University of Nevada, Reno

Correntropy: Answer to non-Gaussian noise in modern SLAM applications?

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computer Science and Engineering

by
Ashutosh Singandhupe

Dr. Hung La/Dissertation Advisor

May 2022

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

Ashutosh Singandhupe

entitled

Correntropy: Answer to non-Gaussian noise in modern SLAM applications?

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Hung La, Ph.D.
Advisor

Sushil Louis, Ph.D.
Committee Member

Alireza Tavakkoli, Ph.D.
Committee Member

Tin Nguyen, Ph.D.
Committee Member

Hao Xu, Ph.D.
Graduate School Representative

David W. Zeh, Ph.D., Dean
Graduate School

May, 2022

Abstract

by Ashutosh Singandhupe

The problem of non-Gaussian noise/outliers has been intrinsic in modern Simultaneous Localization and Mapping (SLAM) applications. Despite numerous algorithms in SLAM, it has become crucial to address this problem in the realm of modern robotics applications. This work focuses on addressing the above-mentioned problem by incorporating the usage of correntropy in SLAM. Before correntropy, multiple attempts of dealing with non-Gaussian noise have been proposed with significant progress over time but the underlying assumption of Gaussianity might not be enough in real-life applications in robotics.

Most of the modern SLAM algorithms propose the 'best' estimates given a set of sensor measurements. Apart from addressing the non-Gaussian problems in a SLAM system, our work attempts to address the more complex part concerning SLAM: (a) If one of the sensors gives faulty measurements over time ('Faulty' measurements can be non-Gaussian in nature), how should a SLAM framework adapt to such scenarios? (b) In situations where there is a manual intervention or a 3rd party attacker tries to change the measurements and affect the overall estimate of the SLAM system, how can a SLAM system handle such situations?(addressing the Self Security aspect of SLAM). Given these serious situations how should a modern SLAM system handle the issue of the previously mentioned problems in (a) and (b)?

We explore the idea of correntropy in addressing the above-mentioned problems in

popular filtering-based approaches like Kalman Filters(KF) and Extended Kalman Filters(EKF), which highlights the 'Localization' part in SLAM. Later on, we propose a framework of fusing the odometeries computed individually from a stereo sensor and Lidar sensor (Iterative Closest point Algorithm (ICP) based odometry). We describe the effectiveness of using correntropy in this framework, especially in situations where a 3rd party attacker attempts to corrupt the Lidar computed odometry. We extend the usage of correntropy in the 'Mapping' part of the SLAM (Registration), which is the highlight of our work. Although registration is a well-established problem, earlier approaches to registration are very inefficient with large rotations and translation. In addition, when the 3D datasets used for alignment are corrupted with non-Gaussian noise (shot/impulse noise), prior state-of-the-art approaches fail. Our work has given birth to another variant of ICP, which we name as Correntropy Similarity Matrix ICP (CoSM-ICP), which is robust to large translation and rotations as well as to shot/impulse noise. We verify through results how well our variant of ICP outperforms the other variants under large rotations and translations as well as under large outliers/non-Gaussian noise. In addition, we deploy our CoSM algorithm in applications where we compute the extrinsic calibration of the Lidar-Stereo sensor as well as Lidar-Camera calibration using a planar checkerboard in a single frame. In general, through results, we verify how efficiently our approach of using correntropy can be used in tackling non-Gaussian noise/shot noise/impulse noise in robotics applications.

Acknowledgements

I would like to thank Dr. Hung La, Dr. Sushil Louis, Dr. Alireza Tavakkoli, Dr. Tin Nguyen, and Dr. Hao Xu for being on my committee, with special thanks to Dr. Hung La for giving me the opportunity to complete this research as part of the Advanced Robotics and Automation (ARA) Laboratory. I would also like to thank my lab-mates for their help with this research: Habib Ahmed, Cadence Motley, Son Nguyen and Andrew Washburn . Lastly, I would like to thank my friends and family for helping to support me through this hectic time in my life.

This work is partially supported by the U.S. National Science Foundation (NSF) under grants NSF-CAREER: 1846513 and NSF-PFI-TT: 1919127, and the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (USDOT/OST-R) under Grant No. 69A3551747126 through INSPIRE University Transportation Center (http://inspire-utc.mst.edu) at Missouri University of Science and Technology, and the National Aeronautics and Space Administration (NASA) Grant No. NNX15AI02H issued through the NVSGC-RI program under sub-award No. 19-21, the RID program under sub-award No. 19-29, and the NVSGC-CD program under sub-award No. 18-54.

The views, opinions, findings and conclusions reflected in this publication are solely those of the authors and do not represent the official policy or position of the NSF, USDOT/OST-R, and NASA.

Contents

List of Figures

Chapter 1

Introduction

Autonomous navigation is a widely researched topic in robotics, augmented/virtual reality and, more dominantly, in self-driving cars. Robotic autonomous navigation has been there for more than 30 years and has contributed significantly to the industry targeting from small scale driven applications to large scale, which resulted in the advent of this decade's self-driving cars and other autonomous robots. The ease with which most animals and human beings navigate in the environment can be replicated in robots. However, this complex process of navigation, no wonder how well we do, can not be easily represented mathematically. The only way that dumb robots can be made to navigate in an environment is to represent the environment in some simpler forms, which can be algorithmically justified. Simultaneous Localization and Mapping (SLAM) is an algorithmic process of a robot/sensor system, which involves perceiving the environment using sensors and estimating its position of itself

in the environment simultaneously [1]. For a robot, the environment is represented as a culmination of different geometrical structures (landmarks, obstacles etc.), also called a map. The term pose or robot state represents the position and orientation of the robot. It is generally termed as the robot state. The map assists the human operator in visualizing an unknown environment and setting up the robot's path for navigation. Another significant advantage the map provides is that it helps in minimizing the error while estimating the robot state (pose) during navigation. For example, in keeping track of 'visited landmarks', the robot can detect a loop, minimizing localization error, which is quite similar to how we humans navigate. Research in autonomous navigation has resulted in numerous algorithms such as Rapidly exploring Random Trees (RRT), extended RRT (RRT*), Rapidly-exploring Random Graph (RRG), Probabilistic Roadmap (PRM), etc., [2] [3]. These algorithms have directed many researchers to explore and improve robot navigation in complex environments. However, the current state of the art demands more improvement since it is yet to be fully solved for real-time dynamic environments.

1.1 Simultaneous Localization and Mapping

SLAM is a heavy component in modern autonomous systems. The drive for SLAM research was ignited with the inception of robot navigation in Global Positioning Systems (GPS) denied environments. Although GPS improves localization, numerous SLAM techniques are targeted for localization with no GPS in the system. Initially,

probabilistic estimation techniques were introduced, like Kalman Filters (KF), which were later extended to Extended Kalman Filters (EKF), and Unscented Kalman Filters (UKF) for non-linear systems [1]. Particle filters like Rao-Blackwellized and Monte Carlo filters have also contributed significantly to the SLAM research [1]. Another approach that has grabbed attention is the graph-based SLAM, where the robot pose is represented as a node/vertex in a graph, and the edges represent the errors in measurements from various sensors. Subsequently, the process involves generating a pose graph and minimizing the error using mathematical techniques like Gauss-Newton/Levenberg–Marquardt [4]. SLAM techniques like Oriented fast and Rotated Briefs-SLAM (ORB2-SLAM) [5, 6] are based on graph-based localization. Another interesting approach has grabbed attention since the advent of deep learning with a focus on Convolutional Neural Networks (CNN). Quite interesting results were observed, especially with the work on CNN-SLAM [7]. Experiments show that robot pose or localization could be achieved from a pair of images acquired by a moving robot through deep learning or CNN. Even though the CNN-SLAM approach is promising, this approach has invited a few challenges that need to be addressed. Deep learning requires high-end Graphics Processing Unit (GPU) systems, which is still a challenge for robotic embedded systems. Moreover, SLAM systems are seen to be directed on continuous open-world scenes where the environment keeps changing. These changes need to be learned continuously for a deep learning system. To our best knowledge, deep learning has not significantly evolved in the current state of SLAM to learn the dynamic changes in the environment robustly. From the various techniques

introduced in SLAM, one can observe that SLAM is inclined to combine various fields like signal processing, deep learning (CNN-SLAM), and computer vision.

The problem of estimation (or Localization in SLAM) is one of the most significant components of robotics. If the system is linear, Kalman Filter (KF) is usually used to solve the estimation problem. In the presence of Gaussian noise, the KF performs well [8]. However, in the presence of non-Gaussian noise, the performance of the KF deteriorates, especially in the presence of impulsive noises. KF is based on the known Minimum Mean Square Error (MMSE) criterion, which is sensitive to significant outliers and causes the deterioration of the robustness of the KF in non-Gaussian noise environments [9]. Our work is focused on using Correntropy in various filtering and SLAM applications.

Despite the tremendous progress made in SLAM in the past 30 years, one question still bothers the robotics community, 'Is SLAM solved'? [1]. Our understanding is that SLAM is an estimation problem. Given the complexities of the environment and uncertainties in sensor measurements, SLAM is yet to arrive at a complete solution. As of our knowledge, SLAM is still unsolved. A good solution significantly relies on the environment, the robot, the uncertainties in the sensor measurements and the level of performance that we intend to achieve.

1.2 Correntropy

Different optimization criteria (for removing outliers) based on information learning have gained significant attention in the past few years. Information-theoretic quantities can capture higher-order statistics and offer potentially significant performance improvement in machine learning and signal processing applications. Correntropy as a non-linear similarity measure in kernel space has its root in Renyi's entropy [10] [11]. Prior to Correntropy, multiple methods were introduced to handle state estimation in the presence of non-Gaussian noise. To the best of our knowledge, there are three main approaches to improving the system's robustness.

The first approach uses filters that assist in removing both outliers and non-Gaussian noise. Noise distributions such as t-distribution and heavy-tailed distribution are considered in [12] and [13], but it faces the problem of handling more than one-dimensions. Computational cost and implementation difficulty are the other factors that prohibit its use.

The second approach dictates the use of a Multiple-Model (MM) filter [14]. This method assumes that any non-Gaussian distribution can be approximated as a finite set of Gaussian distributions with different modes. The Probability Density Function (PDF) that dictates the state posterior is considered the weighted sum of Gaussians. The Gaussian Sum Filter (GSF), which uses a bank of KFs, is an excellent example of this, but it is very computationally expensive since the number of modes increases exponentially with the number of filters in the bank [15].

The third approach uses Monte Carlo (MC) sampling that allows approximate representation of any probability distribution [16]. Particle filters are an excellent example of this, where the state posterior is represented by a set of random samples with associated weights. Ensemble KF (EnKF) is another example close to particle filters where the state posterior is estimated using a finite set of random samples. Similar to the Particle filter, another approach named Unscented KF (UKF) uses a deterministic sampling approach that estimates the mean and the covariance matrix of the state with a minimum set of points called sigma points [17]. Again, the computational cost is a significant problem in all of the above methods.

In this work, we explore the idea of Correntropy in applications in SLAM for handling outliers/non-Gaussian noise. We also incorporate Correntropy in addressing the problems of registration/alignment of point clouds. The major contributions of our work are given in the next subsection.

1.3 Contributions of Proposed Approach

This work aims to address the problem of non-Gaussian noises in modern SLAM systems. More importantly, we use the idea of Correntropy to address this issue and employ its usage in various algorithms applied in modern autonomous systems. The major highlights of this issue are addressed as follows:

- We present the idea of Correntropy and highlight its importance in handling non-Gaussian noise/outliers.

- We introduce a framework for using Correntropy in a KF and evaluate our approach when the sensor measurements are affected by non-Gaussian noise.

- For all the experiments conducted, we add non-Gaussian noise along with the original data coming from the sensor measurements. This serves two purposes:

  - Since we manually add noise in the sensor measurements, we verify the robustness of a system that incorporates Correntropy with Kalman Filter compared to traditional methods.

