work with systems that rely on monocular vision-based perception, and often assume that obstacle geometries are known [33, 85] or require depth sensors during operation [82, 67, 27, 83].

We introduce *UrbanFly*, a new planning method that can use monocular vision-based perception while addressing uncertainty. Our approach has two main elements: (i) a representation of the environment that incorporates uncertainty, and (ii) trajectory optimizers that are tailored to this representation. The unique features and advantages of our approach are highlighted below.

Algorithmic Contribution: UrbanFly uses a data-driven neural network to segment the high-rises in the image into their planar constituents. The resulting plane masks along with the point cloud obtained from the triangulation from monocular VI-SLAM are used to build an *Uncertainty Integrated cuboid* `UIC` representation of the obstacles (skyscrapers). We show how the non-parametric point cloud uncertainty can be mapped to the uncertainty in the parameters of the cuboid and finally to the distribution over distance to the closest obstacle. The sample estimates of closest obstacle distance can be coupled with the notion of distribution embedding in Reproducing Kernel Hilbert Space (RKHS) to estimate collision probability. We propose a trajectory optimizer capable of minimizing this estimate along with the conventional smoothness cost. Our optimizer uses the gradient-free cross-entropy method to compute trajectories that explicitly trade-off collision risk and smoothness cost.

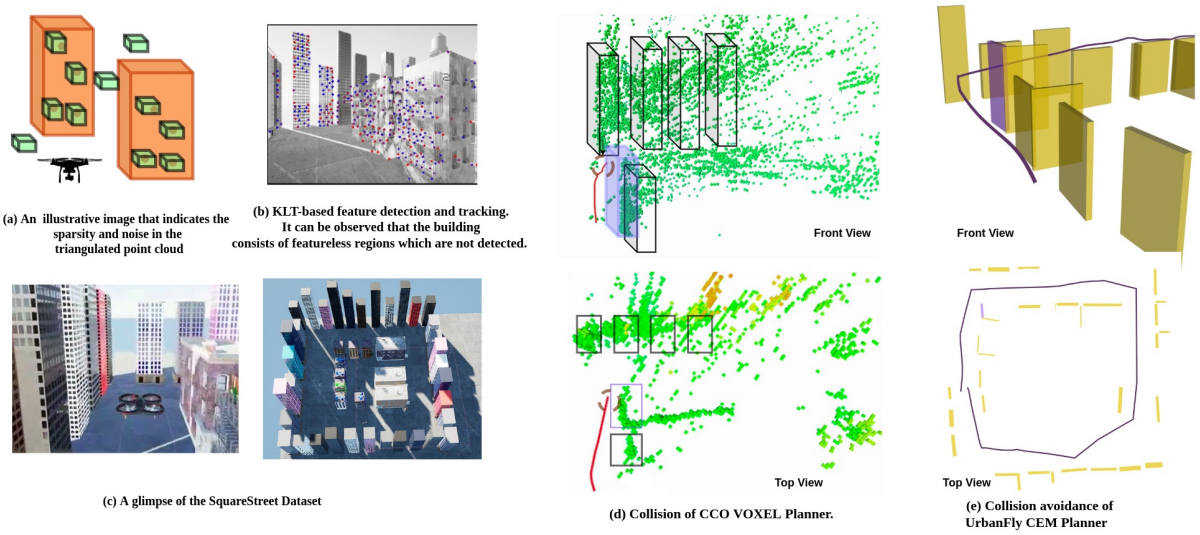


Figure 4.1: **Overview**: Accurate distance measurement to the closest obstacle and associated gradient cannot be expected from a noisy and sparse voxel grid representation built from the triangulated point cloud of a monocular visual-inertial SLAM system (VI-SLAM). This, in turn, poses a critical challenge for trajectory planning. Fig (a) and Fig(b) show that only a few key points that belong to feature-rich regions of the building are detected and tracked by the VI-SLAM resulting in a sparse point cloud (green cuboids in Fig. (a) represent the sparse triangulated point cloud as a discrete voxel). This motivates us to use planes to represent the obstacle instead of the commonly used voxel-grid representation derived from point clouds. Fig. (d) CCO-VOXEL [27], which plans on voxel-grid representation, fails to plan the trajectory using the triangulated point cloud from VI-SLAM [57]. CCO-VOXEL uses Octomap as the voxel grid representation, shown as green cubes. The transparent elongated cuboids are the ground truth location of the buildings (obtained from the simulator), and the building in which CCO-VOXEL has collided is highlighted in purple. Fig. (e) UrbanFly's CEM planner successfully plans a collision-free trajectory while considering the uncertainty-integrated cuboid obstacle representation. Fig. (c) The *SquareStreet* simulation environment that we created to test our planner. Please refer to our †Project page for the simulation results.

Advancements over CCO-VOXEL: Although our previous work [27] addressed trajectory planning in an uncertain distance field, it expected dense point clouds from the `RGBD` camera and could not handle sparse point clouds. In contrast, through *UrbanFly*, we have relaxed this requirement and developed a planner that can handle extremely sparse and noisy point clouds generated through triangulation from *VI-SLAM*. Furthermore, *Voxel-Grids* were the primary world representation in [27], here we demonstrate that given a *VI-SLAM* setup our proposed cuboidal representation is more effective in capturing the world geometry in comparison to the voxel grids. Lastly, we show that the cuboidal representation allows for faster retrieval of distance to collision measurements in comparison to voxel grids.

4.1 Pipeline Overview

Perception Module: Our perception module (detailed in section 4.2) satisfies a two-fold objective:

1. **State estimation:** An on-board monocular VI-SLAM (VINS-Mono) [57] is used to perform state estimation. The state includes the 6DOF pose of the quadrotor and an upto scale 3D key point locations.

2. **Uncertainty aware Mapping**: The sparse and noisy point clouds from monocular VI-SLAM are used to represent the obstacles in the environment as axis-aligned cuboids. The perception module also quantifies the uncertainty in cuboid parameters.

Planning Module: The planning module has a single objective: to perform uncertainty aware trajectory planning by leveraging upon the cuboid representation (`UIC`) of the world obtained from the perception module. Our trajectory optimizer follows the stochastic chance-constrained optimization paradigm and formulates a trade-off between probabilistic safety and smoothness cost. We use the gradient-free CEM to solve the resulting optimization problem. The formulation of our trajectory optimizer has been detailed in section 4.3.

4.2 Monocular Visual-Inertial SLAM based Obstacle Reconstruction

4.2.1 Monocular visual-inertial SLAM

Although a detailed description of monocular visual-inertial SLAM [57] is beyond the scope of this thesis, we briefly describe the steps involved in estimating the odometry from images and IMU.

The system starts with measurement pre-processing, wherein 2D key points are extracted and tracked. Simultaneously, we obtain the relative motion constraints from the IMU through a process popularly known as IMU pre-integration in SLAM literature [18]. Then the initialization procedure provides all necessary values, including pose, velocity, gravity vector, gyroscope bias, and three-dimensional (3D) key points (the point clouds), for bootstrapping the subsequent sliding-window-based nonlinear optimization. Finally, the drift is minimized through a global pose graph optimization. We use the resulting

camera pose to transfer the 3D key points into the world frame that are subsequently used in step 4.2.2 for plane reconstruction.

Assumption

We assume that the buildings are planar and feature-rich. We expect a reliable state estimate from the SLAM but acknowledge the uncertainty in the estimated 3D planes obtained from the sparse triangulated point clouds of the monocular visual-inertial SLAM. We also assume that the VI-SLAM estimate of the gravity axis is sufficiently accurate.

4.2.2 3D plane reconstruction

Notations: In the section 4.2.2.1 and 4.2.2.2, we use the symbols $\mathbf{n}, \mathbf{p}, \mathbf{s}$ to refer to the normal, 2D plane origin and size of the plane respectively. However, the nominal plane estimates are denoted by $\mathbf{n_0}, \mathbf{p_0}, \mathbf{s_0}$. Furthermore, in section 4.2.2.2 where we sample multiple planes, the parameters of the individual samples are represented by the symbols $\mathbf{n_i}, \mathbf{p_i}, \mathbf{s_i}$ (i $\in$ 1,2, .., N).

There are two key steps in the uncertainty aware plane modelling or (`UIC` reconstruction), we first estimate the nominal plane parameters and in the second step we model the `SE(2)` uncertainty using these nominal parameters as an initialization.

There are two key procedural steps to obtain the nominal estimates of the plane. Firstly, we have access to the plane masks obtained through PlaneRCNN [40] and associate each 3D key-point obtained from the triangulation of *VI-SLAM* to be associated with a particular plane if the corresponding 2D feature is within the plane masks. This results in the clustering of the 3D point clouds. After clustering, we run a `RANSAC` algorithm on each cluster to estimate the nominal plane parameters.

The plane has been parameterized through three quantities namely: the plane normal ($\mathbf{n_0} = [n_{0x}, n_{0y}, n_{0z}]$), 2D-plane origin ($\mathbf{p_0} \in R^2$) obtained by projecting the center of the building plane onto the ground plane, and the length and height of the plane which is referred to as the size ($\mathbf{s} = [l, w]$) (where l is length and w is width of the plane).

4.2.2.1 Modelling and Analysis of the SE(2) Uncertainty in 3D planes

The triangulated point cloud from the monocular VI-SLAM is highly noisy and can lead to inaccurate detection of planes. We map the uncertainty present in the triangulated point cloud to the uncertainty in the parameters of the estimated plane. To this end, we assume that building facades in the scene are vertical. This results in only SE(2) (a 3DOF) movement for the building plane viz. 2D planar translation along the ground plane and a 1D yaw rotation along the axis perpendicular to the ground plane.

As explained previously, the plane is modeled through three parameters, the normal ($\mathbf{n}$), plane origin $\mathbf{p}$, and size $\mathbf{s}$. To model the error, we need access to the ground truth parameters of the plane; we set up an experiment in Unreal [16], a high-fidelity gaming engine, to obtain the ground truth parameters of the plane through simulation. To determine the error, a human pilot would fly the quadrotor through

the simulation environment (the environment details are given in section 4.4.2), and the procedure mentioned in the previous section would be performed to obtain the nominal plane estimates. The difference between the observed (the nominal values) and ground truth measurements would be used to model the error. The uncertainty in each of the plane parameters is modeled as follows:

(i) Normal $\mathbf{n}$: The normal determines the orientation of the plane, and from our SE(2) assumption, we notice that this orientation is the 1D yaw (viz., the rotation along the axis perpendicular to the ground plane). Therefore, we parameterize the uncertainty in the normal through the error in the yaw (orientation) denoted by Δ_ψ. Through simulation, we obtain the ground truth orientation of the plane denoted by ψ_{GT}, and we further obtain the nominal plane estimates; we measure the observed orientation given by ψ_0. Where $\psi_0 = \arctan \frac{n_{0y}}{n_{0x}}$ for the observed plane normal $\mathbf{n_0}$. and $\Delta_\psi = \psi_0 - \psi_{GT}$. From Fig. 4.3 (top right) we observe that this error is non-parametric.

(ii) Size $\mathbf{s}$: The size of the plane refers to the length and width of the plane and it is measured in meters. Through the simulator, we obtain the ground truth size of the plane, and the nominal plane parameters are obtained as explained above. The difference between them would result in the error histogram shown in Fig. 4.3 (bottom row). We further observe that the size distribution does not follow the parametric Gaussian form.

(iii) Plane origin $\mathbf{p}$: The error in the plane origin is represented by a vector in R^2. It is obtained by taking the difference between the true plane origin (obtained from simulation) and the observed plane origin. The distribution of plane origin is also non-Gaussian, as shown in Fig. 4.3 (top left) (for ease of illustration, we only show the error in one of the dimensions in R^2, the distribution of error on the other axis is similar).

4.2.2.2 Sampling multiple planes

Assume that we can fit a non-parametric distribution to the plane parameters and sample $\Delta_{\psi_i}, \Delta_{s_i}$ and Δ_{p_i} from that. Note the fitted form need not be even analytically known. We can generate multiple samples of the plane parameters through (4.1). The process of sampling multiple planes by perturbing individual plane parameters and combining them is shown in Fig. 4.2

$$\psi_i = \psi_0 + \Delta_{\psi_i}, \mathbf{s_i} = \mathbf{s_0} + \Delta_{s_i}, \mathbf{p_i} = \mathbf{p_0} + \Delta_{p_i} \tag{4.1}$$

4.2.2.3 Cuboid approximation of the 3D plane segment

The primary objective of modeling the uncertainty in the plane is to exploit its properties in a downstream task such as trajectory planning. Performing collision checks is of key importance in trajectory planning and can be rapidly performed by querying an analytical *Signed Distance Function* (SDF). To take advantage of the geometrical properties of the SDF, we consider each finite plane as a cuboid of infinitesimal thickness.

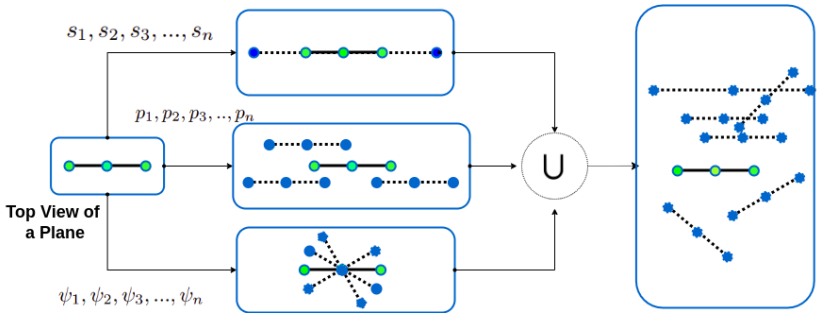


Figure 4.2: In the top view, a plane appears as a line, as shown in the extreme left (the thick line with three green dots indicates the nominal plane estimate). Perturbing each plane parameters individually can distort the plane in different ways, as shown through the middle blocks. The symbol ∪ represents an union of all the random perturbations that results in a total of $n_\psi \times n_s \times n_o$ samples of the plane; an illustration of the sample set is shown on the extreme right.