  - Handling 'manually added non-Gaussian noise' also addresses how an autonomous system, when attacked by a third party, can safely estimate it's state.

- We also propose a technique to incorporate Correntropy in addressing the problem of registration which is the highlight of this work. Concerning the problem of registration, we highlight the following observations:

  - Through results, we see how well the Correntropy can be used to address the registration problem. We also see that in comparison to the previous approaches, our approach outclasses other state-of-the-art approaches under various rotations and translations between the '*Source*' and the '*Target*' datasets.

  - We also show the efficiency of our approach when the point cloud data is affected with manually injected non-Gaussian noise and how well our approach can benefit in addressing the problem of registration.

- Using our proposed registration algorithm, we calibrate the Lidar and the Stereo sensor using a planar checkerboard. The entire process requires only a single frame of data acquired from both sensors.

1.4 Summary

This work is based on incorporating Correntropy in various applications in SLAM. This chapter began by describing the scope and the importance of SLAM in various robotics applications. Alongside, we also present Correntropy, and it's importance in removing non-Gaussian noise/outliers. Next, we touch upon the background of correntropy and SLAM techniques in modern robotics applications.

The remaining portion of the work is described in multiple chapters, where Chapter 2 describes the Literature review of concepts relating to SLAM and Correntropy. Chapter 3 describes the concepts relating to the idea of Correntropy, and it's scope and applicability in SLAM. Chapter 4 describes the idea of fusing Kalman Filter with the idea of Correntropy, and we see the importance of Correntropy in handling non-Gaussian noise in the basic Kalman filtering framework. Chapter 5 describes a framework for introducing the Maximum Corentropy criterion in the Extended Kalman Filter for handling non-Gaussian noise. Chapter 6 describes the idea of

using Correntropy in addressing the problem of registration which is the highlight of this work. We evaluate our approach in multiple datasets and compare the RMSE to most of the other state-of-the-art approaches. Chapter 7 and Chapter 8 describe an application of using Correntropy Similarity Matrix Iterative Closest Point Algorithm (CoSM-ICP) in performing Lidar-Stereo and Lidar -Camera Calibration respectively.

Chapter 2

Literature Review

The whole approach of SLAM is based on a robot localizing itself, given the sensor measurements. However, the traditional SLAM algorithms do not extend to performing the task of a robot driving itself to collect more information about the environment. In a more general sense, traditional SLAM algorithms only estimate localization when it is navigated or assisted by an external source or through an external command. There is no 'conscious' effort from the robot to navigate itself and collect the data from the environment. Active SLAM, exploration and localization done at the same time, is an approach that attempts to solve this problem. Active SLAM basically tries to solve it in 3 steps. The first step specifies the robot trying to find possible actions (e.g., turn right, turn left, forward, backward, path selection, etc.) that it could take given the map space; however it exposes the challenge of increased computational complexity. The second step says that, even if an action is

confirmed to be taken, the logic behind performing that action needs to be justified with respect to the goal of the task, as well as the complexity of the future action that could be taken to achieve that goal. The final step indicates that even if the action is performed, it is quite difficult to arrive at a conclusion about whether the exploration task has been completed or not. Based on our knowledge, active SLAM still requires mathematical proofs at various aspects [1].

This chapter primarily focuses on reviewing various SLAM techniques that were tried and tested on various autonomous robots. We are risking an attempt to classify various SLAM techniques, which are based on sensors used for localization and the ability of the SLAM algorithms to detect a loop closure. Loop closure is a technique for detecting a visited landmark or a scene in an environment. As of our knowledge, very few of the state-of-the-art algorithms we have encountered have solved the loop closure problem with respect to various autonomous systems.

The remainder of the chapter is organized as follows. It introduces techniques, which are further divided into Light Detection and Ranging (Lidar) based techniques (including Lidar and monocular camera) and Stereo-based SLAM techniques. We also intend to explore the security aspects in relation to autonomous systems and explore briefly the threats associated with them. Finally, conclusions are discussed at the end of this chapter.

2.1 Lidar based odometry

There are numerous algorithms written for estimating odometry using Lidar. Implicit Moving Least Squares SLAM (IMLS-SLAM) [18] is quite popular and uses a scan-to-model matching framework. Initially, it uses an algorithm to sample the 3D scans and uses IMLS for surface reconstruction, which is claimed to have an improved matching quality. One key factor to note in this work is that it uses only 3D Lidar sensors for odometry estimation. The work claims to perform better results than the state of the art algorithm for odometry estimation, Lidar Odometry and Mapping (LOAM) [19, 20]. However, the KITTI website[21] shows that LOAM outperforms every other algorithm that has been tested on all the odometry datasets.

Another work by [22], also called as Lidar-Monocular Visual Odometry (LIMO), have proposed a method which uses data from both Lidar and monocular camera. It first calculates the camera features and estimates the depth using the Lidar data corresponding to those features. Fusing the data together, it estimates the motion using a technique called bundle adjustment. The system for this algorithm can be better explained in steps/blocks. The first block relates to the camera, where it extracts the features. It includes a feature tracking step and a feature association step. Feature tracking is done using the Viso2 library. The feature association step is mainly related to extracting the depth of the camera features using the highly precise Lidar data. The Lidar point cloud is transformed into the camera frame and projected into the image plane. Then, for each feature, the following steps are performed. (1)

From a projected set of Lidar points, the algorithm chooses a set of points that are around a given feature. They use a rectangle to define the neighbourhood around that point. (2) Then, it performs a plane estimation using histograms of depths with fixed bandwidth. This helps in estimating the depth around corner features as well. (3) Then, the algorithm estimates the plane that fits the feature using the triangulation method. However, depth estimation for the features on the road includes another preemptive processing, which includes RANSAC for plane fitting. After performing the above steps, the next step includes a frame to frame odometry estimation using the perspective-n-point problem. Besides the procedures described above, specific steps need to be addressed, including strategies to select the data to increase efficiency and robustness. It is essential to select only a few important landmark features since there could be many in a dynamic environment that would dramatically increase computation complexity. For these reasons, the landmarks are classified into near, middle and far. Finally, it uses bundle adjustment on these detected features to estimate the ego-motion (aka motion estimation of a camera system). This approach was entirely evaluated on the KITTI dataset. The estimated trajectories are precise with low drift, but it does not solve the loop closure problem. On the KITTI datasets, LIMO has a translation error of 0.93% and a rotation error of 0.00026 deg/meter, which has proved to be a significant contribution to the robotics community.

A different method proposed in [23, 24] deserves attention. Here, the authors put forward a method that utilizes depth data to estimate the camera motion. It also uses

bundle adjustment to refine the motion estimation. At the time of the release of this method, it was ranked as the first in the KITTI benchmark visual odometry methods. As a first step or block, visual features are detected and tracked. Depth images, which could be from the RGB-D camera or from the point clouds, are registered in the depth registration block using the estimated motion. The final step is called the frame-to-frame motion estimation, which uses features as input acquired using the sequence of images, and then these features are fed to the bundle adjustment procedure. The results were evaluated on the KITTI dataset, wherein in the urban environment, the relative mean position error was 1.05%, and in the highway, it was 1.86%.

Another work by [25] offers a novel technique called Simultaneous Trajectory Estimation and Mapping (STEAM). This technique trains a Gaussian process model using the ground truth. The input to this system is a well-detected feature extracted from the point clouds, and the output of the system is the predicted poses that are computed using the estimator and the ground truth. On a deeper level, this algorithm starts with Lidar point cloud downsampling, where the heavy data of point clouds is reduced to sparse points called key points using normalized intensity values. A point can be selected as a key point or not if it satisfies certain conditions based on the proposed algorithm. Then these sparse point clouds are matched based on Euclidean distance. For estimating the trajectory, they implement the STEAM framework in which continuous-time trajectory is estimated as Gaussian process regression. The authors have also mentioned a significant point that, for continuous-time trajectory

estimation, the Gaussian regression problem is quite different from predicting odometry data. In order to reduce the errors in the odometry, the algorithm calculates the pose change from frame to frame and then compares it against the ground truth [26–31]. In Gaussian process modelling, the model is learned from noisy observations. So it becomes significant to select the features to detect in order to build a correct model. The results were evaluated on a KITTI dataset, and the overall error from all the path segments was 1.16%.

A different approach to the SLAM problem is the Closest Probability, and Feature Grid SLAM (CPFG-SLAM) [26], which has proposed a technique for localizing an unmanned vehicle in the off-road environment. In essence, it combines the features of the point cloud with probability and the occupancy probability of the grid map. Expected Maximization (EM) is further used to build the optimization function to match between the point cloud and grid map. This technique comprises three steps: data pre-processing, pose estimation and updating the feature grid map. Data pre-processing constitutes the filtering and classification of the point cloud. Pose estimation comprises estimating the pose and the position by matching the point cloud to the map. Finally, updating the point cloud features consists of extracting point cloud features and updating the probability of the grid. Later on, the EM algorithm is performed using the Levenberg-Marquardt algorithm. Despite high localization accuracy, this algorithm is not robust against dynamic environments, and also it does not solve the loop closure problem.

Another approach, by previously acclaimed authors [23], have proposed real-time monocular odometry, which is enhanced by depth data. This method is worth mentioning since it estimates depth from camera motion using sparse depth data too. It achieves this result by a method called triangulation that uses previously estimated motion and features from the image for which depth data is unavailable. Later on, it uses bundle adjustment to refine the estimated motion. Firstly, it tracks visual features from the images. Visual features are computed using the Harris corner detection algorithm and are tracked using Kanade Lucas Tomasi (KLT) method [32]. Next, it uses the depth data (either from an RGB-D camera or a Lidar) to register the point clouds with the depth using the estimated motion. The frame to frame motion estimation is done using bundle adjustment, whose inputs are the features extracted from the sequence of images. One interesting thing to note in this algorithm is that it uses both known and unknown depths of features in order to estimate the odometry of the camera.

One of the novel techniques that demands attention is from [31], which uses a learning approach. This technique essentially trains a Gaussian process regression model using data with ground truth. All the high-level features that are derived from the Lidar point clouds are used as input, and the predicted biases between poses from the estimator and the ground truth are the output of the system. However, the whole process is divided into a number of steps. First, the point clouds are downsampled to represent only the key points or the well-featured points in the point cloud. In this step, it calculates the eigenvalues of the matrix, which represent the k-nearest

neighbours of the point on the point clouds and sets a threshold through some functions to classify it as a key feature point. It uses the libpointmatcher library to do so [33]. Secondly, for two such downsampled point clouds, the point clouds are matched based on their Euclidean distance. It uses libnabo for matching [34]. Thirdly, it uses the STEAM framework, as discussed previously by [25], for trajectory estimation. However, only the odometry section of the STEAM framework is implemented here. More information about how the error is predicted and corrected is derived from the STEAM framework. On the KITTI datasets, it performs relatively better but not as good as IMLS and LOAM algorithms, as discussed earlier. Based on the authors, since this is strictly odometry, it does not solve the loop closure problem or reduce the drift in the odometry. Our understanding is that this algorithm has the potential to use a deep learning framework for better odometry estimation and correction rather than using a Gaussian prediction model.

Another novel approach for localization was made by [35] where a Surfel based map is used. The changes in the robot pose can be estimated by the data association between the current scan of the Lidar and the model view from the surface map. This technique is also called Surfel based Mapping (SuMa), which builds globally consistent maps. In addition to that, Surfel allows us to represent large scale environments and also maintains detailed geometric information of the point clouds. Based on the current rapid development in computation, rendering surfels is relatively fast. Odometry is computed using frame-to-model ICP with a point-to-plane error metric. The error is minimized using the Gauss-Newton minimization algorithm. This algorithm has been

evaluated on the KITTI dataset, which shows an average rotational error of 0.00032 deg/m and a translational error of 1.4%.