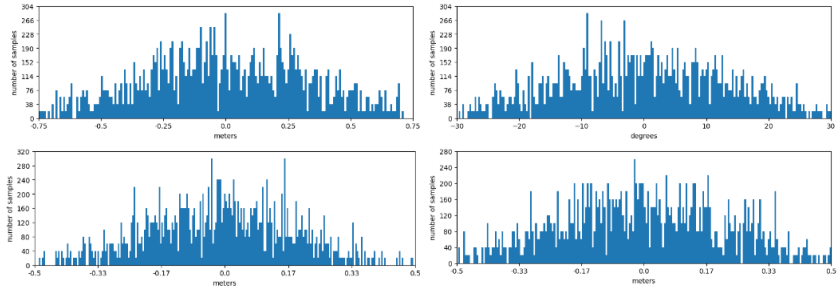


Figure 4.3: Error Histogram for the plane parameters, the yaw error is shown in top-right, the origin error is shown in the top left, and the bottom row refers to the error in size length and width, respectively.

4.3 Uncertainty aware Trajectory Planning

4.3.1 Distance Queries to Cuboid

The distance to the surface can be obtained by querying the analytical SDF as given in (4.2); this allows for rapid collision checks. For a given query point $\mathbf{q} = [x, y, z]$ (measured in world frame), the resulting distance d_{ijk} represents the distance measured to the cube using the i^{th} sample from the yaw angle distribution ψ_i, the j^{th} sample from the size distribution $\mathbf{s_j}$ and k^{th} sample from the origin distribution $\mathbf{p_k}$, the matrix $\mathbf{T_i}$ is a transformation matrix defined by (4.3) that transforms a given query point from the world frame to the local frame of the cuboid, $|.|$ denotes the absolute value function. The $\max$ function would perform the element-wise maximum operation and would return a zero vector if the query point is on or within the cuboid. The $..._2$ would compute the l_2 norm of the resulting vector.

$$d_{ijk} = \max\left(\left(|\mathbf{T}(\psi_\mathbf{i}, \mathbf{p_k})\mathbf{q}| - \mathbf{s_j}\right), \mathbf{0}_{3\times 1}\right)_2 \tag{4.2}$$

$$\mathbf{T_i} = \left[\begin{array}{c|c} \mathbf{R_i^T} & -\mathbf{R_i^T}\mathbf{p}_k \\ \hline \mathbf{0}_{1\times 3} & 1 \end{array}\right] \qquad \mathbf{R_i^T} = \begin{bmatrix} \cos\psi_i & -\sin\psi_i & 0 \\ \sin\psi_i & \cos\psi_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

Since we draw n_ψ, n_s and n_o samples from the black-box error distribution of the yaw, size and origin uncertainties, we get a total of $n_\psi \times n_s \times n_o$ distance measurements for each query point. The process of relating distance queries as collision constrains is described in the next section 4.3.2 .

4.3.2 Surrogate for Collision Probability

Distribution over Constraint Violations: Our vision-based perception module enforces some additional requirements on the planning side. It is not merely enough to maintain a large conservative distance from the obstacles (sky-scrapers) as in that case, we lose the coarse features that are present at the boundaries of the obstacles. On the other hand, flying very close to the obstacles may compromise safety. We thus propose to bound the distance estimates d_{ijk} between r_{min} and r_{max}. To this end, we formulate the following constraint violation function.

$$\overline{f}_{ijk} = \max(d_{ijk} - r_{max}, 0) + \max(r_{min} - d_{ijk}, 0) \tag{4.4}$$

Now, as explained in Section 4.3.1, d_{ijk} are in fact sample approximations of some unknown true estimate of distance $\hat{d}$ to the closest obstacle. Thus, (4.4) maps the distribution over distance estimates to distribution of constraint violation. Moreover, $\overline{f}_{ijk}$ are the sample approximations of the true distribution of $\overline{f}$ which we henceforth call as $p_{\overline{f}}$. Although computing the exact analytical shape of the distribution $p_{\overline{f}}$ is intractable we can make the following remarks about its shape.

Remark

Remark 3. *The best possible shape of $p_{\overline{f}}$ is given by a Dirac Delta distribution p_δ .*

Remark 4. *As $p_{\overline{f}}$ becomes increasingly similar to p_δ, the probability that the quadrotor is at-least r_{min} away from the obstacle an at-most r_{max} away from the obstacle increases.*

Probabilistic Safety : Remarks 3, 4 forms the basis of our notion of probabilistic safety. Instead of directly measuring the probability of collision avoidance; we define the following surrogate.

Definition 1. *Let $l_{dist}(p_{\overline{f}}, p_\delta)$ be a positive semi-definite function that quantifies the similarity between the $p_{\overline{f}}$, and p_δ. That is, l_{dist} decreases as the distribution becomes similar. Then, l_{dist} can act as an estimate of the probability of collision avoidance.*

4.3.3 Maximum Mean Discrepancy as l_{dist}

The Maximum Mean Discrepancy or MMD is a popular approach to quantify the similarity between two distributions in *Reproducing Kernel Hilbert Space* (RKHS) using just the sample level information. Thus, it can be a potential option for l_{dist}. To this end, let $\mu_{p_{\overline{f}}}$ and μ_{p_δ} represent the RKHS embedding of $p_{\overline{f}}$ and p_δ computed in the following manner.

$$\mu_{p_{\overline{f}}} = \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \alpha_i\beta_j\gamma_k k(\overline{f}_{ijk},\cdot) \tag{4.5}$$

$$\mu_{p_\delta} = \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \lambda_i\varphi_j\theta_k k(0,\cdot) \tag{4.6}$$

Where $k(.,.)$ is the kernel function (we use an *Radial Basis Function* (RBF) kernel) , $\overline{f}_{ijk}$ represents the constraint violation function defined in (4.4). The constants α_i, β_j and γ_k are the weights associated with the $\overline{f}_{ijk}$ sample of the constraint violation function. Similarly, λ_i, φ_j and θ_k are the weights associated with the sample of the Dirac-Delta distribution. Typically, these weights can be just set to $\frac{1}{n}$. Alternately, we can adopt a more sophisticated approach based on a reduced set or Auto-Encoder-based dimensionality reduction [27]. Please note that (4.6) follows from the fact that the samples from a Dirac-Delta distribution are all zeros. Using (4.5) and 4.6 we will proceed to derive the following algebraic form for (l_{dist}) as given in (4.7).

$$l_{dist} = \left\| \mu_{p_{\overline{f}}} - \mu_{p_\delta} \right\|_2^2 \tag{4.7}$$

It is worth noting that MMD is a function of the d_{ijk}, which in turn is a function of the query point. In the context of planning, the query points will belong to the trajectory.

4.3.4 Matrix Form of MMD

We can efficiently evaluate l_{dist} through the so-called kernel trick. To show how, we expand (4.7) as follows

$$\|\mu_{p_{\overline{f}}} - \mu_{p_\delta}\|^2 = \langle \mu_{p_{\overline{f}}}, \mu_{p_{\overline{f}}} \rangle - 2\langle \mu_{p_{\overline{f}}}, \mu_{p_\delta} \rangle + \langle \mu_{p_\delta}, \mu_{p_\delta} \rangle \tag{4.8}$$

Substituting the kernel mean functions 4.5 and 4.6 in 4.8

$$\langle \mu_{p_{\overline{f}}}, \mu_{p_{\overline{f}}} \rangle = \langle \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \alpha_i\beta_j\gamma_k k(\overline{f}_{ijk},\cdot), \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \alpha_i\beta_j\gamma_k k(\overline{f}_{ijk},\cdot) \rangle \tag{4.9a}$$

$$\langle \mu_{p_{\overline{f}}}, \mu_{p_\delta} \rangle = \langle \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \alpha_i\beta_j\gamma_k k(\overline{f}_{ijk},\cdot), \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \lambda_p\varphi_j\theta_k k(0,\cdot) \rangle \tag{4.9b}$$

$$\langle \mu_{p_\delta}, \mu_{p_\delta} \rangle = \langle \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \lambda_i\varphi_j\theta_k k(0,\cdot), \sum_{i,j,k=1}^{n_\psi,n_s,n_o} \lambda_i\varphi_j\theta_k k(0,\cdot) \rangle \tag{4.9c}$$

Using the kernel trick, (4.9) can be expressed in matrix form as given below

$$\|\mu_{p_{\overline{f}}} - \mu_{p_\delta}\|^2 = \sum_{k=1}^{n_o} \mathbf{C}_{\alpha\beta/\gamma_\mathbf{k}} \mathbf{K}^k_{\overline{f}\overline{f}} \mathbf{C}^\mathbf{T}_{\alpha\beta/\gamma_\mathbf{k}} -$$

$$2\mathbf{C}_{\alpha\beta/\gamma_\mathbf{k}} \mathbf{K}^k_{\overline{f}\delta} \mathbf{C}^\mathbf{T}_{\lambda\varphi/\theta_\mathbf{k}} + \mathbf{C}_{\lambda\varphi/\theta_\mathbf{k}} \mathbf{K}^k_{\delta\delta} \mathbf{C}^\mathbf{T}_{\lambda\varphi/\theta_\mathbf{k}} \tag{4.10}$$

where $\mathbf{C}_{\alpha\beta/\gamma_\mathbf{k}}$ and $\mathbf{C}_{\lambda\varphi/\theta_\mathbf{k}}$ are the weight vectors

$$C_{\alpha\beta/\gamma_k} = \left[\alpha_0\beta_0\gamma_k, \alpha_0\beta_1\gamma_k, ..., \alpha_n\beta_n\gamma_k\right];$$
$$C_{\lambda\varphi/\theta_k} = \left[\lambda_0\varphi_0\theta_k, \lambda_0\varphi_1\theta_k, ..., \lambda_n\varphi_n\theta_k\right]$$

$$\mathbf{K}^k_{\overline{f}\overline{f}} = \begin{bmatrix} \mathbf{K}^k_{00} & \mathbf{K}^k_{01} & \dots & \mathbf{K}^k_{0n} \\ \mathbf{K}^k_{10} & \mathbf{K}^k_{11} & \dots & \mathbf{K}^k_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}^k_{n0} & \mathbf{K}^k_{n1} & \dots & \mathbf{K}^k_{nn} \end{bmatrix} \qquad \mathbf{K}^k_{\overline{f}\delta} = \begin{bmatrix} \mathbf{K}^k_{\delta 00} & \mathbf{K}^k_{\delta 01} & \dots & \mathbf{K}^k_{\delta 0n} \\ \mathbf{K}^k_{\delta 10} & \mathbf{K}^k_{\delta 11} & \dots & \mathbf{K}^k_{\delta 1n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{K}^k_{\delta n0} & \mathbf{K}^k_{\delta n1} & \dots & \mathbf{K}^k_{\delta nn} \end{bmatrix}$$

$$\mathbf{K}^k_{ab} = \begin{bmatrix} k(\overline{f}_{a0k}, \overline{f}_{b0k}) & \dots & k(\overline{f}_{ank}, \overline{f}_{b0k}) \\ k(\overline{f}_{a0k}, \overline{f}_{b1k}) & \dots & k(\overline{f}_{ank}, \overline{f}_{b1k}) \\ \vdots & \ddots & \vdots \\ k(\overline{f}_{a0k}, \overline{f}_{bnk}) & \dots & k(\overline{f}_{ank}, \overline{f}_{bnk}) \end{bmatrix}$$

$$\mathbf{K}^k_{\delta ab} = \begin{bmatrix} k(\overline{f}_{a0k}, 0) & k(\overline{f}_{a1k}, 0) & \dots & k(\overline{f}_{ank}, 0) \\ k(\overline{f}_{a0k}, 0) & k(\overline{f}_{a1k}, 0) & \dots & k(\overline{f}_{ank}, 0) \\ \vdots & \vdots & \ddots & \vdots \\ k(\overline{f}_{a0k}, 0) & k(\overline{f}_{a1k}, 0) & \dots & k(\overline{f}_{ank}, 0) \end{bmatrix}$$

$$\mathbf{K}^k_{\delta\delta} = \mathbf{1}_{n_\psi n_s \times n_\psi n_s}$$

Furthermore, [27] proposes an effective method to utilize supervised learning to map distance measurements to a low dimension feature space, this was found to increase the computational speed significantly. Here we adopt a similar approach for the computation of MMD.

4.3.5 CEM Planner

4.3.5.1 Trajectory Parameterization

The *CEM Planner* uses a polynomial parameterization of the trajectory, defined as

$$\begin{bmatrix} x(t_1) \\ x(t_2) \\ \dots \\ x(t_n) \end{bmatrix} = \mathbf{P}\mathbf{c}_x, \begin{bmatrix} \dot{x}(t_1) \\ \dot{x}(t_2) \\ \dots \\ \dot{x}(t_n) \end{bmatrix} = \dot{\mathbf{P}}\mathbf{c}_x, \begin{bmatrix} \ddot{x}(t_1) \\ \ddot{x}(t_2) \\ \dots \\ \ddot{x}(t_n) \end{bmatrix} = \ddot{\mathbf{P}}\mathbf{c}_x. \tag{4.11}$$

where, $\mathbf{P}, \dot{\mathbf{P}}, \ddot{\mathbf{P}}$ are matrices formed with time-dependent basis functions ($x(t_i)$) (e.g polynomials) and $\mathbf{c}_x$ are the coefficients associated with the basis functions. Similar expressions can be written for $y(t), z(t)$ in terms of coefficients $\mathbf{c}_y, \mathbf{c}_z$, respectively.

4.3.5.2 Optimization Problem

The CEM-based planner minimizes the following cost function, for a given weight w.

$$\min_{\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z} l(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z) + w l_{dist}(p_{\overline{f}}, p_\delta) \tag{4.12}$$

The function $l(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z)$ maps trajectory coefficients to penalties on higher-order motion derivatives such as jerk. It also penalizes violations of acceleration and velocity limits. The second term in (4.12) is the MMD-based distribution matching cost derived in the previous subsection. Please note l_{dist} as presented in (4.7) is a function of d_{ijk} which in turn is a function of the query point. In the context of trajectory optimization, the query points belong to the planned trajectory and are thus a function of $(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z)$. In other words, different values of trajectory coefficients will map to distribution matching cost l_{dist}.