2.2 Stereo Based Odometry

Perhaps the current state of the art algorithm for the stereo visual odometry is the SOFT-SLAM [36] that relies on a feature tracking based algorithm. It builds a feature-based pose graph and then optimizes it by running it in 2 separate threads. One is the odometry thread, and the other is the mapping thread, which allows it to support large loop closing and global consistency. It achieves good localization with the use of featured visual odometry compared to the use of bundle adjustment, which is computationally very expensive. Unlike other algorithms like ORB-SLAM2, SOFT-SLAM algorithms are more deterministic (e.g., it results in the same output for the same dataset.)

Among the popular ones, we would like to mention the contribution of Large Scale Direct monocular SLAM (LSD-SLAM) [37]. LSD-SLAM has been one of the most popular SLAM techniques. While most the visual SLAM algorithms are based on features extracted from the images, the LSD-SLAM algorithm is a featureless algorithm, which allows us to build consistent large-scale maps of the environment in addition to tracking the motion of the camera. The reason behind this is that the features being used in most SLAM algorithms are completely dependent on the type of features being extracted, which in a larger complex environment can be different. In this algorithm,

the global map is represented as a pose graph, which consists of keyframes as vertices, and the 3D similarity transforms as edges. The classic LSD-SLAM mainly has a 3-step process: tracking, depth map estimation, and map optimization. The tracking section tracks new image frames, which allows estimating the rigid body frame with respect to the current keyframe. Depth estimation uses tracked frames to refine the current keyframe. The depth map generated after the depth map estimation block is fed into the global map using the map optimization component.

The above work of LSD-SLAM was also extended to stereo cameras [37]. It is based on almost the same technique, but the authors have exploited the use of a stereo camera setup as well. In essence, the depth estimation is done concurrently in 2 setups. One is from the stereo camera setup with a fixed baseline, and the other is from the multi-view stereo established from the camera motion. The advantage of having a stereo with a fixed baseline setup is that it avoids scale drift, which typically occurs in monocular LSD-SLAM. It also handles sudden illumination changes in the image frames using direct image alignment. This algorithm has been evaluated on the KITTI dataset, and it is still one of the most popular odometry estimation algorithms with an overall Root Mean Square Error (RMSE) of 1.21%.

Another stereo approach, given by [38], is based on the Exactly Sparse Delayed State Filter (ESDSF). This algorithm preserves the state space geometry by representing it as an algebraic Lie group. Since the approach is based on ESDSF, which is derived from the Extended Information Filter, the main advantage is that it uses a

sparse information matrix. One of the major features of this method is that it uses a novel ESDSF on a Lie group, which not only presents all the advantages of classical ESDSF but also holds the state space geometry using Lie groups. This algorithm was evaluated on the KITTI dataset and has been compared to various other SLAM algorithms like ORB-SLAM2, Stereo-Parallel Tracking and Mapping (S-PTAM), and Stereo LSD-SLAM (S-LSD SLAM). It has shown improvement in the odometry from most of the popular stereo based SLAM algorithms.

An approach by [39] presents an iterative 2-stage process for frame-to-frame feature-based odometry estimation. This algorithm attempts to analyze the characteristics of optical flows and re-projection errors that are generated from the 6-DOF motion. They have justified the re-projection error that we generated from the optical flow algorithm, which is dependent on the coordinate of the features.

One of the algorithms that use direct visual odometry was proposed by [24]. This algorithm attempts to solve the problem of getting stuck at local optima at large displacements. It is done by dual Jacobian optimization infusion with a multi-scale pyramidal scheme. In addition to this, it introduces new features based on gradients, which are robust to illumination changes. Finally, joint odometry is proposed to incorporate more information from the last frame to previous keyframes.

Stereo algorithms that track key points and select effective frames are accomplished by another technique called Selective SLAM (SSLAM) [40]. The basic idea in this

approach is that the error in localization or pose estimation is because of the uncertainty of 3D points. This uncertainty is higher for distant points. One key feature is that it does not require any loop closure or bundle adjustment. It uses a Harris corner detector to detect the features and Gradient Location and Orientation Histogram (GLOH) descriptor to match them.

Given the complexity of the graph-based SLAM, one approach that attempts to simplify the implementation of graph-based SLAM and also challenges the state-of-the-art SLAM algorithms is the ProSLAM [41]. It's main goal is to use the stereo images to generate a 3D map. It is evident that landmarks are essential since they allow for close detection of loop closures. In this approach, the node of the graph represents the pose (rotation and translation component), and the edge represents the spatial constraints. It is generated either by tracking the camera motion or by aligning local maps acquired at distant times, which also leads to loop closure that is again helpful for re-localization. The whole approach is taken into four steps: (1) Frame point generation, which takes the pair of stereo images and generates 3D points; (2) Position tracking, which estimates the relative motion of the camera from two subsequent image pairs acquired at different times as the camera moves; (3) Map management, which collects all the acquired map (3D point clouds) obtained from the trajectory and represents it in a compact form; (4) Re-localization, which compares each acquired point cloud from the previously acquired map and corrects the pose as well as the global map of the environment. This algorithm was evaluated on the KITTI dataset and has shown less than around 1% of translational error for most of the

sequences. It has outperformed popular SLAM techniques like S-PTAM [42, 43] and LSD-SLAM but is on a competitive level on the ORB-SLAM2 [5].

Most SLAM techniques face the challenge of correcting the scale drift from the acquired images. Research by [44], has attempted to resolve this issue. This work uses a monocular camera, and it estimates the ground plane using a novel cue combination framework. It achieves results comparable to stereo algorithms and could be carried out over long data sequences. From every monocular image, it uses sparse features that are acquired from the stereo images. At the same time, it performs object detection as well. It provides a model-learning approach from the training data, which is related to the covariances of the observation cues. Based on the KITTI dataset evaluation, this technique has outperformed VISO2 [45, 46] for both monocular and stereo cameras. The above method has attempted to build a bridge between monocular and stereo structure from motion in addition to correcting the scale drift. From the same researchers, another work that deserves attention was parallel visual odometry estimation [47]. The approach uses multi-threading for scenes with large motions and rapidly changing images. It uses three or more CPU parallel threads, and across all the threads, the system estimates the pose using 3D-2D correspondences, which is again followed by bundle adjustment.

Using features and descriptors has been one of the most popular approaches to performing localization. [48] uses feature detection algorithms like Oriented fast and Rotated Briefs (ORB) to detect the features from the image sequences and computes

feature descriptors using the Fast Retina Keypoint (FREAK) algorithm. This approach has been popularly known as the Circular Freak ORB (CFORB) algorithm. One of the key advantages it presents is that since it uses ORB features, it is invariant to both rotation and scale changes. It has also highlighted that it is invariant to the environment to uneven terrain changes. Another thing to highlight is that it uses two geometrical constraints in order to remove invalid geometrical feature matches, which was not implemented in the common visual odometry algorithm. On the KITTI benchmark dataset, this has shown an average translational error of 3.73% and a rotation error of 0.0107deg/rad. This approach has been tested in indoor environments as well, which performs slightly better than the outdoor environment, and in addition, it also performs well in the heavily textured environment.

2.3 Security in Autonomous Systems

Although autonomous vehicles, as projected, may lead to safer roads, reduce congestion and solve the parking problem, it still faces the challenge of security over the network. Since autonomous cars heavily rely on digital systems or computers, it is very likely to have communication protocols and frameworks employed in the system. At this point, there are two levels of networking among autonomous vehicles: Vehicle to Vehicle, Inter-vehicle networking around the vicinity of the vehicle in a local area, and Vehicle to Infrastructure - networking between vehicle and infrastructure system.

Sharing vital information among vehicles, like speed, could assist in efficient navigation on the roads. However, this system of complex architecture across wide networks can be attacked by various techniques, broadly classified into two categories: Active and Passive attacks [49].

Passive attacks are not intended to change the system's functionality; instead, it is done by a potential attacker to acquire confidential information about the system. A simple scenario could be eavesdropping, where an attacker could obtain information by intercepting the data traffic. Even if encryption is used, successful decryption can be counted as a passive attack. Another scenario could be to analyze the traffic signal, which allows the attacker to understand certain properties like information transaction based on duration, timing, bandwidth and number of participants in the traffic [49].

Active attacks are way more inclusive in terms of functionality and changing the system as well. There are various ways where a system could be hacked, like a man-in-the-middle attack, Denial of Service (DoS) attack and Replay attack. In a man-in-the-middle attack [50, 51], the attacker can fraudulently get access to the system. In addition to this, a man-in-the-middle attack sends an acknowledgement to the sender that it is an authorized user, thereby deceiving the whole system. In a replay attack, the attacker can observe the data traffic and replay a previous message which could force the system into an unstable state or could request further data for attacking at different levels of security. DoS means that the system has

been compromised, which could be done by disabling the communication services or jamming the communication channels.

With respect to autonomous cars, there have been demonstrations by researchers who have successfully attempted to gain complete control of the autonomous system, including disabling the brakes, stopping the engine, locking the doors etc., and most importantly, completely ignoring the driver input signals. Most autonomous cars follow the Controller Area Network (CAN)-protocol. However, this has presented various vulnerabilities in communication in the automotive industry. Primarily, CAN has a broadcast nature, and it sends packets to all the nodes in the network, which might allow the attacker to insert malicious components easily. CAN is also vulnerable to DoS attacks. In addition, CAN has no proper authentication mechanism. Anyone can send a packet to any node quite easily. It is quite important to mention that any attacker can attack an autonomous system easily, which makes it very essential for further research improvements in the autonomous driving industry [49, 52, 53].

2.4 Conclusion

This chapter has briefly described various SLAM techniques in relation to autonomous robots. We have attempted to classify the SLAM techniques in both Lidar-based odometry and Stereo-based odometry. In the end, we have also attempted to describe the security vulnerabilities in autonomous driving systems. We have described various

types of attacks that have been introduced and demonstrated by various researchers, which makes it a popular topic for future research.

Chapter 3

Correntropy: Concepts

3.1 Basics of Correntropy

This chapter is focused on the meaning of correntropy in geometrical and probabilistic terms. In general regression and adaptive filtering scenarios, the goal is to bring the system output as 'close' to the desired signal as possible. The 'closeness' is defined via distance function or similarity measure. It is well known that the Mean Squared Error (MSE) is normally employed as a cost function which can be represented as :

$$MSE(Y,Z) = E[(Y-Z)^2] = \int_y \int_z (y-z)^2 p_{YZ}(y,z)dydz = \int_e e^2 p_E(e)de \qquad (3.1)$$

where the desired signal is represented as $Z = \{z_i\}, i = 1, ..., N$ and the system output is represented as $Y = \{y_i\}, i = 1, ..., N$. One can see that the MSE is a

quadratic function in the joint space with a valley along $z = y$ line.

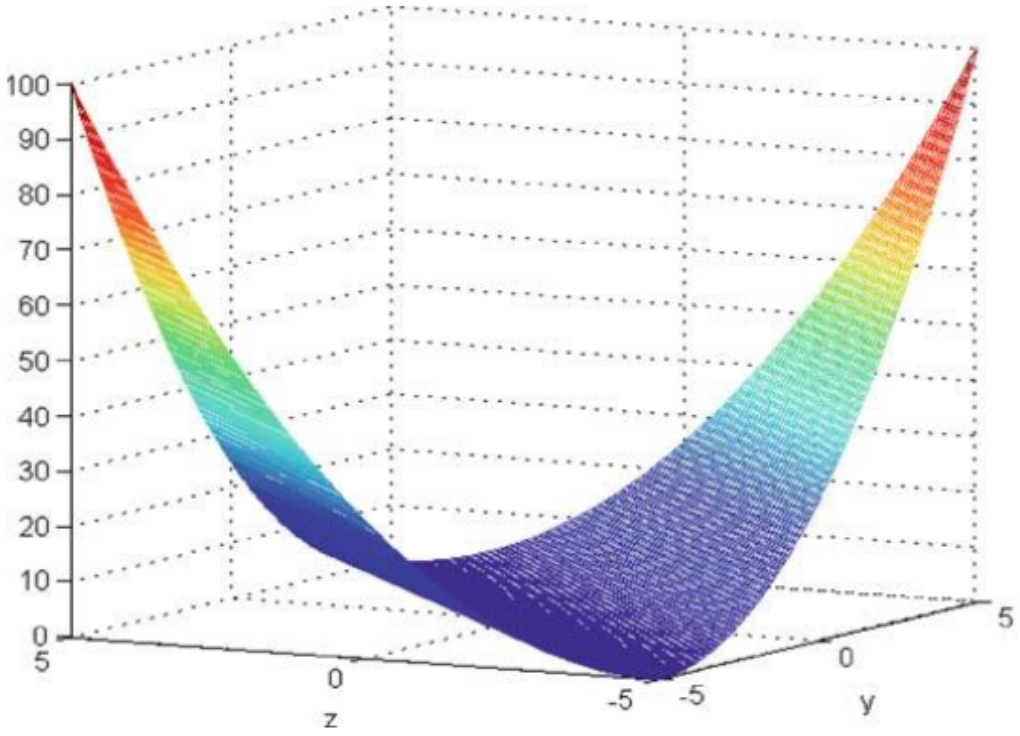


FIGURE 3.1: Joint space representation of the MSE.

From Fig. 3.1, one can clearly see the 'geometric' meaning of correntropy. One observation suggests that MSE is a quadratic function in the joint space with the valley along the $z = y$ line. In a probabilistic sense, similarity means a specific quantification of how close Z is to Y, which intuitively explains why MSE is a similarity measure in the joint space, as mentioned in Fig. 3.1. Equ. 3.1 signifies that the PDF of the error weighs the error square(e^2). It simply means that the values away from the $z = y$ line contribute quadratically to the error term. If the error PDF is Gaussian distributed, then MSE is optimal, which may not be the case for a lot of real-time applications. For samples that are far away from the mean value of the error distribution, the error

term amplifies because of the quadratic nature of the MSE. This makes MSE optimal for short-tail distribution (e.g., Gaussian). For other fat-tail data distribution like laplacian, the MSE is suboptimal.

Now , *cross-correntropy* or simply *correntropy* of 2 random variables Z and Y can be defined as:

$$v(Z,Y) = E_{ZY}[G_\sigma(Z-Y)] = \int\int G_\sigma(z-y)p(z,y)dzdy \tag{3.2}$$

If we are sampling from densities, then correntropy can be estimated as:

$$\hat{v}(Z,Y) = \frac{1}{N}\sum_{i=1}^{N} G_\sigma(z_i - y_i) = \frac{1}{N}\sum_{i=1}^{N} G_\sigma(e_i) \tag{3.3}$$

Fig. 3.2 represent the joint space representation of correntropy when $\sigma = 1$. One can observe that it exponentially attenuates contribution away from the line $z = y$.

From Equ. 3.3 one can define the cost function that maximizes the error probability density at the origin which can simply be written as Maximum Correntropy Criterion (MCC) algorithm.

$$MCC = \max_{w} \hat{v}(E). \tag{3.4}$$

Equ.3.4 shows the general format of correntropy where the parameter w control the error PDF $E = Z - Y$. If we use the Parzen method, the error PDF $p_E(e)$ can be

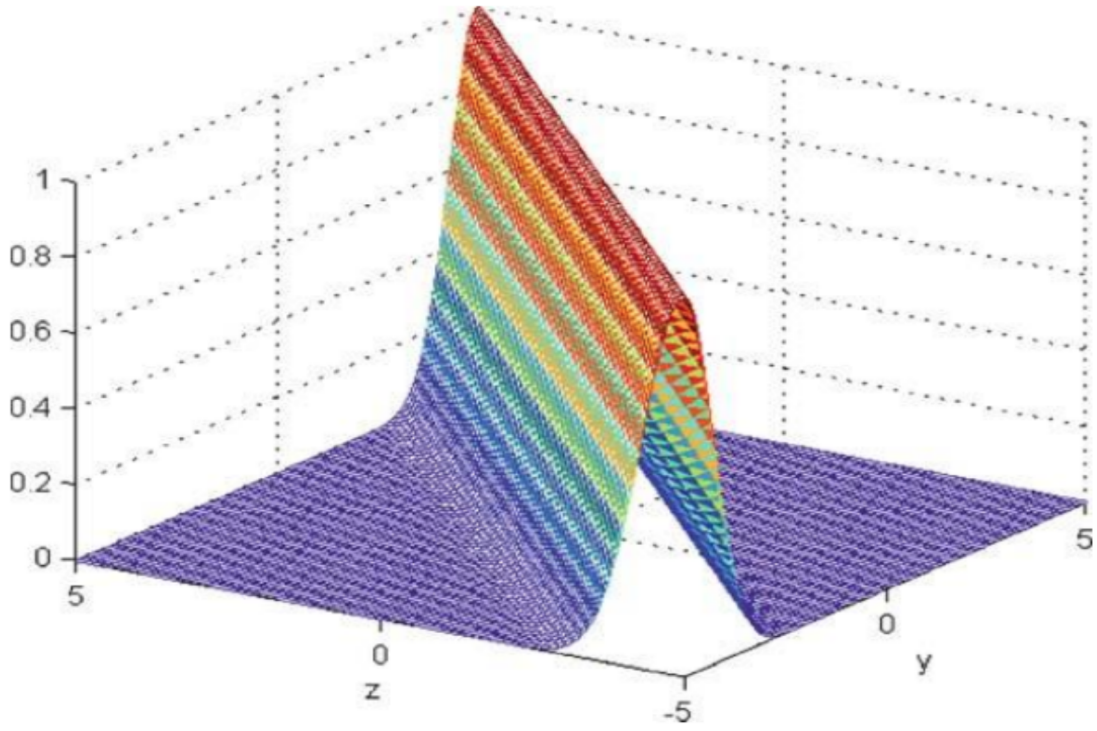


FIGURE 3.2: Joint space representation of Correntropy/Cross-correntropy.

estimated as

$$p_E(e) = \frac{1}{N}\sum_{i=1}^{N} G_\sigma(e - e_i). \tag{3.5}$$

Evaluating this PDF at $e = 0$, we obtain $\hat{v}(Z, Y) = p_E(0)$.

One can understand the overall idea of correntropy as the probability of how similar two random variables are in the neighbourhood of the joint space controlled by the kernel bandwidth, which can be interpreted as the spotlight controlling the 'observation window' in which the similarity is assessed. If the above definition is still unclear, one can think of the following example: Suppose you are given 2 sample biological cells of a living organism in a petri dish. You are asked to find out if the two cells are from the same organism or from different organisms. You have a microscope in your

lab with the potential to 'observe' these two living cells with great detail, and the microscope can be equipped with a lens of different magnification powers. Now, to find out if they are from the same organism, the 'difference' or 'distance' between the two cells in some metric must be small (or in some simple sense, if the 'difference' is less, they are very similar), else, the two cells are from different organisms. This 'distance' or 'difference' is dependent on two simple factors: the type of microscope and it's magnification power. Now proceeding with this understanding, we can say that the kernel function G_σ acts as a microscope and the σ denotes the magnification power of the lense (bandwidth in the earlier example). Hence the kernel bandwidth act as a spotlight controlling the 'observation window' (Note: Different kernel function(or microscope) can be used with different lenses(bandwidth parameter))

This idea has been heavily explored in the domain of Machine learning applications. Our work is focused on bringing this concept to the modern SLAM framework and evaluating how well correntropy can be used to resolve non-Gaussian noise.

3.2 Summary

This chapter describes the underlying concepts of correntropy and its comparison to the popular MSE. This foundation is essential for our future correntropy deployment in multiple SLAM applications. We have attempted to briefly understand correntropy and highlighted its importance in dealing with non-Gaussian error distributions.

Chapter 4

Correntropy Kalman Filter

4.1 Motivation

In this chapter, we discuss the state of the art of state estimation in the presence of non-Gaussian noise and how the Kalman filter handles this kind of noise. We show through numerical simulations that the Kalman filter does not perform well, for example, in the presence of shot noise. We then present the concept of maximum correntropy (MCC) and how to use it inside a KF. The MCC is an approach to measure the similarity between two random variables using information from higher-order signal statistics. The new formulation of the KF dealing with non-Gaussian noise is tested using a simulated dataset to prove that the MCC-KF can better handle non-Gaussian noise.

This paper is organized as follows. Section 4.2 reviews the basic Kalman Filter based on the weighted least square method. Section 4.3 introduces the concept of correntropy and describes the maximum correntropy criterion. Section 4.4 demonstrates the simulation results in both the basic KF method and the MCC-KF method. Finally, the chapter is concluded in Section 4.5.

4.2 Kalman Filter Based on WLS method

Consider the linear stochastic discrete-time system

$$\begin{aligned} x_k &= Fx_{k-1} + w_k \\ y_k &= Hx_k + v_k \end{aligned} \tag{4.1}$$

where

- $x_k \in \mathbb{R}^n$ is the state vector.
- $y_k \in \mathbb{R}^m$ is the measurements (observations) vector.
- w_k is the zero-mean process, with covariance matrices $Q_k = E[w_k w_k^T]$.
- v_k is the measurement noise, with covariance matrix $R_k = E[v_k v_k^T]$.
- F is the state transition model which is applied to the previous state x_{k-1}.
- H is the observation model which maps the true state space into the observed space.

There are several methods that could be employed to derive the KF equation, such as the orthogonality principle, the Maximum A Posterior (MAP) approach, the Unbiased Minimum Variance (UMV) approach, and the Weighted Least Square (WLS) method. In WLS, the quadratic objective function is

$$J = \tfrac{1}{2}(y_k - H\hat{x}_k)^T R_k^{-1}(y_k - H\hat{x}_k)+ \\ \tfrac{1}{2}(\hat{x_k} - F\hat{x}_{k-1})^T P_{k|k-1}^{-1}(\hat{x}_k - F\hat{x}_{k-1}) \tag{4.2}$$

where

- $P_{k|k-1} = E[e_k^- e_k^{-T}]$.
- $e_k^- = x_k - \hat{x}_k^-$.
- $R_k = E[v_k v_k^T]$.
- $Q_k = E[w_k w_k^T]$.

The KF equations presented above can be derived by solving

$$\frac{\partial J}{\partial \hat{x}_k} = 0. \tag{4.3}$$

Then the KF is written as

$$\hat{x_0} = E[x_0], \tag{4.4}$$

$$P_0 = E[e_0 e_0^T], \tag{4.5}$$

$$\hat{x}_k^- = F\hat{x}_{k-1}, \tag{4.6}$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k, \tag{4.7}$$

$$K_k = (P_{k|k-1} + H^T R_k^{-1} H)^{-1} H^T R_k^{-1}, \tag{4.8}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - H\hat{x}_k^-), \tag{4.9}$$

$$P_{k|k} = (I - K_k H)P_{k|k-1}(I - K_k H)^T + K_k R_k K_k^T. \tag{4.10}$$

where

- $e_0 = x_0 - \hat{x}_0$, for k=1,2,... .
- $E(\cdot)$ is the expected value operation
- $K_k \in \mathbb{R}^{n\times m}$ is the Kalman gain.
- $\hat{x}_k^-$ is the priori estimate of the state x_k. It is based on measurements up to and including time $k-1$, and has covariance $P_{k|k-1}$.
- $\hat{x}_k$ is the posteriori estimate of the state x_k. It is based on measurements up to and including time k, and has covariance $P_{k|k}$.