We solve (4.12) through gradient-free CEM following closely the steps outlined in [27] and [53]. We provide only a brief overview here. We first draw different random values of $(\mathbf{c}_x, \mathbf{c}_y, \mathbf{c}_z)$ from a Gaussian Distribution and evaluate the cost (4.12) on all of these generated samples. We then identify some best-performing samples and then fit a new Gaussian distribution on them for sampling in the subsequent iteration.

4.4 Experiments and Validation

In this section, we demonstrate that voxel-grid representations can be ineffective in a monocular setting due to the inherent sparsity and noise in the triangulated point clouds. Furthermore, we show that *UrbanFly* can generate long-range smooth trajectories and execute them without the failure or drift in VI-SLAM. Finally, we provide an empirical analysis of the convergence of CEM. Simulation videos can be found in the project page `https://github.com/sudarshan-s-harithas/UrbanFly`

4.4.1 Simulation Setup

We used Airsim [63], a state-of-the-art simulation framework which implements a physics engine, flight controller, and photorealistic scene for multirotor. Airsim runs inside Unreal Engine [16] Editor. Our planners are implemented on a computer with 16GB RAM, core i7-10710 CPU. A dedicated simulation server equipped with NVIDIA-1070 graphics card, AMD Ryzen 7 3800x 8-core processor × 16 CPU, and 64GB RAM is used to run the Unreal Engine. The mapping module and CEM planner are programmed in C++.

4.4.2 Simulation Environments

Our methods are validated on the following two datasets:

4.4.2.1 SquareStreet (Synthetic Dataset)

Our approach was facilitated by the use of *Unreal Engine (UE) Editor* [16]. With it, we created a custom street scene in the shape of a square, as shown in Figure 4.1(c). This scene featured 47 buildings over an area of 0.16 square kilometers. The environment is rich in features, making it a suitable option for testing monocular vision-based perception and planning pipelines. Additionally, the environment is modular and can be easily modified to suit specific needs.

4.4.2.2 Yingrenshi (Real city model)

The *UrbanScene3D* [41] dataset offers a unique collection of both real and virtual city models that can be integrated into the Unreal Engine. One of the featured models is the Yingrenshi, as seen in figure 4.4(b) . This virtual replica comprises 252 buildings within an area of 1 square kilometer. It's an replication of actual buildings and streets which serves as a valuable tool for testing our approach in a realistic setting.

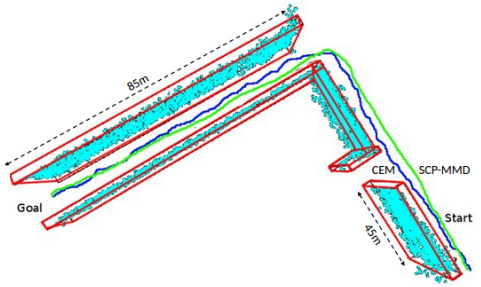
4.4.3 Baseline Comparison and Performance Analysis

We compare the performance of *UrbanFly* with voxel grid based planners [27, 82]. More than 100 trials were conducted in each simulation environment with various start and goal configurations.

CCO-VOXEL [27] is a robust planning framework agnostic to the underlying uncertainty in the noisy distance map. FastPlanner [82] is one of the state-of-the-art motion planning algorithms known to generate optimal and safe trajectories in noiseless environments. However, both [27], and [82] require dense point clouds to generate voxel maps, whereas the point cloud from a monocular VI-SLAM is sparse.

Both the *UrbanFly* optimizers demonstrated superior performance over the competing baselines primarily due to two reasons. Firstly, the VI-SLAM pipeline would not faithfully reconstruct the featureless regions of the buildings, leading to an incorrect perception of the world. Specifically, the planner will falsely consider all the unoccupied regions as free and generate a trajectory through the walls. Secondly, the noisy point clouds would lead to smearing of voxels; this leads to an increase in the clutter level, and the planner would fail, this situation is depicted in Fig. 4.1 (d). On the other hand, our trajectory optimizers work with the polygonal representation of the world while accounting for the uncertainty in its estimation Fig. 4.1(e). Hence it can generate a safe and smooth trajectory.

Table 4.1 compares the performance of our trajectory optimizers with voxel grid-based planners [27, 82], the latter being a deterministic planner that operates under the assumption of zero perception noise. For *SquareStreet* environment, we observed that both our CEM planner was able to achieve more



(a) Demonstration of obstacle reconstruction and trajectory execution of the the proposed planners in *Yingrenshi*

(b)The visualization of the trajectories (top-left) in *Yingrenshi*

Figure 4.4: Qualitative Results: (a) demonstrate the ability of *UrbanFly* to reconstruct planar obstacle boundaries from triangulated point clouds and plan trajectories in real-time. In these figures, CEM trajectories are shown in Blue. In (b), the top-left tile shows the top view of the scene and the executed trajectory of 160m. Other tiles show three different viewpoints along the trajectory while avoiding large buildings.

than **4** times improvement in success-rate over [27, 82]. The improvement in smoothness achieved by both stands at **39.34%** and **42.43%**. A similar trend can be observed for the *Yingrenshi* environment in Table 4.1. *CEM* planners outperform the competing baselines by **3.5** times in terms of success rate and demonstrated an improvement of close to **13%** in terms of smoothness cost.

We also evaluated our planners and the baselines regarding how far they enable the quadrotor to traverse before a failure of VI-SLAM. The results are summarized in the sixth and seventh columns of Table 4.1. Our trajectory optimizer achieved similar performance and were 8 times better than [27, 82].

The computation time (5th column Table 4.1) of our CEM-based trajectory optimizer is comparable with the closest baseline [27] that also performs uncertainty-aware planning. Both approaches used a similar number of distance queries, which were obtained using different methods. While our approach used analytical SDF (4.2), [27] relied on distance queries from a voxel grid representation of the environment. Thus, our approach is expected to scale better if the problem complexity demands an increase in the sample size of CEM. The SCP-MMD planner is faster than [33] while unsuprisingly the deterministic planner [82] has the lowest run-time. The qualitative demonstration of *UrbanFly* in *UrbanScene* is depicted in Fig 4.4 and on *SquareStreet* in Fig. 4.1(e) .

Table 4.1: Benchmark Comparison

Environment	Method	Smoothness	Success	Compute time	Traversed Length	
		(m^2/s^5)	%	time (s)	mean(m)	std(m)
SquareStreet	Ours (CEM)	**8.82**	**86.666**	0.095	52.44	12.37
	Ours (SCP-MMD)	18.63	82.51	0.015	51.35	10.23
	CCO Voxel [27]	14.54	16.66	0.097	6.44	1.37
	FastPlanner [82]	15.32	9.66	0.018	9.44	2.87
Yingrenshi	Ours (CEM)	**10.82**	**83.87**	0.103	150.214	16.304
	Ours (SCP-MMD)	20.85	80.47	0.030	133.82	21.53
	CCO Voxel [27]	12.54	18.42	0.089	46.44	1.37
	FastPlanner [82]	13.54	11.36	0.016	35.44	4.67

4.4.4 CEM Convergence Analysis

Here we empirically validate the ability of CEM to minimize the cost function and obtain an optimal trajectory. The convergence of CEM is shown in Fig. 4.6 where the cost profile of the mean trajectory has converged after 6 iterations. Furthermore, a monotonic decrease in variance implies that with every iteration, the optimizer needs to progressively sample closer to the mean trajectory to find low-cost regions.

Fig. 4.6 depicts the convergence of distribution of constraint violation function (defined in (4.4)) to the Dirac-Delta distribution (the ideal distribution as defined in Remark 3) over the CEM iterations. These convergence results have been empirically verified over large number of trials and validates the observations made in Remark 4.

4.4.5 Ablations

Here we demonstrate a key advantage of the polygonal representation; fast distance queries. The polygonal obstacle representation allows us to query an analytical function (4.2)) to perform rapid collision checking. Fig. 4.5 depicts the results of the querying experiment. Our batch querying formulation provides a significant improvement in querying speed over the traditional *Euclidian Distance Transform* (EDT) (dynamicEDT3D) .

4.5 Chapter Summary

In this chapter through *UrbanFly* [26], we demonstrated that constructing a polygonal representation of the world can significantly help to counter the inherent sparsity and uncertainty of the point clouds generated by the *VI-SLAM* system. We developed a trajectory optimizer that can leverage our world model and obtain fast distance queries which was key in deriving an optimal and smooth trajectory in real-time. Our optimizer showed improvement over several strong baselines in terms of success rate, trajectory smoothness, arc length, etc. In the next chapter, we will be exploring the relationship between scene representation and autonomy further. However, we will be shifting our focus to the state estimation

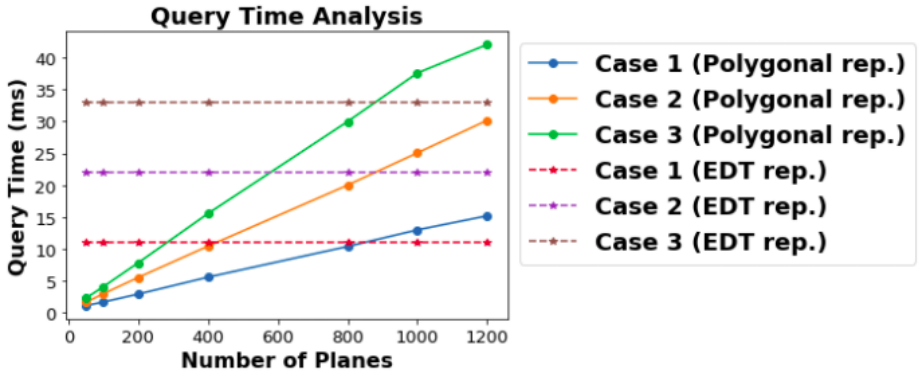


Figure 4.5: The query times recorded for both EDT and cuboid representation for multiple query points and multiple planes. In *Case 1* we test with 50000 query points, similarly for *Case 2* and *Case 3* we record the computation time for 100000 and 150000 query points respectively.

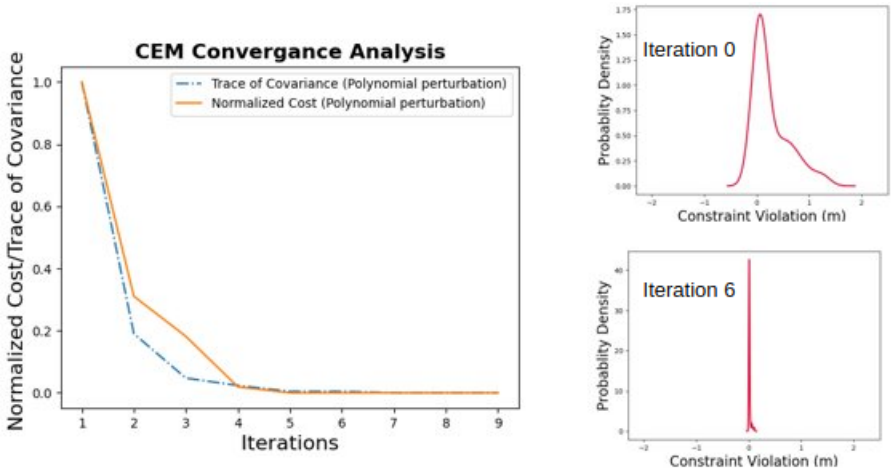


Figure 4.6: Left Figure: The normalized cost decreases and converges as iterations progress. With every iteration, we observe that the distribution of the collision-constraint violation approaches the Dirac delta function and therefore the MMD cost (4.7) is minimized. Note that since plotting is based on approximate kernel density estimation from finite samples, a tiny part of the distribution appears to the left of 0 as well.

module within the navigation stack and would aim to build a robust *Loop Detection and Closure* system for `LiDAR SLAM`.

Chapter 5

FinderNet: A Novel Technique for LiDAR based 6DOF Distributed Loop Detection and Closure

The previous chapters detail the development of motion planning frameworks that are resilient to the inaccuracies within the perceptual representation. Through *FinderNet* we now turn our attention to the state estimation component and develop a robust a *Loop detection and closure* module for LiDAR SLAM (*Simultaneous Localization And Mapping*) pipeline, which is critical to reduce accumulated drift in the estimation process. As single-agent SLAM systems have matured, the research community is increasingly focusing on multi-agent scenarios, and collaborative SLAM [62, 37, 38]. Since, transmitting large point clouds between agents is impractical, judicious use of the limited bandwidth is necessary for distributed Loop Detection and Closure (LDC) [38, 11, 10]. This paper focuses on robust, and data efficient distributed LDC. Though, most components of our pipeline applies to generic point clouds, we assume LiDAR as the essential sensing modality.

LDC techniques are broadly split into two styles: **(1)** those focusing solely on loop detection (and not closure) and generating viewpoint-invariant global descriptors by performing feature aggregation of the local descriptors, and **(2)** those performing joint LDC (both detection and closure) [8, 61, 9]. These methods include an invariant branch whose features are used for loop detection and an equivariant branch whose features are used for loop closure. Typically, both of the above styles rely on massive data augmentation, and apply randomly sampled rigid transforms to the input point clouds to achieve viewpoint invariance. We take an original approach and leverage canonical representations exposing rich spatial structure, and achieve viewpoint invariance. This allows us to train our model without any data augmentation.