4.3 Maximum Correntropy Criterion

Correntropy is generally defined as the similarity measure of two random variables. Given two random variables $X, Y \in R$ having joint distribution function $f_{xy}(x, y)$,

the correntropy is defined as:

$$C_\sigma(X,Y) = E[k_\sigma(X,Y)] = \int\int k_\sigma(x,y) f_{xy}(x,y) dxdy \tag{4.11}$$

where

- $E[.]$ denotes the expectation operator.
- $k_\sigma(.,.)$ is positive definite kernel function.
- $f_{xy}(.,.)$ is the joint density functions of X and Y.

For a finite number of data points N available in practical applications, the estimator can be computed as:

$$\hat{C}_\sigma(X,Y) = \frac{1}{N}\sum_{i=1}^{N} k_\sigma(x_i, y_i). \tag{4.12}$$

In our experiments we use a Gaussian kernel, which can be written as:

$$\hat{C}_\sigma(X,Y) = \frac{1}{N}\sum_{i=1}^{N} G_\sigma(x_i - y_i), \tag{4.13}$$

where

$$G_\sigma(x_i - y_i) = exp(-\frac{\|x_i - y_i\|^2}{2\sigma^2}), \tag{4.14}$$

with the bandwidth σ.

The Maximum Correntropy is realized if the Gaussian Correntropy function is positive, bounded and reaches its maximum if and only if $X = Y$.

It can be understood from Figure 6.2.

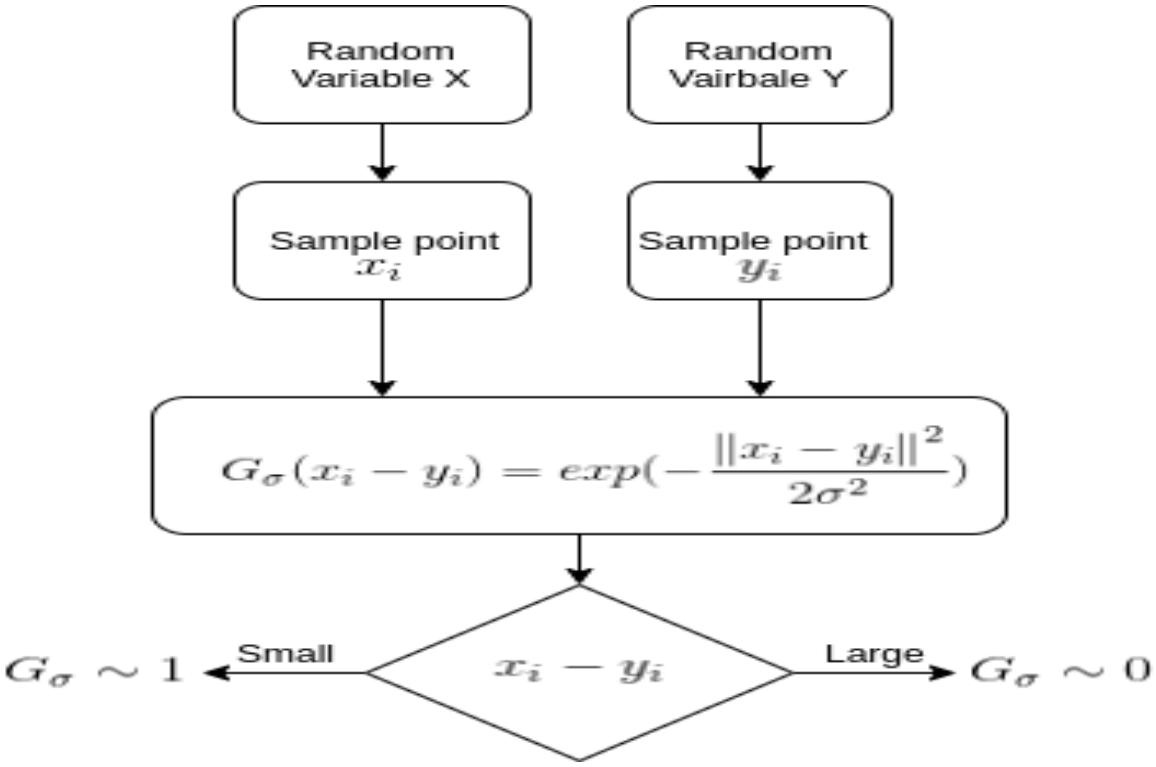


FIGURE 4.1: Correntropy Criterion.

To introduce the kernel inside the standard KF formulation, the objective function in eq. (4.2) can be re-written as:

$$J = G_\sigma(||y_k - H\hat{x}_k||_{R_k^{-1}}) + G_\sigma(||\hat{x}_k - F\hat{x}_{k-1}||_{P_{k|k-1}^{-1}}). \tag{4.15}$$

as derived in [54] and [55].

Then the equations for the MCC-KF can be written as follows:

$$\hat{x_0} = E[x_0], \tag{4.16}$$

$$P_0 = E[e_0 e_0^T], \tag{4.17}$$

$$\hat{x}_k^- = F\hat{x}_{k-1}, \tag{4.18}$$

$$P_{k|k-1} = FP_{k-1|k-1}F^T + Q_k. \tag{4.19}$$

The remaining state estimation can be calculated as:

$$L_k = \frac{G_\sigma(||y_k - H\hat{x}_k||_{R_k^{-1}})}{G_\sigma(||\hat{x}_k - F\hat{x}_{k-1}||_{P_{k|k-1}^{-1}})}, \tag{4.20}$$

$$K_k = (P_{k|k-1}^{-1} + L_k H^T R_k^{-1} H)^{-1} L_K H^T R_K^{-1}, \tag{4.21}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - H\hat{x}_k^-), \tag{4.22}$$

$$P_{k|k} = (I - K_k H)P_{k|k-1}(I - K_k H)^T + K_k R_k K_k^T. \tag{4.23}$$

From the derivation of L_k, it can be seen that if y_k appears to be too big (outlier measurement), G_σ approaches zero, and so is L_k. If this is the case, L_k tends to zero and the Kalman Gain, K_k, becomes zero. This means that the state update in eq.

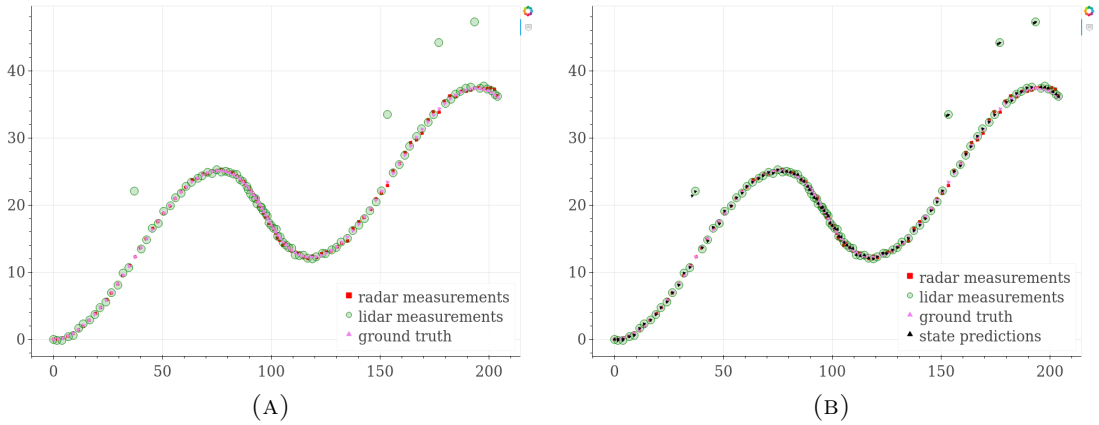
(4.22) is updated by the state of the system as given by eq. (4.18). The mathematical representation of the described formulations can be written as:

$$\lim_{y_k \to \infty} L_k = 0, \tag{4.24}$$

$$\lim_{L_k \to 0} K_k = 0, \tag{4.25}$$

$$\lim_{K_k \to 0} \hat{x}_k^- = \hat{x}_k^- . \tag{4.26}$$

4.4 Simulation Results



(A) (B)

FIGURE 4.2: (a) Lidar Measurements with shot noises. (b) KF response to shot noises (labeled as state predictions)

To conduct our study, we implement both a Kalman Filter (KF) and a Maximum Correntropy Criterion Kalman Filter (MCC-KF) on a simulated dataset. We use lidar and radar measurements collected by a robot navigating in a predefined trajectory.

The dataset we use to evaluate our experiment is https://github.com/mithi/fusion-ekf/tree/master/data. Detailed information about data can be found in the introduction section mentioned in https://github.com/mithi/fusion-ekf. However, we are focusing our approach on the Kalman Filter, as mentioned in the equation in Section 4.3. The goal is to fuse the data coming from the lidar and from the radar, using the KF to estimate the position of the robot with respect to a fixed coordinate frame system. For the purpose of our simulation, we can say that we are dealing with two noisy sensors. In particular, lidar computes the position in a cartesian coordinate system (x, y), and the radar evaluates the position and the relative velocity in a polar coordinate system (ρ, ϕ, θ). In our simulation, we add shot noise at random times in the lidar measurements only. Then we compare the response of the standard KF to shot noises as well as the response of the MCC-KF to the shot noises. Our framework of evaluation is shown in Figure 4.3.

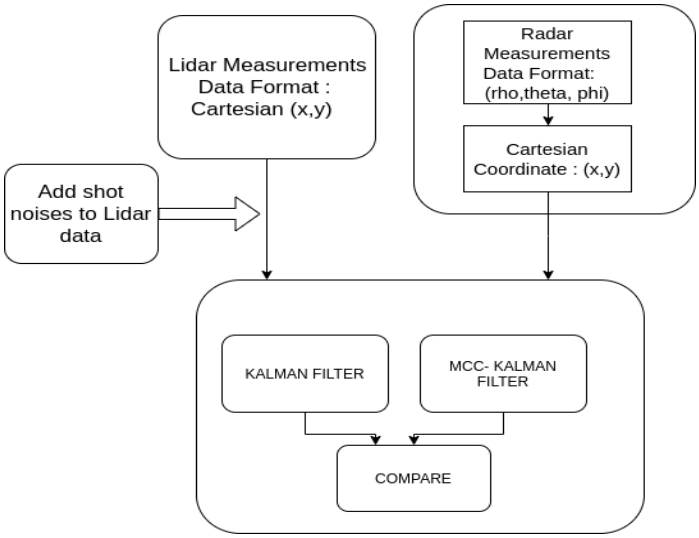


FIGURE 4.3: Framework of our evaluation.

In short, we describe our approach in the following steps:

- Read the 2D Lidar measurements in the cartesian coordinates (x, y) from the dataset file.
- Read the Radar Measurements in the Polar coordinates from the data file. Convert the polar coordinate data to the corresponding cartesian coordinate.
- Introduce shot noises at random locations in the Lidar data.
- Implement Kalman Filter and evaluate the response of the Kalman filter to the shot noises.
- Implement MCC Kalman Filter and evaluate the response of the MCC Kalman filter to the shot noises.

For this dataset we have the following state equations:

- State Vector

$$\begin{aligned} x_k &= Fx_{k-1} + w_k, \\ y_k &= Hx_k + v_k, \end{aligned} \tag{4.27}$$

where

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.28}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

(4.29)

v_k is the process noise with covariance R_k where, for Lidar measurments R_k is:

$$R_k = \begin{bmatrix} 0.0225 & 0 \\ 0 & 0.0225 \end{bmatrix}$$

(4.30)

and for Radar measurements R_k is:

$$R_k = \begin{bmatrix} 0.009 & 0.0 & 0.0 \\ 0.0 & 0.09 & 0 \\ 0 & 0 & 0.09 \end{bmatrix}$$

(4.31)

- The process covariance matrix Q_k is :

$$Q_k = \begin{bmatrix} 2.25 & 0 & 4.5 & 0 \\ 0 & 2.25 & 0 & 4.5 \\ 4.5 & 0 & 9 & 0 \\ 0 & 4.5 & 0 & 9 \end{bmatrix} \tag{4.32}$$

We compare the estimation from a KF to the MCC-KF to predict both the position of and the velocity of the travelling robot, given the noisy sensor data.

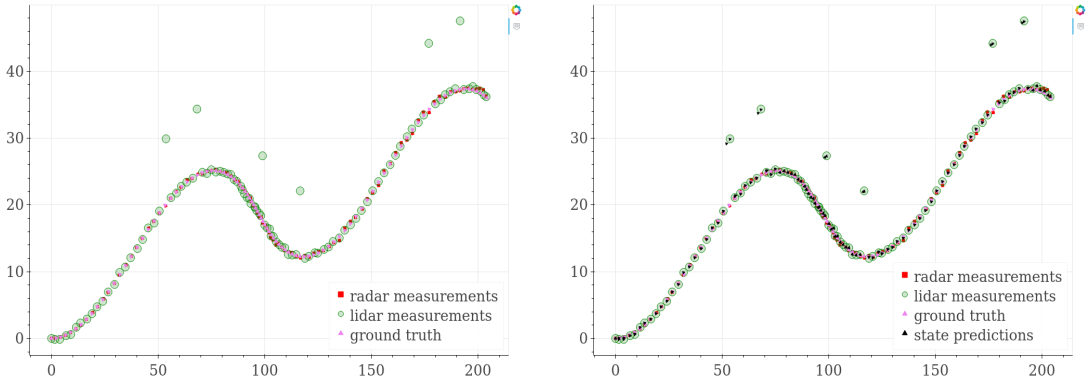
The data is read from a file having the following format:

- L(lidar): $x\ y\ t\ gt_x\ gt_y\ gt_vx\ gt_vy$.
- R(radar): $\rho\ \phi\ \theta\ t$.

where:

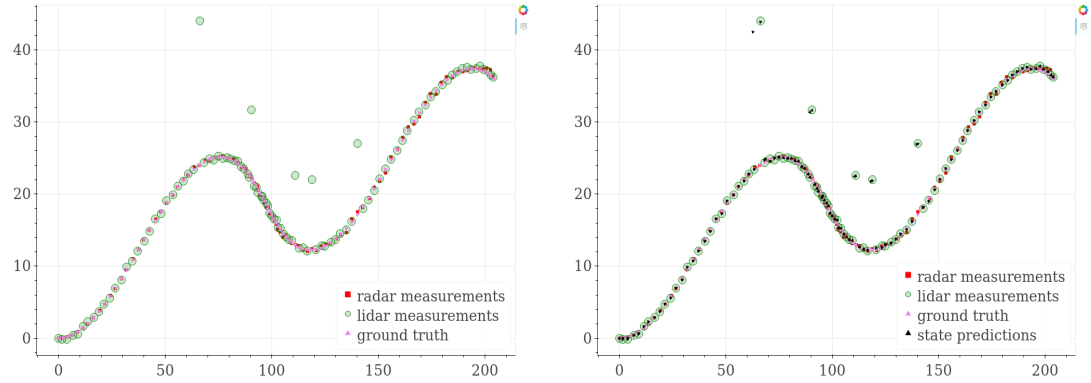
- (x, y) is a measurement given by the lidar.
- (ρ, ϕ, θ) is a measurement given by the radar in polar coordinates.
- (t) is the timestamp associated to a measurement.
- $(gt_x, gt_y, gt_vx, gt_vy)$ is the real ground truth state of the system.

In Figure 4.2, 4.4, 4.5, 4.6 and 4.7 (a) we can observe that after introducing shot noise in the Lidar measurements at random locations the KF fails drastically. The lidar measurements (green circles) are intentionally randomly disturbed. In Figure 4.2, 4.4,



(A) (B)

FIGURE 4.4: (a) Lidar Measurements with shot noises. (b) KF response to shot noises (labeled as state predictions)

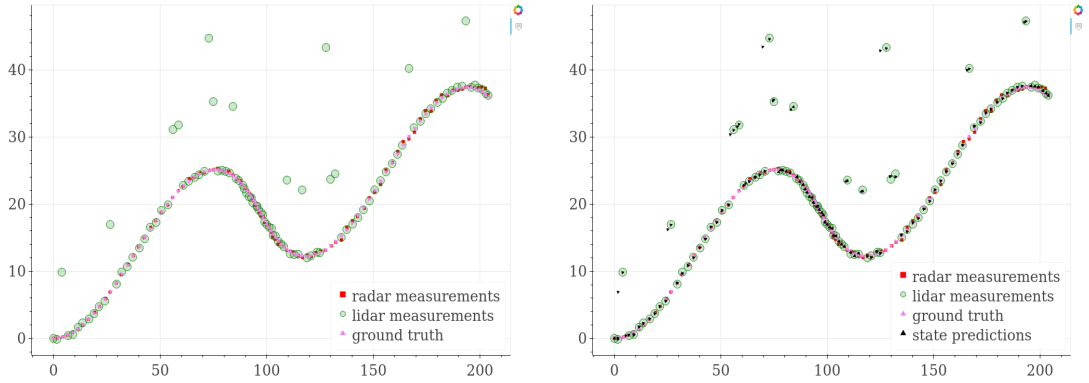


(A) (B)

FIGURE 4.5: (a) Lidar Measurements with shot noises. (b) KF response to shot noises (labeled as state predictions)

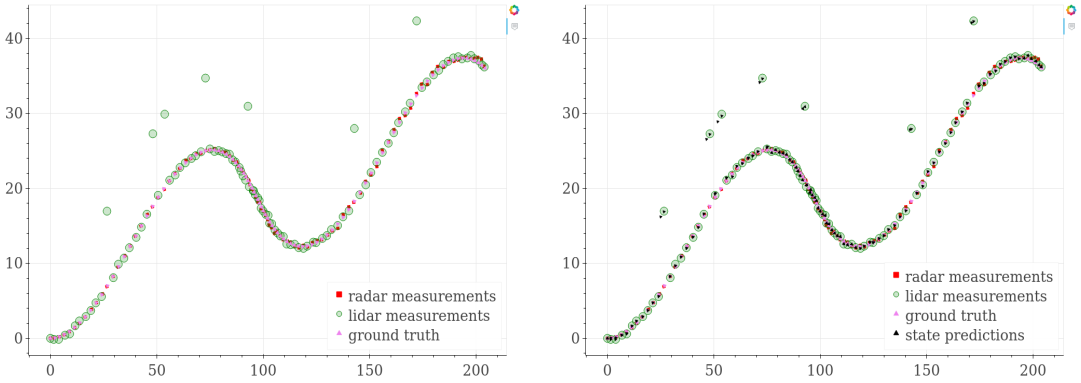
4.5, 4.6 and 4.7 (b), it is possible to observe the comparison between the estimated trajectory (violet triangles), lidar measurements (green circles), radar measurements (yellow squares) and ground truth (black triangles) in the presence of the above described shot noise. Note that all the plots are illustrated in position (m) versus time (s).

We introduce random shot noises using the *rand*() function in $C++$. For all the



(A) (B)

FIGURE 4.6: (a) Lidar Measurements with shot noises. (b) KF response to shot noises (labeled as state predictions)



(A) (B)

FIGURE 4.7: (a) Lidar Measurements with shot noises. (b) KF response to shot noises (labeled as state predictions)

simulations, we picked different random locations within the range of 100 sample data points, and, within a range of 20, different random values were inserted at these locations.

The RMSE results for the KF without any noise are shown in Table 4.1. The RMSE values for the KF with simulated noise input are shown in Table 4.2. The RMSE values for the MCC-KF with simulated noise input are shown in table 4.3. It can

be clearly seen that the KF fails in the presence of shot noise. Gaussian correntropy function can be used to solve this problem.

Table 4.1: Standard KF RMSE error with no noise

	RMSE Errors
RMSE for x	0.185913
RMSE for y	0.190259

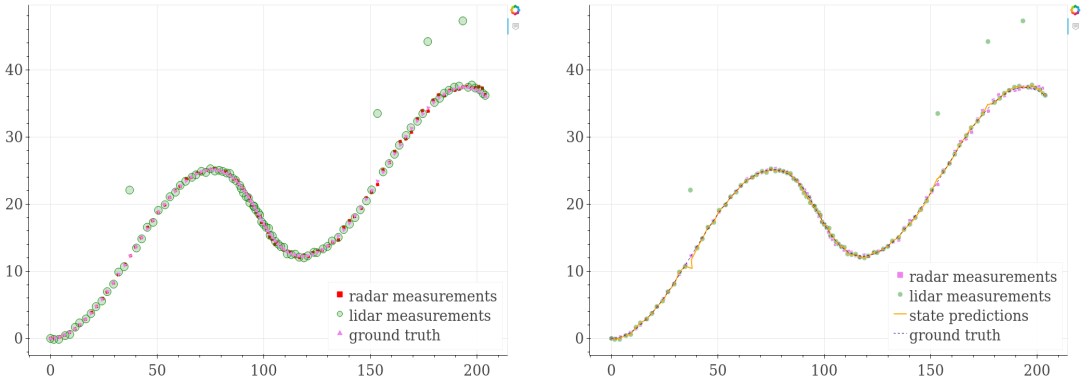
Table 4.2: KF RMSE error with noise

	RMSE Error in x	RMSE Error in y
Simulation 1 (Fig. 4.2)	0.252935	1.95544
Simulation 2 (Fig. 4.4)	0.261795	2.38964
Simulation 3 (Fig. 4.5)	0.319903	2.71349
Simulation 4 (Fig. 4.6)	0.468542	4.89599
Simulation 5 (Fig. 4.7)	0.30235	2.52797

Table 4.3: MCC-RMSE error with noise

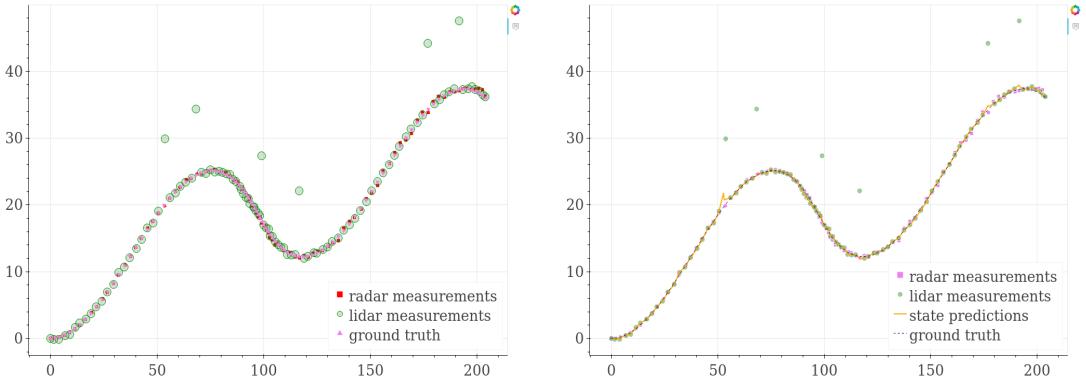
	RMSE Error in x	RMSE Error in y
Simulation 1 (Fig. 4.8)	0.1890	0.2414
Simulation 2 (Fig. 4.9)	0.1933	0.2509
Simulation 3 (Fig. 4.10)	0.1900	0.2689
Simulation 4 (Fig. 4.11)	0.2049	0.2738
Simulation 5 (Fig. 4.12)	0.1905	0.2321

Comparing the results obtained in both cases, it can be noticed that the estimation is more robust and reliable in the case of the MCC-KF.