The cornerstone of our efforts is a *Roll and Pitch (RP) canonicalizer* and a *Differentiable Yaw Transformer* (DYT). The *RP Canonicalizer* makes use of the dominant ground plane hypothesis (commonly encountered in autonomous driving and drone applications) [42, 35, 9, 61, 43] to compensate for the roll and pitch between two point clouds. The roll and pitch canonicalized point clouds are converted into a *Digital Elevation Map* (DEM), a visual explanation of the process is given in Fig. 5.1. We further develop a *Differentiable Yaw Transformer* (DYT) that operates on the latent feature embeddings of the

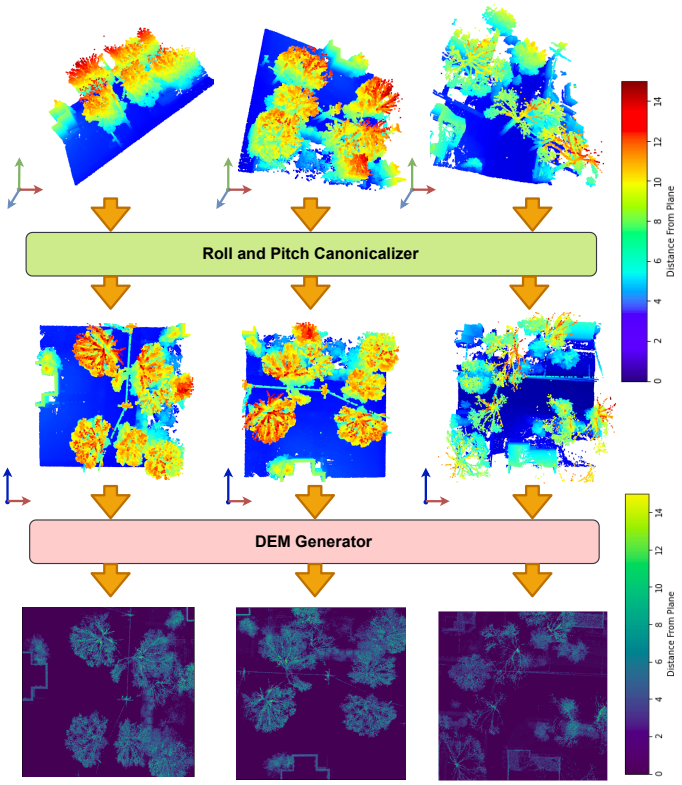


Figure 5.1: We observe that raw LiDAR point clouds (first row) lack spatial structure for robust distributed loop detection and closure (LDC). We perform local roll and pitch canonicalization (second row), followed by discretization along the z-axis (third row), which leads to output similar to digital elevation maps (DEMs) and exposes rich scene structure in the input. Our model performs LDC on such DEMs, leading to high data efficiency, robustness, and generalizability to 6-DOF viewpoint variations.

DEM to achieve yaw invariance, and provide viewpoint invariant loop detection with 6-DOF (SO(3)) relative motion. This is in contrast to existing techniques focusing only on yaw rotation [35, 75, 9, 34].

Contributions: **(1) Novel Pipeline:** Instead of directly operating on the raw point clouds that inherently lack structure, we convert the point clouds into a regularly spaced DEM via a roll and pitch canonicalizer with only the yaw to further deal with. The canonicalized DEM representation provides a structure that CNN backbones can readily process, bypassing equivariance issues that typically plague point cloud representations. While *Pointnet* [55] and its variants [56] handle equivariance, the superiority of the proposed pipeline over *Pointnet* inspired architectures [68, 79, 74] is tabulated in the Results Section over a diverse set of LDC related performance metrics.

(2) Multi Functionality: By enabling LDC on the compressed latent space, we provide for low bandwidth data transfer, which is critical in a multi-agent setting. The querying agent is not required to transmit its entire point cloud, but only its compressed latent embedding. Moreover, the proposed architecture provides for multiple functionalities such as Low Bandwidth Data Efficiency, 6-DOF invariance and LDC unlike prior art.

Method	Venue	DE	VI	NDA	LD	LC
[9]	RSS'20	✗	✗	✗	✓	✓(Yaw)
[35]	IROS'18	✗	✗	NA	✓	✓(Yaw)
[71]	RAL'22	✓	✓	NA	✗	✓
[72]	ICRA'22	✓	✓	✗	✓	✗
[73]	RAL'21	✓	✓	✗	✗	✗
[8]	TRO'22	✗	✗	✗	✓	✓
[61]	IROS'19	✗	✗	✗	✓	✓(Yaw)
[68]	CVPR'18	✗	✓	✗	✓	✗
[79]	CVPR'19	✗	✓	✗	✓	✗
[74]	CVPR'21	✗	✓	✗	✓	✗
[15]	ECCV'20	✗	✗	✗	✓	✓
Ours	****'23	✓	✓	✓	✓	✓

Table 5.1: Comparison with SOTA. Acronyms: **DE:** Data Efficiency though learnt embeddings, **VI:** 6-DOF View Invariance, **NDA:** No large Data Augmentation requirement, **LD:** Loop Detection capability, **LC:** Loop Closure capability, **NA:** Not Applicable.

(3) 6-DOF recovery: Unlike previous approaches that show loop closure only as a SE(2) alignment, the proposed method recovers 6-DOF pose between the two candidate point clouds, even as it precludes the need for data augmentation, exploiting the inherent viewpoint invariance of the pipeline. The proposed framework goes beyond SOTA on a number of public datasets such as KITTI [20], GPR [78] and Oxford RobotCar [45] on established performance metrics for LDC. Specifically, the proposed framework is the best performing on 6-DOF pose recovery and it outperforms most prior art on the SE(2) LDC task. 5.1 gives a conceptual comparison of our method with contemporary techniques.

5.1 Methodology

Our goal is to develop a 6-DOF viewpoint invariant place recognition framework for 3D point clouds for LDC. The overview of our method is shown in 5.2. We first canonicalize the point cloud, and then discretize it to get a DEM representation (5.1.1). We use an *autoencoder* style encoder-decoder network to learn the compressed latent representation for the DEM (5.1.2). The latent representation is transferred between the agents for loop detection and closure, which reduces the data bandwidth requirement

To achieve yaw invariance for loop detection in the compressed space, we have designed a *Differentiable Yaw Transformer* (DYT), it transforms the latent query embedding to rotationally align with the latent embedding of the database sample (5.1.3).The output of the DYT is used in conjunction with the agent dataset for loop detection . Once a loop is detected, the decoders decompress the latent DEM representation and use the decoded DEM from the query and dataset to estimate a 6-DOF relative pose for the loop closure .

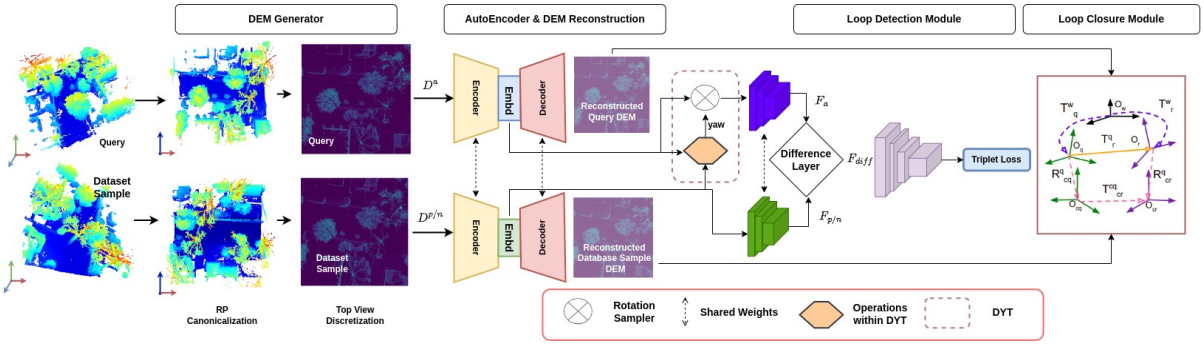


Figure 5.2: The figure demonstrates the overview of our pipeline; the two point clouds in the extreme left are the input query and database sample; the *DEM Generator* (section 5.1.1) generates a discredited top view of the point cloud; and the autoencoder structure further compresses the *DEM* (section 5.1.2). The *Differentiable Yaw Transformer* (DYT) (section 5.1.3) is used for the yaw alignment, the operations within the DYT include *CPC*, *Horizontal padding of polar embedding*, and *Correlation*; each of these are explained in section 5.1.3. The complete set of operations is shown as a single orange hexagon; the result of these operations is a scalar yaw value, which is fed into the rotation sampler. We design a network to perform loop detection (section 5.1.4) and closure (section 5.1.5) using these compressed embeddings without the need for explicit decompression.

5.1.1 DEM Generation

DEMs are digital representations of an input point cloud, capturing the elevation of the terrain or overlaying objects. DEMs have rich representation power, preserving the feature rich regions like edges and corners, and at the same time conserve bandwidth by allowing for aggressive compression and recovery at high quality. Moreover, unlike range images that preserve only yaw [9], DEMs preserve both yaw and planar translation, making them a useful representation for 6-DOF point cloud registration.

Plane Parameterization: Consider an input point cloud $\mathbf{P}_c$ with its corresponding ground plane $\mathbf{r}_c$, and the world ground-plane $\mathbf{r_w}$. We aim to align the planes $\mathbf{r}_c$ and $\mathbf{r}_w$ by estimating the relative roll and pitch (RP) between them. We center the input point cloud ($\mathbf{P}_c$) and extract the ground plane $\mathbf{r}_c$ using RANSAC. The ground plane is parameterized by ($\mathbf{n}_c$, $\mathbf{C}_c$), where $\mathbf{n}_c \in \mathbf{R}^3$ is a unit vector perpendicular to the plane and $\mathbf{C}_c = \{c_i^c \mid i = \{1...n\}\} \in \mathbf{R}^{n\times 3}$ is the set of points $c_i^c \in \mathbf{R}^3$, s.t. c_i^c lies on $\mathbf{r_c}$ and $\|c_i^c\| = 1$. The world ground plane $\mathbf{r}_w$ is parameterized similarly as ($\mathbf{n}_w$, $\mathbf{C}_w$), where $n_w = [0, 0, 1]$ and $\mathbf{C}_w = \{c_i^w \mid i = \{1...n\}\} \in \mathbf{R}^{n\times 3}$ is the set of points $c_i^w \in \mathbf{R}^3$, s.t. c_i^w lies on $\mathbf{r_w}$ and $\|c_i^w\| = 1$. Note that the world ground plane is not estimated through data, instead is a constructed canonical plane of reference. The canonicalization for roll (α), and pitch (β) involves two steps. First we obtain a coarse estimate of (α) and (β) by aligning the normals $\mathbf{n}_c$ and $\mathbf{n}_w$. Post that, we do a finer estimate through Iterative Closest Point (ICP).

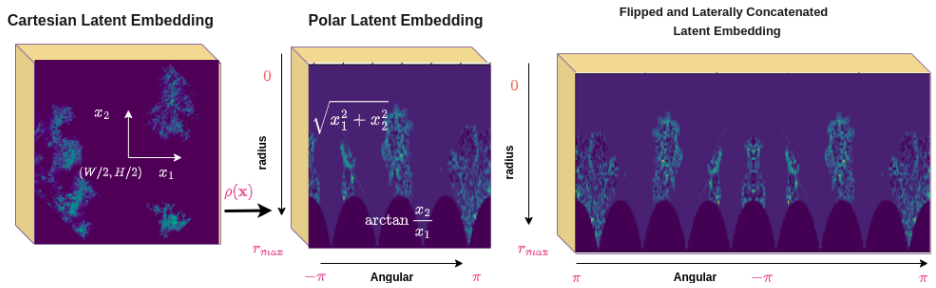


Figure 5.3: The image to the extreme left shows a sample *DEM* latent space in Cartesian form. The image in the center depicts the same embedding in a polar form; the image to the right is the result of flipping and concatenation operation.

Coarse RP Canonicalization: Given the normals $\mathbf{n}_c = [n_{c,x}, n_{c,y}, n_{c,z}]$ from `RANSAC`, and $\mathbf{n}_w = [0, 0, 1]$, we estimate the relative roll α and pitch β by solving:

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ \sin(\beta)\sin(\alpha) & \cos(\beta) & -\cos(\alpha)\sin(\beta) \\ -\cos(\beta)\sin(\alpha) & \sin(\beta) & \cos(\alpha)\cos(\beta) \end{bmatrix} \begin{bmatrix} n_{c,x} \\ n_{c,y} \\ n_{c,z} \end{bmatrix}.$$

The obtained closed-form solution is given as:

$$\alpha = \arctan\left(\frac{-n_{c,x}}{n_{c,z}}\right), \text{ and}$$
$$\beta = \arctan\left(\frac{n_{c,y}}{n_{c,z}\cos(\alpha) - n_{c,x}\sin(\alpha)}\right).$$

Fine Grained Canonicalization with ICP: We use the coarse estimates of α and β as described above, and refine them using `ICP` (initialized with coarse estimates) as:

$$R(\alpha, \beta) = \underset{(\alpha,\beta)}{\arg\min} ||\mathbf{C_w} - R(\alpha, \beta)\mathbf{C_c}||^2$$

Top-view Discretization: After performing the roll and pitch canonicalization, the top-view of the point cloud is discretized into uniform $2D$ grid cells to obtain the `DEM` $\mathbf{D}_c$ of point cloud $\mathbf{P}_c$. We define a grid G of dimension $G_w \times G_h$, and resolution d_g. Each grid cell $g_i \in G$ is assigned a set of points P_i^g based on the resolution d_g, and given a height value $h_i^g = \max(h(p) \mid p \in P_i^g)$, where $h(p)$ is the height of the point p, thus converting a point cloud to a `DEM`. Such a grid representation can be readily assimilated by Deep Networks ideally suited to exploit such structural information.

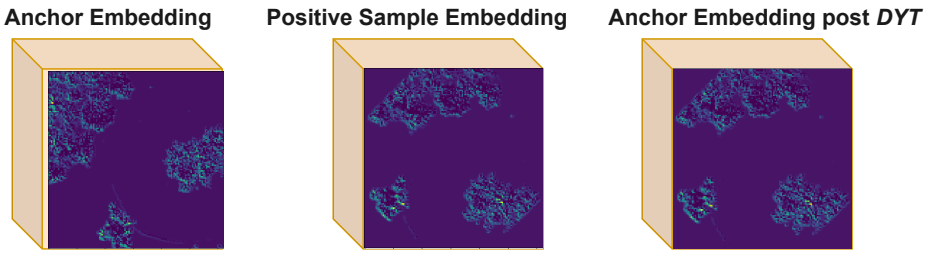


Figure 5.4: Visualization of the yaw alignment using DYT. Note that the anchor and the positive sample are not yaw aligned initially, however, post the *DYT* operation the two embeddings are aligned. We show the first channel of the feature volume as a binary image for ease of visualization.

5.1.2 Learning Compressed Latent Representation

We intend to use the DEMs to perform the LDC task, and an autoencoder architecture is used to generate compressed latent embeddings of a DEM. The *Encoder* consists of sequentially stacked CNN layers for feature extraction, and the encoding process acts as a compressor, resulting in a feature volume (latent embedding) requiring significantly less memory to store and transmit, in comparison to the original DEM (or its corresponding point cloud). We use ϕ to denote the embedding, and $\phi \in R^{4\times125\times125}$. Loop detection module is designed to perform view-point invariant place recognition using ϕ. However, for loop closure, we decode ϕ back to the DEM before performing pose estimation. Detailed architecture of our encoder-decoder is given below

The DEM is reconstructed in its original dimension at the receiver, aiding the *Loop Closure* module. The *AutoEncoder* architecture is shown in Table 5.2.