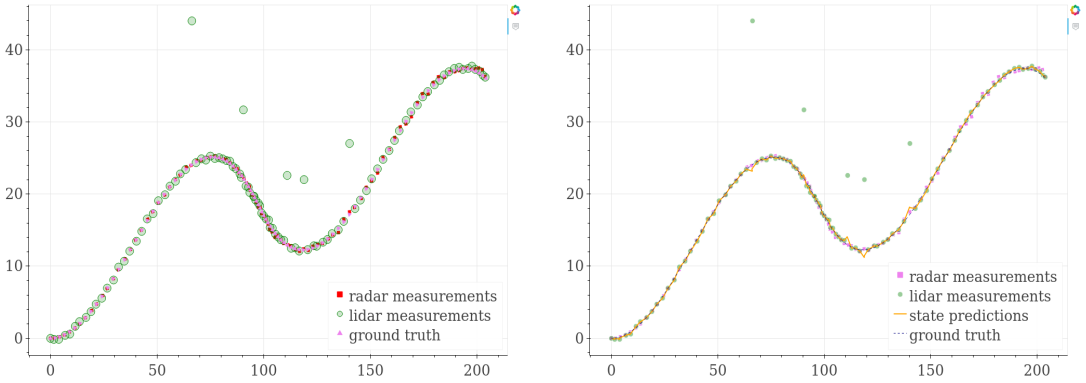
(A) (B)

FIGURE 4.8: (a) Lidar Measurements with shot noises. (b) MCC-KF response to shot noises (labeled as state predictions)



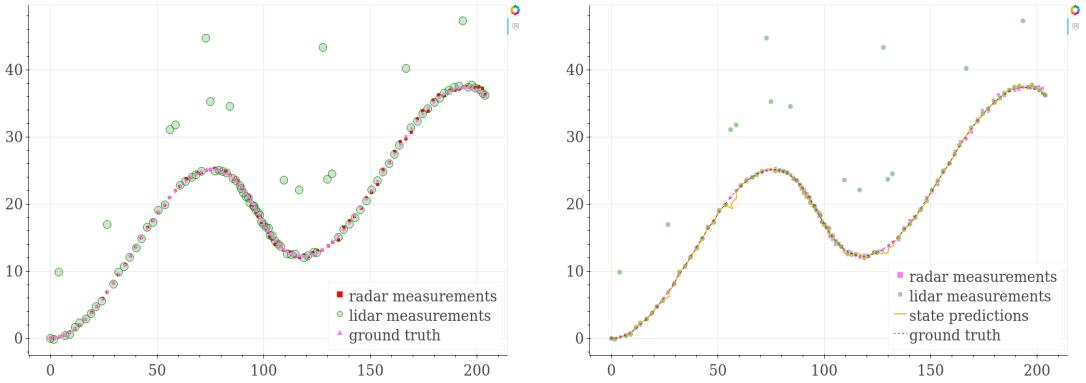
(A) (B)

FIGURE 4.9: (a) Lidar Measurements with shot noises. (b) MCC-KF response to shot noises (labeled as state predictions)



(A) (B)

FIGURE 4.10: (a) Lidar Measurements with shot noises. (b) MCC-KF response to shot noises (labeled as state predictions)



(A) (B)

FIGURE 4.11: (a) Lidar Measurements with shot noises. (b) MCC-KF response to shot noises (labeled as state predictions)

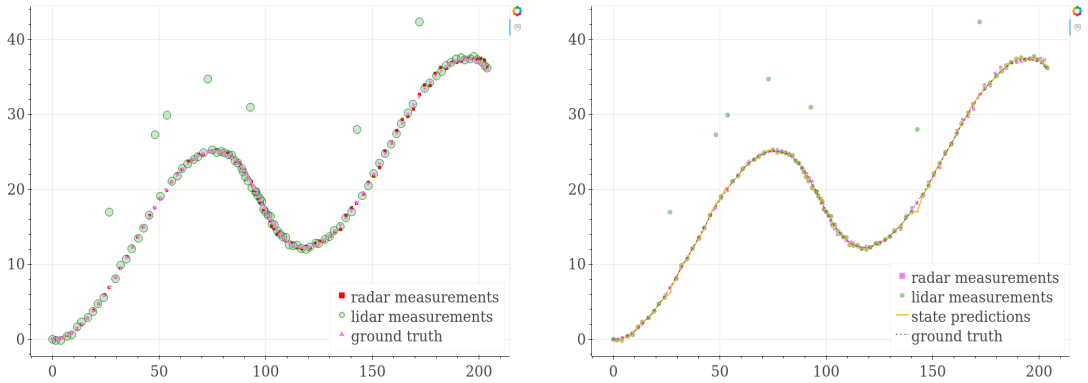


FIGURE 4.12: (a) Lidar Measurements with shot noises. (b) MCC-KF response to shot noises (labeled as state predictions)

4.5 Conclusion

In this chapter, we extensively reviewed and discussed the KF to better understand how it works and how it can be modified to take into account higher-order signal statistics. We exposed the main implementation and ran a different simulation to prove that when dealing with non-Gaussian noise, it is necessary to introduce a new element inside the filter called correntropy. This addition lets us use higher-order statistics to improve state estimation. In a preliminary phase, we run a total of five simulations in which, each time, we introduce a random shot noise to the lidar measurements to prove that the standard formulation of the KF does not behave as well in the presence of non-Gaussian noise. After deriving the MCC-KF equations, multiple simulations give more statistical value to our results. Finally, in this paper, we demonstrate that, in the presence of non-Gaussian noise, the utilized MCC-KF handles better this type of noise if compared to the standard KF, resulting in better and more robust state estimation.

Chapter 5

MCC-EKF for Autonomous Car Security

5.1 Motivation

Despite the flood of numerous SLAM algorithms that have been proposed so far, very few of them address the problem of securing the autonomous system in case it gets attacked. Numerous incidents have been reported where researchers have attacked the autonomous car systems (for example, Tesla) that made the car change it's naturally estimated trajectory [56]. This has raised a serious concern in modern autonomous systems and needs to be addressed [50, 51, 53]. Although cyber-security experts have proposed various solutions to solving those issues, these security/hacking problems are still an open challenge. It is important to note that any system can potentially

be attacked. Now, if a SLAM system is vulnerable, it becomes a challenge to deploy it in real-time scenarios. So, can a SLAM system be secured by itself? Which means can a SLAM system be designed in a way where it can detect an attack/outliers by itself to avoid the change in it's natural estimated trajectory?

This chapter is focused on addressing the SLAM security problem by building a self-secure SLAM framework that can detect potential outliers/attacks. We propose to use the Maximum Correntropy Criterion - Extended Kalman Filter (MCC-EKF) approach in our framework and show through the results how our approach avoids attacks. The framework that we build is based on using the odometry from two SLAM methods and fusing them using MCC-EKF, which results in the overall estimation of the autonomous system. This also answers the question - Can we use two odometry from different SLAM methods and get a better estimate of the trajectory? The results show that we can improve the overall trajectory. The prime motivation for our work is to resolve a situation when the system gets attacked that forces the autonomous system to change it's naturally estimated trajectory [49] [55]. In several examples, we have seen how an attacker can attack an autonomous system and change its trajectory to the attacker's own desired location [56]. We propose to use the MCC-EKF SLAM algorithm in our system, which has the potential to give a solution to this problem. We use this approach in our system as well as evaluate our approach in the popular KITTI dataset [57] and discuss the results.

5.2 Proposed Methodology

Figure 5.1 gives the basic system architecture of our proposed approach. As mentioned in Section 1, we use two different SLAM algorithms to get the odometry. We also use two different sensors to accomplish our goal. We use a simulated Velodyne Lidar (VLP-32) and a simulated stereo camera in our setup. We have evaluated our system in a gazebo simulated environment from an open-source repository [58]. We attempt to evaluate how the odometry information from the two SLAM methods [59, 60] can be used to enhance the overall odometry of the autonomous system (e.g., autonomous car).

For calculating the Lidar odometry, we use the Iterative Closest Point (ICP) SLAM, which uses 3D Lidar data for estimating the 6 DOF position of the Lidar sensor. For our purpose, we are using only the raw Lidar measurements. In this approach, at first, the Lidar data is downsampled to reduce computational complexity. We use a voxel grid filter of size 0.2 to downsample the point clouds. Later on, it estimates the sensor pose using the ICP, which is applied iteratively in consecutive frames. We use the point-to-plane error metric for faster convergence. Every estimated pose is then fed to a pose graph approach to optimize the relative pose as well as to detect the loop closure. The results of this approach are shown in the Results section. For the ICP parameters, we have set the maximum iterations to 100, and the maximum corresponding ratio is set to 0.01.

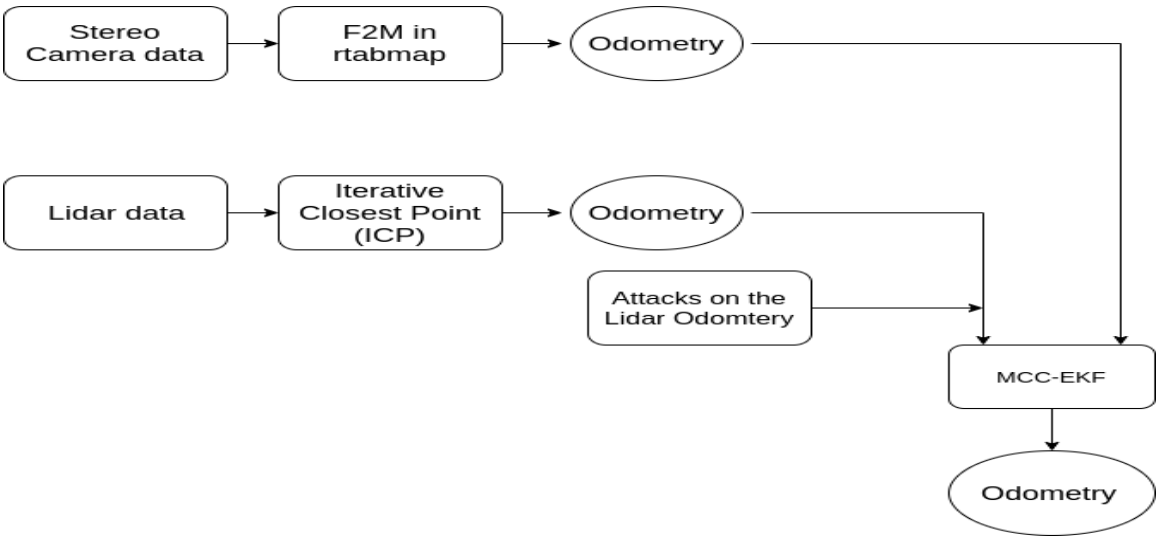


FIGURE 5.1: System Architecture.

Using the stereo camera data, we calculate the odometry using RTAB-Map Frame-To-Map (F2M) strategy [59].

Our approach takes advantage of these two SLAM methods, and then we feed their respective calculated odometry in our MCC-EKF framework. Here we also propose to introduce shot noises or attack vectors into the estimated Lidar odometry and see how our framework responds to the attacks. Let the attack vector at time k be $\mathbf{a}_k$. So the measurement equation is updated as $\mathbf{y}_k^{'} = \mathbf{y}_k + \mathbf{a}_k$. Then Equ. (4.20) gets updated as:

$$\mathbf{L}_k = \frac{G_\sigma(||\mathbf{y}_k^{'} - \mathbf{H}\hat{\mathbf{x}}_k||_{\mathbf{R}_k^{-1}})}{G_\sigma(||\hat{\mathbf{x}}_k - \mathbf{F}\hat{\mathbf{x}}_{k-1}||_{\mathbf{P}_{k|k-1}^{-1}})}. \tag{5.1}$$

So when $\mathbf{y}_k^{'}$ is large as defined by the kernel bandwidth, $\mathbf{L}_k$ is 0, which forces the Kalman Gain $\mathbf{K}_k$ to be 0. So the next state is updated by the system state as given in Equ. (4.18). This process, as one can see, can reject the attacks/outliers, thus securing the system.