Operator	Stride	Filter	Padding	Output Shape
Conv2D	(1,1)	(3,3)	(1,1)	500×500
MaxPool2D	(1,1)	(2,2)	(0,0)	250×250
Conv2D	(1,1)	(3,3)	(1,1)	250×250
MaxPool2D	(1,1)	(2,2)	(0,0)	125×125
Conv2DTrans	(2,2)	(2,2)	-	250×250
Conv2DTrans	(2,2)	(2,2)	-	500×500

Table 5.2: First four rows show the encoder, and next two rows the decoder. *Conv2DTranspose* refers to the convolution 2D transpose operation within PyTorch[52] that is used for convolution with upsampling.

ResNet[28] based CNN layers have been used before the *Difference Layer*, to extract features the anchor's (positive or negative) yaw aligned embedding. As explained in Sect. 5.1.4, the output F_{diff} of the *Difference Layer* is sent to a CNN to get a scalar value indicating the distance between the two DEMs. The architecture of the CNN is shown in Table 5.3, the input to this part of the model is a feature volume of size $10 \times 1024 \times 1024$.

Operator	Stride	Filter	Padding	Output Shape
Conv2D	(2,2)	(5,5)	(0,0)	$64 \times 510 \times 510$
Conv2D	(2,2)	(5,5)	(0,0)	$32 \times 253 \times 253$
Conv2D	(2,2)	(1,1)	(0,0)	$4 \times 127 \times 127$
Linear	-	-	-	1×64516
Linear	-	-	-	1×100
Linear	-	-	-	1×10

Table 5.3: Architecture of the CNN used after the *Difference Layer*.

5.1.3 Differentiable Yaw Transformer *(DYT)*

Viewpoint invariance is an important property for robust place recognition/loop detection. Previous methods [8, 61] try to achieve this through data augmentation, where they rotate an input point cloud through randomly selected rotation angles. However, such methods do not generalize to complex sequences or large changes in viewpoint. [8] acknowledges that data augmentation by itself need not be sufficient for viewpoint invariance. Therefore, we propose a *Differentiable Yaw Transformer* (`DYT`) module that achieves viewpoint invariance without the need for explicit data augmentation. It achieves this by receiving the latent embedding of anchor and positive (or negative) `DEM` (denoted as ϕ_A, ϕ_p, and ϕ_n respectively) as its input and returning the relative yaw denoted by $\psi \in R$ at the output. Then, it rotates the anchor `DEM` so that the relative yaw between the anchor and positive is zero. The operations within the `DYT` are detailed below.

Cartesian to Polar Conversion (CPC): Let $\mathbf{G}$ be a group of rotation transformations (in `SO(2)`) parameterized by ψ s.t. $T_\psi : R^d \rightarrow R^d, \forall T_\psi \in SO(2)$. We define canonical coordinate for $\mathbf{G}$ such that a rotation by T_ψ in the Cartesian coordinates appears as a translation by ψ in the canonical coordinates. The polar coordinate system forms such canonical coordinates for the group of rotation transformations [66, 17], and can be obtained from Cartesian coordinates $\mathbf{x} : (x_1, x_2)$ as:

$$\rho(\mathbf{x}) = \left(\arctan \frac{x_2}{x_1}, \sqrt{x_1^2 + x_2^2} \right). \tag{5.1}$$

We perform the `CPC` independently for each channel of latent embedding tensor ϕ, and generate an output tensor of the same size. The visualization of the *CPC* process is shown in the first two columns of Fig. 5.3

Horizontally Padding Polar Embedding: As described above two embeddings related by a yaw rotation in the Cartesian coordinates are related by a translation after conversion to polar coordinates. However, if we try to estimate translation directly, the estimation process can only correlate between the overlapping regions. We observe that the horizontal axis of the polar latent embedding lies within the range $[-\pi, \pi]$ and is cyclic. The cyclic property allows us to pad the embedding by copying the embedding, flipping it (the flipped embedding will be within the range $[\pi, -\pi]$), and then use the flipped

version to horizontally pad the embedding. The resulting embedding is shown in 5.3 (extreme right). The operation doubles the size of the latent embedding to $4 \times 125 \times 250$, and allows us to use full embedding for translation estimation.

Correlation Layer: After padding the polar embedding from the positive (negative) embedding, we try to locate anchor embedding in it using correlation. We implement the layer as a convolutional layer with polar latent embedding of the anchor as a kernel, and perform cross-correlation over the horizontally padded polar feature volume of the positive/negative sample. This results in a 1D output of size $1 \times 1 \times 126$. The output of the correlation layer divides the 360 degrees of rotation into 126 bins, each of resolution 2.85 degrees (approximately). We apply softmax over the correlation score output to convert the score vector to the probability vector for various candidate translations. The predicted translation is multiplied by 2.85 to convert to predicted rotation angle.

Rotation Sampler: We construct a rotation matrix $R_\psi \in \texttt{SO(2)}$ from the predicted yaw angle (ψ) as determined from the previous step. Similar to [31], we use R_ψ to differentiably sample from the input feature volume and produce a warped output feature map, denoted as $\widehat{\phi}$. The operation is denoted as $\otimes$ in 5.2. Note that the operation is performed on $4 \times 125 \times 125$ dimensional embedding tensor in Cartesian coordinates. 5.4 depicts the result of the DYT module, it may be seen that the anchor and positive sample do not share the same orientation at the input of DYT. However, post-DYT, they have same orientation. For simplicity of illustration, we only show the first channel of the $4 \times 125 \times 125$ tensor. The warped anchor tensor is sent to the next module for loop detection.

5.1.4 Loop Detection

Our pipeline achieves rotation invariance through the RP canonicalizer and the DYT, the difference layer in the loop detection module achieves translation invariance and quantifies the similarity between the two yaw aligned DEMs. A fully convolutional network (CNN) is translation-equivariant. Our loop detection module consists of shared CNN layers to extract features $F_a \in R^{H \times W \times C}$ from the anchor and $F_{p/n} \in R^{H \times W \times C}$ features from the positive or negative sample DEM. The difference layer takes the two feature volumes as input and computes all pairs absolute differences between the pixels. To implement all pairs absolute difference we first construct a tile tensor $T_a \in R^{HW \times HW \times C}$ by first reshaping F_a to a $HW \times 1 \times C$ tensor, and then repeating first column in each channel by HW times. Mathematically: $\forall i \in \{0, 1, 2, .., H-1\}$ and $j \in \{0, 1, 2, .., W-1\}$.

$$T_a(iW + j, k, c) = F_a(i, j, c), \quad \forall k \in [0, HW - 1].$$

We compute T_p and T_n similarly, but additionally transpose each channel of the tensor at the end. This is equivalent to:

$$T_{p/n}(k, iW + j, c) = F_{p/n}(i, j, c), \quad \forall k \in [0, HW - 1].$$

T_a and $T_{p/n}$ allow us to compute all pair difference as: $F_{\text{diff}} = |T_a - T_{p/n}|$.

The difference layer results in a feature volume F_{diff} that quantifies the shared information between the two DEMs. F_{diff} is passed through proposed CNN architecture, resulting in a single scalar value indicating the distance between the two DEMs. A low value indicates loop detection. We train the proposed loop detection module using triplet based contrastive loss:

$$\mathcal{L}_{\text{triplet}} = \max\left(0, d\left(\widehat{\phi}_a, \phi_p\right) - d\left(\widehat{\phi}_a, \phi_n\right) + \xi\right), \tag{5.2}$$

where ϕ is the DEM encoding, $\widehat{\phi}$ is the yaw aligned DEM encoding, d is the distance between the two encoding computed by the loop detection module, and ξ is the margin for the triplet loss.

It should be noted that during back-propagation, the DEM encoder receives gradient both from the MSE loss of the autoencoder, as well as from the above triplet loss. Whereas the decoder is trained only using the MSE loss.

5.1.5 Loop Closure

After completing the loop detection process, we aim to estimate rigid body transform, SE(3), to align the query and retrieved point cloud. This process is known as loop closure (or point cloud registration). Let $\mathbf{P}_q$ be the query point cloud, with pose $\mathbf{T}_q^w$, centered at $\mathbf{O}_q$. Let $\mathbf{P}_r$ be the retrieved point cloud, centered at origin $\mathbf{O}_r$ with a pose $\mathbf{T}_r^w$. Refer to the loop closure block in the extreme right of 5.2. Both $\mathbf{T}_q^w$ and $\mathbf{T}_r^w$(denoted in dotted blue) are in the world frame of reference, and are unknown. We aim to find the relative transformation $\mathbf{T}_r^q$ (in solid yellow) that aligns $\mathbf{P}_q$ and $\mathbf{P}_r$. We estimate the relative pose through a two-step process. First we estimate the relative SE(2) transform between $\mathbf{P}_q$ and $\mathbf{P}_r$. Post that, we combine the SE(2) estimate with the initially estimated roll and pitch canonicalization to obtain the SE(3) pose estimation.

To estimate the relative SE(2) transform we decode the query and retrieved DEMs from their respective encoding. Then, key-points and correspondences between the query and retrieved DEMs is obtained using [14, 60]. An optimization problem is formulated as shown below to extract the relative SE(2) pose:

$$\underset{\psi, \mathbf{t}_{cr}^{cq}}{\arg\min} \| \left(\mathbf{R}(\psi)_{cr}^{cq} a_i + \mathbf{t}_{cr}^{cq}\right) - b_i \|^2. \tag{5.3}$$

Here, a_i, and b_i are the corresponding points on the query and target DEM respectively. The rotation matrix $\mathbf{R}(\psi)_{cr}^{cq} \in$ SO(2) is parameterized by the yaw angle ψ and the translation vector is denoted by $\mathbf{t}_{cr}^{cq} \in R^2$. The translation vector is scaled to the metric scale using the grid resolution d_g (c.f. Top-view Discretization within 5.1.1). The yaw angle ψ for the optimization is initialized using the yaw estimates from the DYT module. Let $\mathbf{R}(\alpha_q, \beta_q)_{cq}^{q}$ and $\mathbf{R}(\alpha_r, \beta_r)_{cr}^{r}$ (shown in pink in Loop Closure module of 5.2) be the rotation matrices that align the query ($\mathbf{P}_q$) and retrieved point cloud ($\mathbf{P}_r$) to their respective roll-pitch compensated frames $\mathbf{O}_q$ and $\mathbf{O}_r$. To estimate the SO(3) rotation matrix, we combine $\mathbf{R}_{cq}^q, \mathbf{R}_{cr}^{cq}$ and $\mathbf{R}_{cr}^q$:

$$\mathbf{R}_r^q = \mathbf{R}_{cq}^q \mathbf{R}_{cr}^{cq} (\mathbf{R}_{cr}^r)^{-1}.$$

We obtain the translation $\mathbf{t}_r^q$ by combining $\mathbf{t}_{cr}^{cq}$ and $d_r - d_q$: where d_r and d_q are the distance of the LiDAR from the estimated ground plane obtained (it is esimated along with the ground plane parameters through RANSAC).

$$\mathbf{t}_r^q = [\mathbf{t}_{cr}^{cq}(0), \mathbf{t}_{cr}^{cq}(1), d_r - d_q]$$

5.2 Datasets and Implementation Details

We use PyTorch, and train on a single NVIDIA GeForce GTX 1080 GPU, using a batch size of 12 and ADAM [36] as optimizer for 200 epochs for 8 hours. The learning rate is initialized to 4×10^{-4} and halved every 50 epochs. A $50m \times 50m$ point cloud is converted to a linearly scaled DEM representation of size 500×500 pixels. The triplet margin, k, in triplet is set to 0.75. Unlike [8, 9], we do not perform any augmentation on the input point clouds.

The model is evaluated on three publicly available datasets: **(1) GPR**[78]: This dataset consists a total of 15 sequences. We use sequence $1, 2, 3, 4, 5, 6, 8, 9, 11, 12$ for training and report evaluation results on sequence 10 and 15. **(2) KITTI** [20]: It consists of 11 sequences, similar to [8] we train on $05, 06, 07, 09$ and test on 00 and 08. **(3) Oxford RobotCar**[45]: The dataset consists of a total of 44 sequences, we use train and test split of the data as recommended by [68]. Similar to [8] we consider two point clouds to form a loop when the ground truth distance between them is less than $4m$. This rule allows us to sample triplets for training, an *anchor* and *positive* pair is formed when the distance between their poses is less than $4m$, *anchor* and *negative* pair is formed when distance is between $4m$ to $10m$. We report results for KITTI [20] and GPR [78] datasets in the next section. Results on Oxford Robotcar [45] are given in Section 5.3.3.

5.3 Experiments and Results

This section demonstrates the multifunctional abilities of our pipeline. We quantitatively and qualitatively demonstrate the superior efficacy of our method to perform 6-DOF LDC in a compressed point cloud space. The experiments are aimed at testing the algorithmic robustness and include challenging scenarios such as *Loop Detection* with a 6-DOF change in viewpoint and *Loop Closure* without any initial guess. We measure the bandwidth gained through our DEM encoding procedure. Furthermore, to measure the efficacy of our pipeline, we integrate it with *LIO-SAM* [64] and report the performance.

5.3.1 Loop Detection Results

We benchmark against both *learning* and *handcrafted feature* based approaches. *LCDNet* [8], *PointNetVLAD* [68], *PCAN* [79] and *OverlapNet* [9] are the SOTA deep learning based methods for loop detection, however they lack robustness and are unable to capture loops with a large change in viewpoint. Moreover, they are trained with significant augmentation which limits generalization. We use

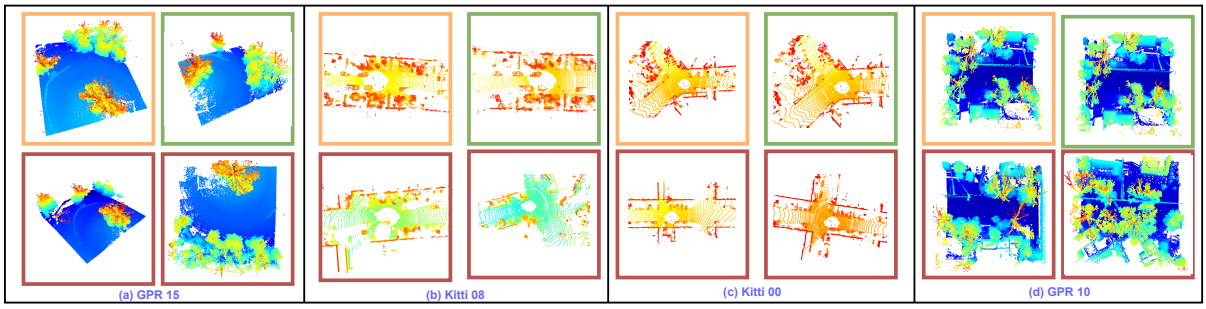


Figure 5.5: The image depicts the recall of our method on various sequences. For each of the four sequences, the point cloud in the orange box (top left) is the query point cloud, and the one within the green box (top right) is the top retrieved one. The point clouds in the red box (second row) are the second and third retrieved point clouds (left to right). This figure demonstrates the ability of our network to learn spatial priors. Note that the top-retrieved results are correct in all cases.

the official code and pre-trained models released by the respective authors. For fairness in comparison we retrain the model on datasets for which the pretrained model was not available. We also benchmark against a classical feature based approach, ScanContext [35].

We use *Precision-Recall (PR)* curves that are proven to be an effective metric to evaluate the loop detection [8, 9]. We follow *protocol 2* suggested by [8] to measure the *Average Precision (AP)*. Here we briefly describe the procedure: for a given query point cloud A we check all the point clouds in the database $\{B\}$. For a given pair of scans (A, B_i), we determine their distance (note that, lower the distance, greater the similarity) through the method explained in 5.1.4. If the distance is lesser than a fixed threshold then its considered to be a loop. We then check the Euclidean distance between the ground truth poses of the LiDAR scans, if the poses are less than $4m$ then we consider them as a *true positive*, if it happens to be greater than $4m$ then its a *false positive*. By varying this fixed threshold we can get multiple values of *precision-recall* which we use to plot the curve. This form of evaluation is proved to be effective in benchmarking algorithms for challenging scenarios such as a large change in viewpoint [8] .

The results are presented in 5.4 and the corresponding *PR* curves are depicted in 5.6. We show PR plots for one sequence from *KITTI* and and *GPR*. *LCDNet* [8] is the SOTA LDC method on the *KITTI* Dataset. We observe that on *KITTI08*, a challenging sequence which involves opposite views, our performance is better than *LCDNet* by approximately $\mathbf{10}\%$. On the *KITTI00* sequence our performance is second best to SOTA (lower by approximately $\mathbf{1}\%$). Similarly on both the *GPR* sequences $(10, 15)$ our method has the highest *AP* beating the closest *LCDNet* by approximately 9% and 1% respectively. Our method outperforms all the other competing benchmarks by a significant margin on both the datasets.

To demonstrate the ability of the DEM to learn the underlying spatial structure of the point clouds, in 5.5 we show the top 3 recalled point clouds for a given query. In addition to recovering the correct point cloud, it is important to note that the model has learnt to identify similar spatial structure. For instance, in the *GPR15* sequence there are trees in both the input (query) and the matched/recalled point cloud. A similar trend can be observed in *KITTI08*, where the query is a junction on the road and the retrieved

Method	KITTI		GPR	
	KITTI-00	KITTI-08	GPR-10	GPR-15
LCDNet [8]	**0.89**	0.76	0.82	0.88
OverlapNet [9]	0.61	0.22	0.75	0.57
PointNetVLAD [68]	0.40	0.39	0.50	0.54
PCAN [79]	0.46	0.20	0.39	0.20
ScanContext [35]	0.49	0.20	0.66	0.62
SOE-Net [74]	0.52	0.47	0.74	0.73
Ours	0.88	**0.84**	**0.91**	**0.92**

Table 5.4: AP Comparison for loop detection with SOTA.

point clouds also comprise of junctions. This shows that DEM is capturing appropriate representation that expose spatial structure of the point clouds which can be further encoded by a CNN model.

5.3.2 Loop Closure Evaluation

In this section we compare the proposed point cloud registration method against three categories of algorithms: **(1) Loop Detection and only yaw estmation** approaches such as ScanContext [35], OverlapNet [9] only estimate the yaw and not the complete 6-DOF pose. **(2) Loop Detection and 6-DOF Pose Estimation:** LCDNet [8] is a SOTA method in the 6-DOF LDC task and we choose to compare against it. **(3) Only 6-DOF relative pose estimation**: We compare with the SOTA point cloud registration technique (they do not perform loop detection), TEASER++ [76]. We also benchmark against classical method, ICP [80].

The official code open-sourced by the authors is used to benchmark [35, 9, 76, 8]. We use *Open3d* [84] implementation of ICP. Our experimental results are shown in 5.5, we evaluate our method based on *Average translation error (ATE)* in meters and *Average Rotation Error (ARE)* in degrees. Our method has the lowest *ATE* on the *KITTI* dataset (both the sequences) and on the *GPR10* sequence. It also has the lowest rotation error on *KITTI08, GPR10* and *GPR15*. On *KITTI00* sequence, our method has an error of 0.91^0 which is higher than *LCDNet*. It is important to note that methods such as [80, 76, 8] work on the entire point clouds and are not suitable for a multi-agent (or distributed) setting, as it would require transfer of large point clouds to close the loop. Our results can be further improved by the use of outlier resilient robust ICP formulation.

The 6-DOF Experiment: A portion of literature solely work on loop detection for SE(2) motion [9, 44, 35, 75], and recent methods such as *LCDNet* that work on 6-DOF. However, there is no explicit experiment designed to validate the performance of these algorithms on 6-DOF. Therefore, we design a novel experiment to validate the performance of the loop detection algorithm for a large 6-DOF change in viewpoint. The experimental setup is as follows, a query point cloud A is to be matched against a database set of point clouds $\{B\}$. We apply a synthetic *roll, pitch and yaw* rotations to both A and $\{B\}$, and use the augmented query and database point clouds to perform loop detection. *Average Precision*

Method	KITTI		GPR	
	KITTI-00	KITTI-08	GPR-10	GPR-15
LCDNet [8]	$0.77/\mathbf{1.07}$	$1.62/3.13$	$1.44/1.14$	$\mathbf{0.50}/4.81$
OverlapNet* [9]	$-/3.6$	$-/65.29$	$-/7.85$	$-/6.25$
Teaser++ [76]	$2.93/16.13$	$3.24/28.98$	$2.68/16.87$	$2.47/20.34$
ScanContext* [35]	$-/1.89$	$-/3.20$	$-/4.37$	$-/4.26$
ICP [80]	$2.23/9.12$	$2.31/161.16$	$2.32/7.81$	$2.87/8.36$
Ours	$\mathbf{0.72}/1.98$	$\mathbf{1.35/2.96}$	$\mathbf{0.95/0.85}$	$0.82/\mathbf{1.14}$

Table 5.5: Point Cloud Registration Comparison with SOTA. Result format: *TE(meters)/RE(degrees)*. * are algorithms that only estimate yaw, and they are not directly comparable with our 6-DOF method. - indicates not applicable.

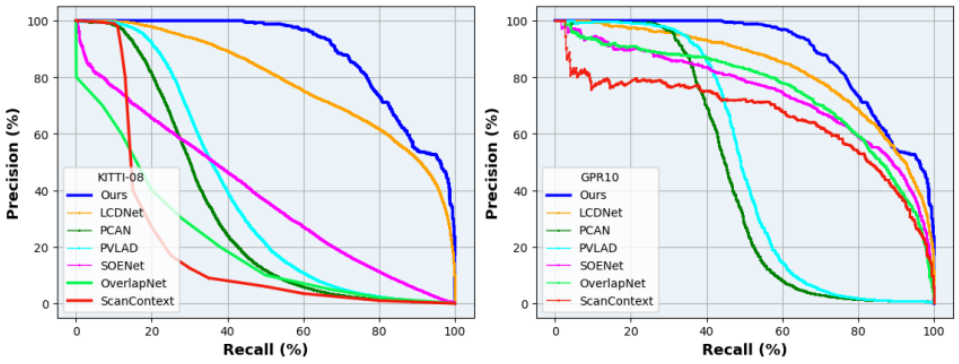


Figure 5.6: Comparison of various loop detection algorithms on the KITTI and GPR datasets. Our method performs best on KITTI08 which involves loops with opposite direction, and GPR10 and 15 sequences. Our method is second best to LCDNet [8] on KITTI00 by less than 1%.

is used as the evaluation metric. The test is conducted on six levels of increasing difficulty. In the first stage a random *Roll and Pitch* rotation between $(-10, 10)$ is applied, in the second we apply a random rotation between $(-20, 20)$ and so-on until the sixth where a random rotation between $(-60, 60)$ is applied. The rotated set of point clouds are fed into each of the algorithms and the resulting *Average Precision* is recorded at every level. [8, 74, 79, 68] are chosen for benchmarking as they are expected to work with 6-DOF motion and *KITTI and GPR* are used for testing. The results obtained are shown in 5.7. We observe that with the increase in the value of synthetic rotation all the baselines see a monotonic drop in performance. However, our method has a constant value of *AP* even for larger values of rotation. The use of the canonicalization procedure is the primary reason for the superior performance of our method. This experiment demonstrates the robustness of our pipeline for large change in viewpoints. It further highlights that pure data-augmentation during training need not result in a good performance in all scenarios. Note that although randomly sampled rotations are used, we apply the same value of rotation to all the algorithms. Even with a large rotation, such as 60 degrees, our method performs better than *LCDNet* on an average by upto 90%.

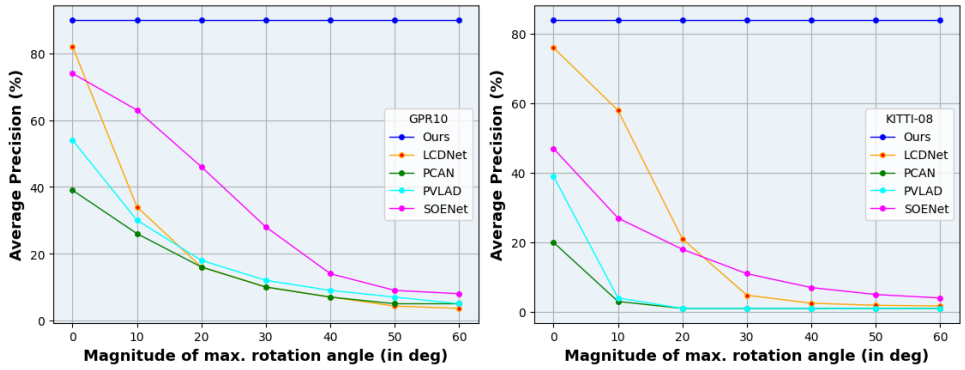


Figure 5.7: The plots demonstrate the results for the *6DOF Experiment*. It can be clearly seen that while other methods' *AP* values monotonically decrease, our method is robust to large changes in viewpoint and has a constant value.

5.3.3 Ablation Study

To evaluate the performance of *RP Canonicalizer* and `DYT`, we conducted ablation studies. The presence of the *RP-Canonicalizer* provided two critical advantages. Firstly, it allowed our method to operate on point clouds in 6-DOF motion, as demonstrated in the `LDC` results on *LUF* dataset (Tab. 5.4 and Tab. 5.5). Secondly, in the absence of the canonicalizer, we had to rely on methods such as [80, 76] to estimate the 6-DOF relative pose. However, we observed from Tab. 5.5 our loop closure pipeline provided improved accuracy over [80, 76]. Furthermore, we independently study the performance of the *RP-Canonicalizer*, we record that the Coarse RP Canonicalizer had an error of 3.249/4.1582 (R/P) degrees, while the fine alignment had an error of 1.2598/1.352 (R/P) degrees.

`DYT` Resolution Analysis:

The `DYT` module is designed to achieve yaw invariance. The predicted yaw is a multiple of the fixed resolution $2.85°$ (refer Sect . 5.1.3). Note that resolution is directly proportional to the spatial extent of the embedding (width $\times$ height). In Table 5.6, we analyse the effect of resolution on memory requirement, along with the effect on *Loop Detection's* performance, on *KITTI-08*.

Increase in resolution (decreasing the bin size) leads to quadratic increase in the spatial extent (width $\times$ height) of the feature volume. This makes it impractical to train high resolution models ($1°$), to make the model GPU compatible we further increase the number of layers in ResNet architecture to reduce the spatial extent (area of the embedding). This lead to lower resolution (increasing bin size, from $1°$ to $5°$), and a tradeoff between yaw error(degrading) and average precision(improving). Further decreasing the resolution (increasing the bin size, from $5°$ to $10°$) leads to a training failure where the model does not converge. Through this analysis we choose the `DYT` architecture with a resolution of $2.85°$, which can be trained effectively on a GPU, and leads to an optimal average precision of 0.84 with an acceptable error of $3.12°$ on *KITTI-08*.

Resolution (bin size)	Memory Requirement (↓)	Yaw Error (↓)	Average Precision (↑)
1°	1100	**1.62°**	0.68
5°	100	6.44°	0.78
10°	5	-	-
2.85°	**600**	3.12°	**0.84**

Table 5.6: Effect of resolution on Memory Requirement (in *kB*), Yaw Error and Average Precision on *KITTI-08*. For 2.85°, the spatial extent of latent embedding is 125×125. -: training fails to converge.

Benchmark on the Oxford Robot Car Dataset : We provide additional results in this section on Oxford Robot Car [45], *KITTI-00* [20] and *GPR-15* [78] [1]. The results on Oxford Robot Car [45] dataset is shown in Table 5.7. In Fig. 5.8, we show case the PR curve for *KITTI-00* [20], *GPR-15* [78] and Oxford Robot Car [45]. In *GPR-15* our method is better than *LCDNet* by 4% and the *KITTI-00* sequence we are the second best to *LCDNet* by less than 1%. In Fig. 5.12, we show qualitative results for *Loop Closure* on *GPR-10* [78] and *KITTI-08* [20].

Method	Oxford Robot Car [45]
PointNetVLAD [68]	0.47
PCAN [79]	0.63
ScanContext [35]	0.50
SOE-Net [74]	0.64
Ours	**0.72**

Table 5.7: Average Precision(AP) comparison for *Loop Detection* with SOTA on Oxford Robot Car [45] dataset.

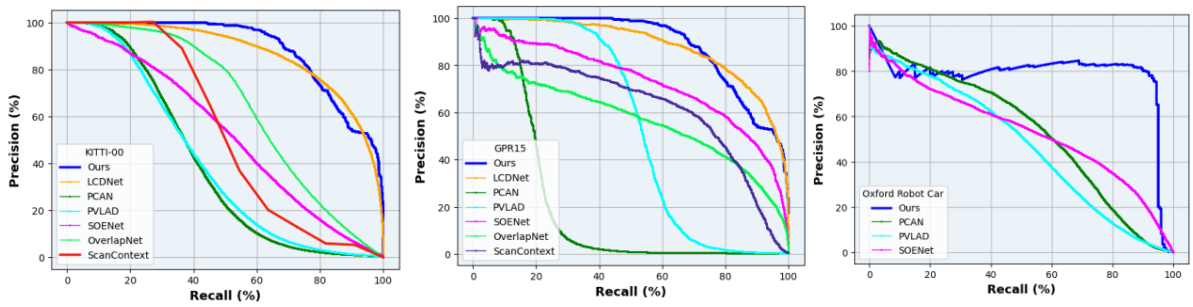


Figure 5.8: Comparison of the proposed approach with SOTA on *Loop Detection* on (left to right) *GPR-15* [78], *KITTI-00* [20] and Oxford Robot Car [45] .

5.3.4 Integration with Lio-SAM

We integrate the proposed pipeline with a backend LiDAR based SLAM system to test its efficacy. LIO-SAM [64] is a tightly coupled LiDAR Inertial SLAM with a factor graph based backend optimiza-

[1] We use the GPR dataset that is given in this link GPR Dataset

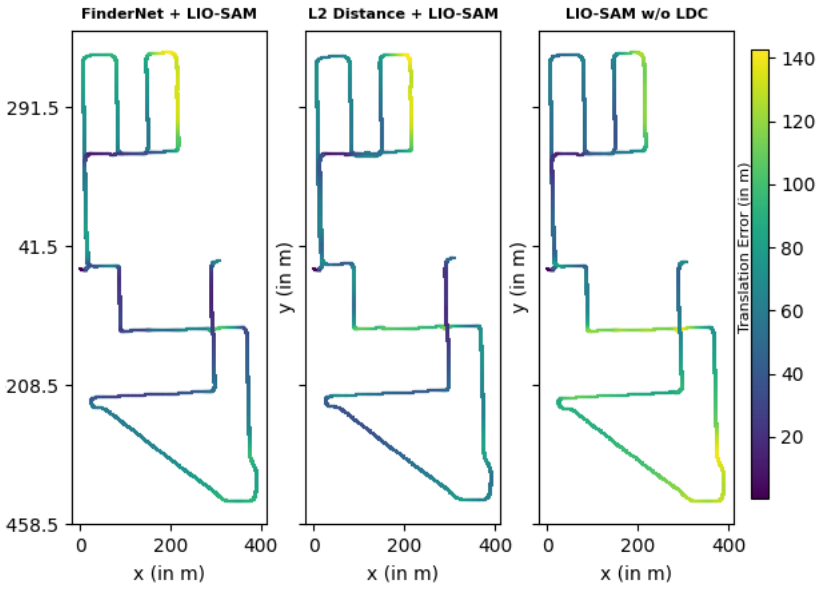


Figure 5.9: The figure depicts the *RMSE* obtained by integrating multiple LDC methods with *LIO-SAM* [64]. The *RMSE* of the total trajectory without LDC (right) is $48.79m$, if the Euclidean Distance based LDC (original full *LIO-SAM* center image) an *RMSE* of $35.69m$ is observed. However, with the integration of *FinderNet* (left) the *RMSE* reduces to $29.96m$.

tion. It is a SOTA method and has been shown to provide robust state estimates. We integrate the proposed pipeline with LIO-SAM to show that it is able to successfully detect and close loops.

Our experimental procedure is as follows, similar to [9, 35] we utilize the geometry of the factor graph for the LDC task, for every new state x_{i+1} added to the factor graph a local area of $15m$ is searched and the closest subset of possible matches is recovered. Our pipeline robustly estimates a viewpoint invariant loop from these initial set of matches. We measure the distance (as explained in Section 5.1.5) between the query all the point clouds in the subset, if the sample with the closest distance is lesser than a fixed threshold it is considered as a loop and a new link would be added into the graph for optimization. We test the integration of LIO-SAM[64] on *KITTI-08* [20] sequence and is depicted in Fig. 5.9. *KITTI-08* [20] is a challenging sequence consisting of loops with a large change in view point such as those with $90°$ and $180°$ (opposite side). Our method is capable of handling such large changes in viewpoint, a qualitative sample of the DYT response is shown in Fig. 5.11.

To evaluate the efficacy of the integration of *FinderNet* with *LioSAM* we benchmark it against the default *Euclidian Distance* based LDC within *LioSAM*. The results are shown in Fig. 5.9, where the use of *FinderNet* leads to a 16% decrease in *RMSE* in comparison to the *LIO-SAM*'s $L2$ distance based LDC [64]. This experiment further proves that our system can operate in real time.

5.3.4.1 Canonicalizer and DYT Performance Analysis

We independently evaluate the performance of the canonicalizer and DYT in this section. The experimental procedure is as follows: **(1)** 100 pairs of point clouds from the *GPR* dataset is sampled with poses

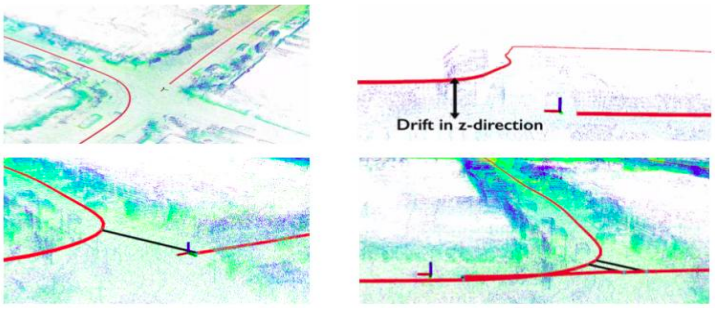


Figure 5.10: Qualitative results for Integration with *Lio-SAM* [64] experiment, the top left image shows the car approaching the loop, the image to the top right-shows the accumulated drift in the z-axis (axis perpendicular to the ground plane). The image in the bottom left shows the resulting detection by *FinderNet*, the pose post correction is shown in bottom right.

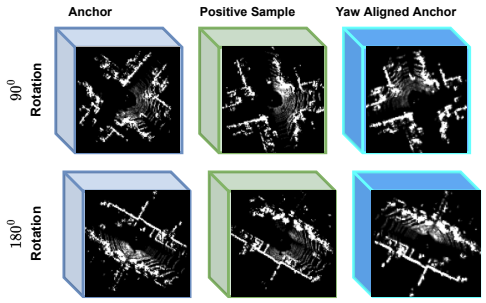


Figure 5.11: DYT response visualization of the latent embedding. It can be observed that the DYT predicts accurate yaw angles even and is able to transform the latent embedding even for large rotation such as $90°$ or $180°$. Observe that for the $180°$ change in viewpoint the latent space appears to be flipped post DYT.

less than $4m$ is sampled. **(2)** A random yet known synthetic 3-*DOF* rotation of up-to $60°$ (in roll, pitch and yaw) magnitude is applied both the point clouds. **(3)** The canonicalization procedure is performed and the estimated relative rotation is recorded. **(4)** The estimated values are compared with the known rotations and the mean of the absolute error values are obtained.. The coarse RP canonicalization had an error of $3.249/4.1582$ (R/P) degrees while the fine alignment works with an error of $1.2598/1.352$ (R/P) degrees respectively. The DYT was able to estimate the yaw with a error of $3.12°$.

5.3.4.2 Memory Analysis

Our method bypasses the computationally expensive decompression procedure for LDC. To measure the true gain in bandwidth we benchmark our method against other point cloud compression techniques. In Table 5.8 we report the bandwidth required to transfer *100* point clouds (100 as it is practical number of point clouds that needs to transferred between agents) which is of 500000 kB in size. The process of DEM creation itself, has lead to a compression by a factor close to 166 (the DEM requires 3000 (kB) of space) , and the latent embedding further compresses this by five times, leading to a representation that

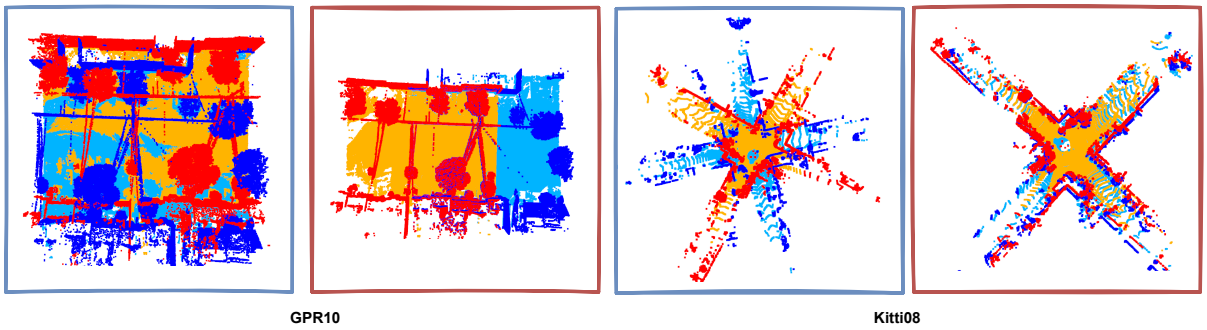


Figure 5.12: Qualitative Results for *Loop Closure* on *GPR-10* (left) [78] and *KITTI-08* (right)[20]. The query point cloud is shown in the orange, the retrieved point cloud is shown in blue. Observe that our method is able to register point clouds even with large displacements of $90°$ between the query and retrieved point clouds. **Note**: The color given to the point clouds are only for visual appeal and does not have a specific meaning.

requires 830 times less bandwidth in comparison to the original point cloud (the embedding is of size 600 (kB)). Our method further leads to closely 1.6 times better compression than [73]. *Range Images* are a popular form of point cloud representation [9], a `DEM` is a sparse representation in comparison to a Range Image and leads to an improvement of 1.50 times. Methods such as [8] are not meant for distributed setting, we however compare against their latent space and observe a massive improvement by a factor of 7000.

Table 5.8: Memory Benchmark Comparison (kB) Acronyms: **R** Range Image, **D** `DEM`, **E** Latent Embedding

Input Size	[73]	[8]	**R**	**D**	**E**
500000	1000	4220000	4500	3000	600

5.3.4.3 The *LiDAR UrbanFly Dataset*

Using the *Unreal* Editor [16] we create a custom environment consisting of a buildings, trees and uneven roads to evaluate `LDC` methods. The environment is scanned by a 64 channel `LiDAR` mounted on a quadrotor in 6-DOF motion. We create four such environments as shown in Fig. 5.13, Sequence $(1, 2, 3)$ are used for training and 4 for testing. The resulting *PR* curves are shown in Fig. 5.14. This experiment directly evaluates the performance of our method with 6-DOF viewpoint changes, we obtain a 10% improvement in *average precision* over *LCDNet* [8].

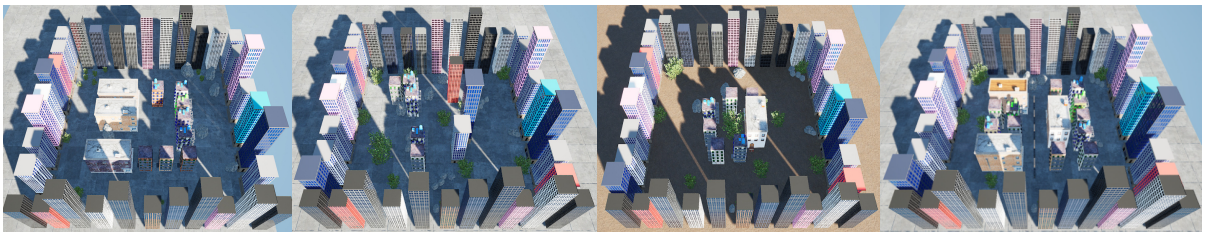


Figure 5.13: A glimpse of the Lidar-UrbanFly Dataset (LUF) Environment that we created. Sequence 1 to 3 were used to train the model and Sequence 4 was the test environment.

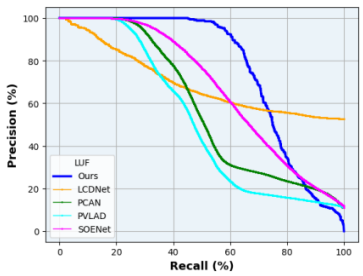


Figure 5.14: The PR Curves for the *LUF* sequence. The Average Precision of the benchmark methods are as follows, *LCDNet*[8] 0.69, *PointNetVLAD*[68] 0.67, *PCAN*[79] 0.58, *SOE-Net*[74] 0.50 and *FinderNet* 0.76.

5.4 Chapter Summary

FinderNet proposes a novel method for 6-DOF `LDC` which works on the latent (compressed) point-cloud representation. Our method is driven by the effective application of the canonicalization and `DYT` which allows us to achieve 6-DOF viewpoint invariance for large rotation angles. Furthermore, unlike the previous works our method does not require data-augmentation for training. Through our experiments we demonstrate significant improvement in performance on the real-world datasets.

Chapter 6

Conclusions

Through this work we explore the interactions between scene representation and robust autonomy. In *CCO-VOXEL* [27] we demonstrate that the non-parametric uncertainty in the distance to collision measurement and gradient field is responsible for the reduction in safety guarantees in deterministic planners. In response, we propose a planning framework that is robust to the distance and gradient inaccuracies present in voxel grids. The presence of uncertainty yields an optimization problem with probabilistic constraints known as chance-constrained optimization (CCO), but solving CCO is intractable in low-compute environments such as a quadrotor. In a novel approach, we reformulated CCO as a distribution matching problem that operates in a low-dimensional feature space learned through a deep auto-encoder. Such a surrogate formulation achieves real time performance with safety guarantees. We further developed a gradient-free trajectory optimization procedure to overcome the noise in the gradient field. CCO-VOXEL demonstrated a significant threefold improvement in safety metrics compared to benchmark algorithms. We also performed extensive work on real-world field testing of *CCO-VOXEL.*

In *UrbanFly*, we demonstrate that voxel grids are ineffective in representing sparse and noisy point clouds obtained from triangulation within a monocular *VI-SLAM* setup. As an alternative, we propose the *Uncertainty Integrated Cuboidal* (`UIC`) representation and show how it can be used to perform safe trajectory planning.

With *FinderNet*, our goal is to perform 6-DOF *Loop Detection and Closure* in a distributed collaborative `SLAM` setting. We found that a `DEM` representation effectively exposes the underlying structure and geometry within point clouds. Additionally, we have developed a `DYT` that can be trained with minimal supervision and no data augmentation.

Although progress has been made in robust autonomy, there is still a significant need for further research at the intersection of scene representation and navigation. For example, [27] could be studied in greater depth to enable uncertainty-aware safe navigation with probabilistic guarantees in dynamic environments. *FinderNet* [25] assumes that the ground plane is common between agents, research can be conducted towards relaxing this assumption by developing methods to directly extract geometric features from the point clouds without an intermediate `DEM`.

Related Publications

1. ***CCO-VOXEL: Chance Constrained Optimization over Uncertain Voxel-Grid Representation for Safe Trajectory Planning*** (2022). **Sudarshan S Harithas**, Rishabh Dev Yadav, Deepak Singh, Arun Kumar Singh, K Madhava Krishna. In IEEE International Conference on Robotics and Automation, ICRA (2022). **paper** , **Video**

2. ***UrbanFly: Uncertainty-Aware Planning for Navigation Amongst High-Rises with Monocular Visual-Inertial SLAM maps*** (2023). **Sudarshan S Harithas**, Ayyappa Thatavarthy, Gurkirat Singh, Arun Kumar Singh, K Madhava Krishna. American Control Conference (ACC2023). **paper**, **Video**.

3. ***FinderNet: A Data Augmentation Free Canonicalization aided Loop Detection and Closure technique for Point clouds in 6-DOF separation.*** (2023). **Sudarshan S Harithas**, Gurkirat Singh, Aneesh Chavan Samrat, Sarthak Sharma, Suraj Patni, Chetan Arora, K Madhava Krishna. Under review at In IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2023.

Other Publications

1. ***RP-VIO: Robust Plane-based Visual-Inertial Odometry for Dynamic environments***. Karnik Ram, Chaitanya Kharyal, **Sudarshan S Harithas**, K Madhava Krishna. In IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2021. **paper**, **Video**, **Data**

Bibliography

[1] I. Alzugaray, L. Teixeira, and M. Chli. Short-term uav path-planning with monocular-inertial slam in the loop. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2739–2746, 2017.

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

[3] L. Blackmore, H. Li, and B. Williams. A probabilistic approach to optimal robust path planning with obstacles. In *2006 American Control Conference*, pages 7–pp. IEEE, 2006.

[4] I. Bogoslavskyi and C. Stachniss. Fast range image-based segmentation of sparse 3d laser scans for online operation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 163–169. IEEE, 2016.

[5] S. Boyd. Stochastic programming. *Lecture Notes, Stanford University*, 2008.

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

[7] M. Castillo-Lopez, P. Ludivig, S. A. Sajadi-Alamdari, J. L. Sanchez-Lopez, M. A. Olivares-Mendez, and H. Voos. A real-time approach for chance-constrained motion planning with dynamic obstacles. *IEEE Robotics and Automation Letters*, 5(2):3620–3625, 2020.

[8] D. Cattaneo, M. Vaghi, and A. Valada. Lcdnet: Deep loop closure detection and point cloud registration for lidar slam. *IEEE Transactions on Robotics*, 2022.

[9] X. Chen, T. Läbe, A. Milioto, T. Röhling, O. Vysotska, A. Haag, J. Behley, and C. Stachniss. OverlapNet: Loop Closing for LiDAR-based SLAM. In *Proceedings of Robotics: Science and Systems (RSS)*, 2020.

[10] T. Cieslewski and D. Scaramuzza. Efficient decentralized visual place recognition from full-image descriptors. In *2017 International symposium on multi-robot and multi-agent systems (MRS)*, pages 78–82. IEEE, 2017.

[11] T. Cieslewski and D. Scaramuzza. Efficient decentralized visual place recognition using a distributed inverted index. *IEEE Robotics and Automation Letters*, 2(2):640–647, 2017.

[12] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen. Collision avoidance under bounded localization uncertainty. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1192–1198. IEEE, 2012.

[13] G. Costante, J. Delmerico, M. Werlberger, P. Valigi, and D. Scaramuzza. *Exploiting Photometric Information for Planning Under Uncertainty*, pages 107–124. 01 2018.

[14] D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018.

[15] J. Du, R. Wang, and D. Cremers. Dh3d: Deep hierarchical 3d descriptors for robust large-scale 6dof relocalization. In *European Conference on Computer Vision*, pages 744–762. Springer, 2020.

[16] Epic Games. Unreal engine.

[17] C. Esteves, C. Allen-Blanchette, X. Zhou, and K. Daniilidis. Polar transformer networks. *arXiv preprint arXiv:1709.01889*, 2017.

[18] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2016.

[19] F. Gao and S. Shen. Quadrotor trajectory generation in dynamic environments using semi-definite relaxation on nonconvex qcqp. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6354–6361. IEEE, 2017.

[20] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[21] B. Gopalakrishnan, A. K. Singh, M. Kaushik, K. M. Krishna, and D. Manocha. Prvo: Probabilistic reciprocal velocity obstacle for multi robot navigation under uncertainty. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1089–1096. IEEE, 2017.

[22] B. Gopalakrishnan, A. K. Singh, and K. M. Krishna. Closed form characterization of collision free velocities and confidence bounds for non-holonomic robots in uncertain dynamic environments. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4961–4968. IEEE, 2015.

[23] B. Gopalakrishnan, A. K. Singh, K. M. Krishna, and D. Manocha. Solving chance-constrained optimization under nonparametric uncertainty through hilbert space embedding. *IEEE Transactions on Control Systems Technology*, 2021.

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

[25] S. S. Harithas, , G. Singh, A. Singh, Chavan, S. Sharma, S. Patni, C. Arora, and K. M. Krishna. Findernet: A novel technique for lidar based 6dof distributed loop detection and closure. *Under Review*, 2023.

[26] S. S. Harithas, A. S. Thatavarthy, G. Singh, A. K. Singh, and K. M. Krishna. Urbanfly: Uncertainty-aware planning for navigation amongst high-rises with monocular visual-inertial slam maps. *arXiv preprint arXiv:2204.00865*, 2022.

[27] S. S. Harithas, R. D. Yadav, D. Singh, A. K. Singh, and K. M. Krishna. Cco-voxel: Chance constrained optimization over uncertain voxel-grid representation for safe trajectory planning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11087–11093. IEEE, 2022.

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

[29] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.

[30] S. Huang, N. Kwok, G. Dissanayake, Q. Ha, and G. Fang. Multi-step look-ahead trajectory planning in slam: Possibility and necessity. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1091–1096, 2005.

[31] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

[32] A. Jaegle, F. Gimeno, A. Brock, O. Vinyals, A. Zisserman, and J. Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.

[33] J. J. Johnson and M. C. Yip. Chance-constrained motion planning using modeled distance-to-collision functions. *arXiv preprint arXiv:2107.10953*, 2021.

[34] G. Kim, S. Choi, and A. Kim. Scan context++: Structural place recognition robust to rotation and lateral variations in urban environments. *IEEE Transactions on Robotics*, 2021.

[35] G. Kim and A. Kim. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4802–4809. IEEE, 2018.

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

[37] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame. Door-slam: Distributed, online, and outlier resilient slam for robotic teams. *IEEE Robotics and Automation Letters*, 5(2):1656–1663, 2020.

[38] P.-Y. Lajoie, B. Ramtoula, F. Wu, and G. Beltrame. Towards collaborative simultaneous localization and mapping: a survey of the current research landscape. *arXiv preprint arXiv:2108.08325*, 2021.

[39] C. Leung, S. Huang, and G. Dissanayake. Active slam using model predictive control and attractor based exploration. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5031, 2006.

[40] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz. Planercnn: 3d plane detection and reconstruction from a single image. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4445–4454, 2019.

[41] Y. Liu, F. Xue, and H. Huang. Urbanscene3d: A large scale urban scene dataset and simulator. 2021.

[42] K.-L. Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 4(10):1–3, 2004.

[43] L. Luo, S.-Y. Cao, B. Han, H.-L. Shen, and J. Li. Bvmatch: Lidar-based place recognition using bird's-eye view images. *IEEE Robotics and Automation Letters*, 6(3):6076–6083, 2021.

[44] J. Ma, J. Zhang, J. Xu, R. Ai, W. Gu, and X. Chen. Overlaptransformer: An efficient and yaw-angle-invariant transformer network for lidar-based place recognition. *IEEE Robotics and Automation Letters*, 2022.

[45] W. Maddern, G. Pascoe, C. Linegar, and P. Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[46] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[47] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 116–121. IEEE, 1985.

[48] R. J. Moss. Cross-entropy method variants for optimization. *arXiv preprint arXiv:2009.09043*, 2020.

[49] C. Mostegel, A. Wendel, and H. Bischof. Active monocular localization: Towards autonomous monocular exploration for multirotor mavs. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3848–3855, 2014.

[50] I. NUC. Intel nuc.

[51] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[53] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius. Sample-efficient cross-entropy method for real-time planning. *arXiv preprint arXiv:2008.06389*, 2020.

[54] S. N. J. Poonganam, B. Gopalakrishnan, V. S. S. B. K. Avula, A. K. Singh, K. M. Krishna, and D. Manocha. Reactive navigation under non-parametric uncertainty through hilbert space embedding of probabilistic velocity obstacles. *IEEE Robotics and Automation Letters*, 5(2):2690–2697, 2020.

[55] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[56] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[57] T. Qin, P. Li, and S. Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[58] I. RealSense. Intel realsense.

[59] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics research*, pages 649–666. Springer, 2016.

[60] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020.

[61] L. Schaupp, M. Bürki, R. Dubé, R. Siegwart, and C. Cadena. Oreos: Oriented recognition of 3d point clouds in outdoor scenarios. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3255–3261. IEEE, 2019.

[62] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli. Covins: Visual-inertial slam for centralized collaboration. In *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 171–176. IEEE, 2021.

[63] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[64] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 5135–5142. IEEE, 2020.

[65] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

[66] K. S. Tai, P. Bailis, and G. Valiant. Equivariant transformer networks. In *International Conference on Machine Learning*, pages 6086–6095. PMLR, 2019.

[67] V. Usenko, L. Von Stumberg, A. Pangercic, and D. Cremers. Real-time trajectory replanning for mavs using uniform b-splines and a 3d circular buffer. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 215–222. IEEE, 2017.

[68] M. A. Uy and G. H. Lee. Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4470–4479, 2018.

[69] L. von Stumberg, V. Usenko, J. Engel, J. Stückler, and D. Cremers. Autonomous exploration with a low-cost quadrocopter using semi-dense monocular slam. *arXiv preprint arXiv:1609.07835*, 2016.

[70] Y. Wang, Z. Sun, C.-Z. Xu, S. E. Sarma, J. Yang, and H. Kong. Lidar iris for loop-closure detection. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5769–5775. IEEE, 2020.

[71] L. Wiesmann, T. Guadagnino, I. Vizzo, G. Grisetti, J. Behley, and C. Stachniss. Dcpcr: Deep compressed point cloud registration in large-scale outdoor environments. *IEEE Robotics and Automation Letters*, 7(3):6327–6334, 2022.

[72] L. Wiesmann, R. Marcuzzi, C. Stachniss, and J. Behley. Retriever: Point cloud retrieval in compressed 3d maps. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[73] L. Wiesmann, A. Milioto, X. Chen, C. Stachniss, and J. Behley. Deep compression for dense point cloud maps. *IEEE Robotics and Automation Letters*, 6(2):2060–2067, 2021.

[74] Y. Xia, Y. Xu, S. Li, R. Wang, J. Du, D. Cremers, and U. Stilla. Soe-net: A self-attention and orientation encoding network for point cloud based place recognition. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 11348–11357, 2021.

[75] X. Xu, H. Yin, Z. Chen, Y. Li, Y. Wang, and R. Xiong. Disco: Differentiable scan context with orientation. *IEEE Robotics and Automation Letters*, 6(2):2791–2798, 2021.

[76] H. Yang, J. Shi, and L. Carlone. TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Trans. Robotics*, 2020.

[77] M. Yguel, O. Aycard, and C. Laugier. Update policy of dense maps: Efficient algorithms and sparse representation. In *Field and service robotics*, volume 42, pages 23–33. Springer, 2008.

[78] P. Yin, S. Zhao, R. Ge, I. Cisneros, R. Fu, J. Zhang, H. Choset, and S. Scherer. Alita: A large-scale incremental dataset for long-term autonomy. *arXiv preprint arXiv:2205.10737*, 2022.

[79] W. Zhang and C. Xiao. Pcan: 3d attention map learning using contextual information for point cloud based retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12436–12445, 2019.

[80] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[81] B. Zhou, F. Gao, J. Pan, and S. Shen. Robust real-time uav replanning using guided gradient-based optimization and topological paths. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1208–1214. IEEE, 2020.

[82] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[83] B. Zhou, J. Pan, F. Gao, and S. Shen. Raptor: Robust and perception-aware trajectory replanning for quadrotor fast flight. *IEEE Transactions on Robotics*, 2021.

[84] Q.-Y. Zhou, J. Park, and V. Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018.

[85] H. Zhu and J. Alonso-Mora. Chance-constrained collision avoidance for mavs in dynamic environments. *IEEE Robotics and Automation Letters*, 4(2):776–783, 2019.