The MCC-EKF algorithm (Algorithm 1) for our approach is given below.

Algorithm 1 MCC-EKF algorithm for autonomous system security.

Computed odometry from MCC-EKF SLAM Lidar Odom (ICP) $\rightarrow \mathbf{L}_o(x, y, z, r, p, y)$
Stereo Odom (F2M) $\rightarrow \mathbf{S}_o(x, y, z, r, p, y)$
Initialization;
Lidar odom $\rightarrow$ ($\mathbf{L}_o$)
Stereo odom $\rightarrow$ ($\mathbf{S}_o$)
Compute $\mathbf{x}_0$ from Equ. (4.16)
Compute $\mathbf{P}_0$ from Equ. (4.17)
Prior Estimation Compute $\hat{\mathbf{x}}_k^-$ from Equ. (4.18)
Compute $\mathbf{P}_{k|k-1}$ from Equ. (4.19)

while get $\mathbf{L}_o$ and $\mathbf{S}_o$ **do** Compute $\mathbf{L}_k$ from Equ. (5.1)
Compute Gain $\mathbf{K}_k$ from Equ. (4.21)
Update state $\mathbf{x}_k$ from Equ. (4.22)
Update $\mathbf{P}_{k|k}$ from Equ. (4.23)

5.3 Results

We evaluate our approach on two systems - the simulated gazebo system and the KITTI dataset. The Gazebo simulated system uses a Prius model with inbuilt Lidar and camera system. The source code and the model description are available as open-source [58]. Figure 5.2 shows the simulated gazebo environment for our work. We have edited the Prius car model in order to include a stereo system so that we can get the odometry from the stereo camera (using RTAB-Map FrameToMap (F2M)).

Figure 5.3 shows one sequence of the data from the KITTI dataset. Again, it is mentioned earlier that we are using two different SLAM algorithms - ICP for Lidar odometry calculation and FrameToMap Visual odometry calculation (from RTAB-Map) for the stereo camera. We are injecting attacks on the Lidar odometry and

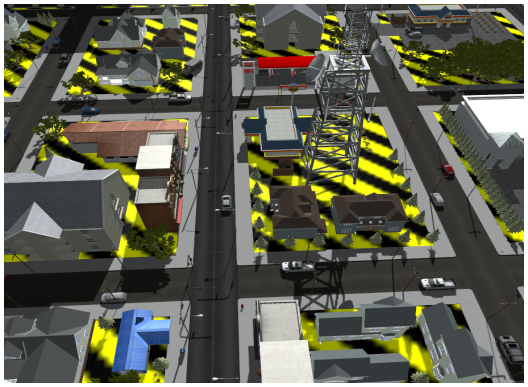
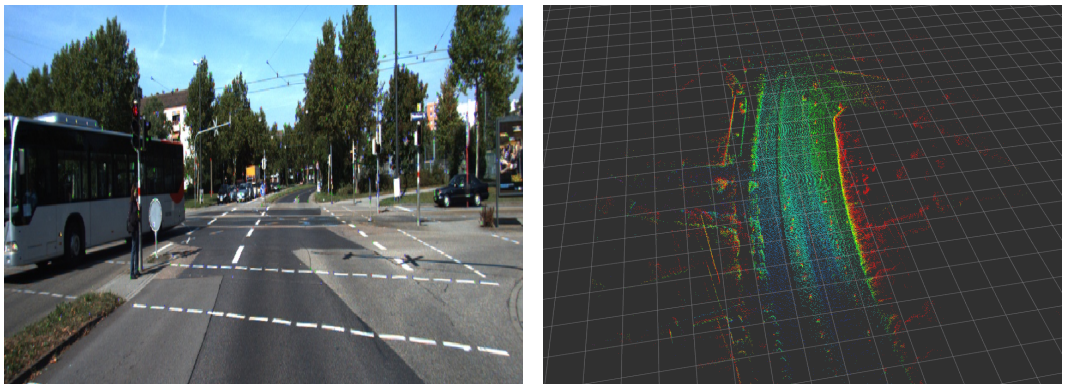


FIGURE 5.2: Gazebo Simulation Environment.

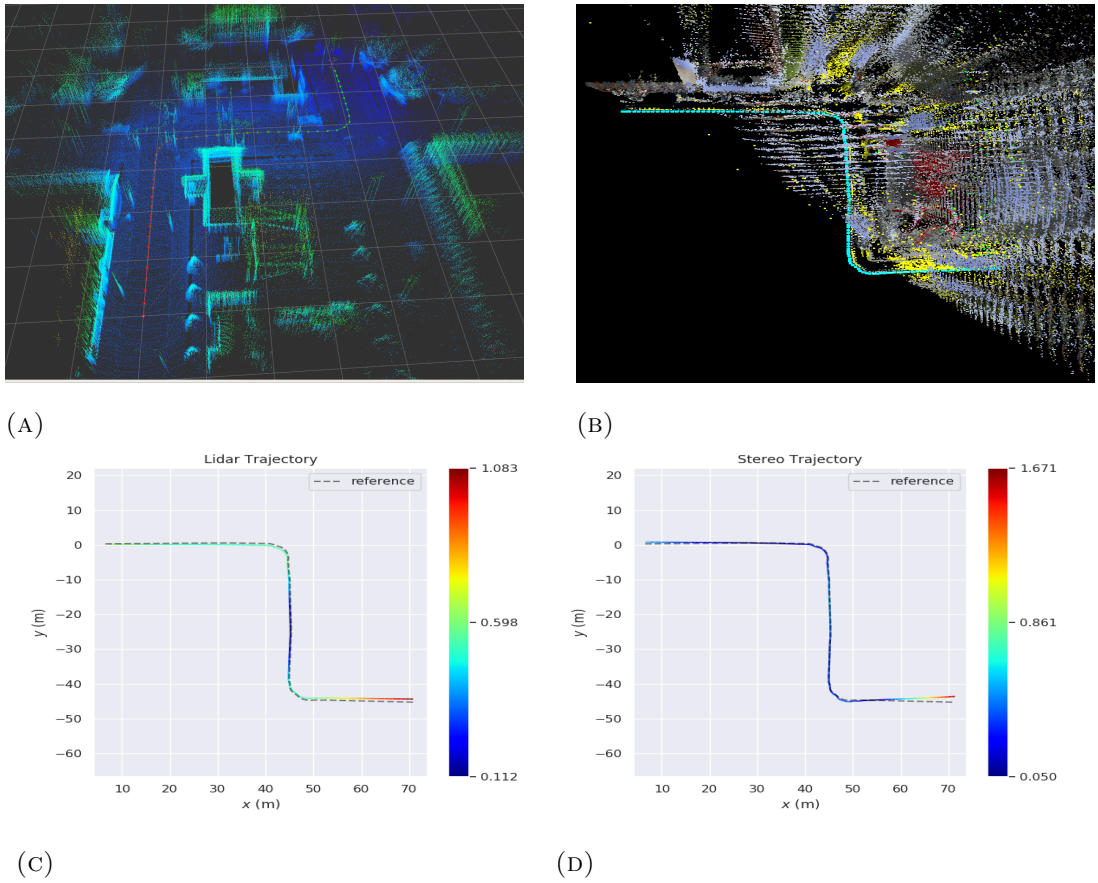
evaluating how the MCC-EKF responds. Generally, for our framework, we can use any two SLAM algorithms to output odometry.



(A) (B)

FIGURE 5.3: KITTI dataset data (SEQ 11): (a)- Environment, and (b) its 3D Lidar map.

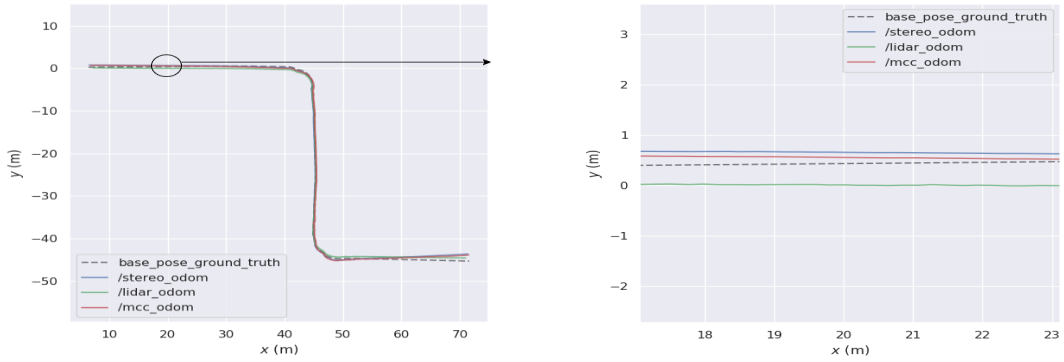
For the Lidar odometry, we use ICP based approach to retrieve the odometry from Lidar. Initially, we downsample the point cloud using voxel grid filtering, and we use the Point-to-Plane error metric for faster convergence of the ICP algorithm. The sample result from our gazebo simulated model is shown in Figure 5.4. Figure 5.4(a) shows the Lidar odometry and the map. Figure 5.4(c) shows the Lidar trajectory with respect to the ground truth. The dotted line is the ground truth.



(A) (B)

(C) (D)

FIGURE 5.4: (a, c) Lidar odometry, and (b, d) Stereo odometry (dotted path shows the ground truth.)



(A) (B)

FIGURE 5.5: (a) Odometry trajectory of each method. (b) Zoom-in at one location.

Using the stereo camera, we calculate another set of odometry using Frame2Map in RTAB-Map. The sample result of the odometry as well as the mapping is shown in

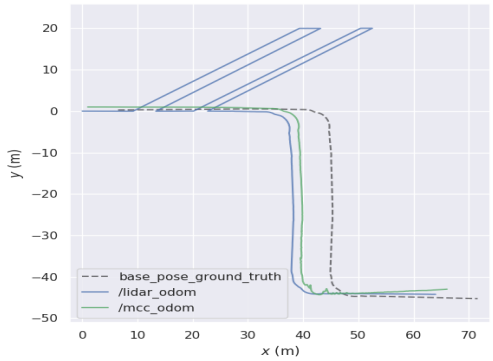
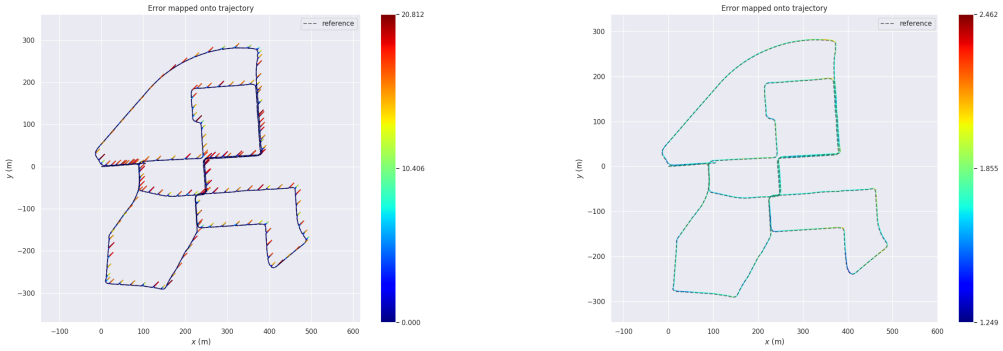


FIGURE 5.6: MCC-EKF response to attacks on Lidar data.



(A) (B)

FIGURE 5.7: Evaluation on KITTI sequence number 27. Here, (a) Normal EKF response to attacks in Lidar data, RMSE: 10.406; (b) MCC-EKF reponse to attacks, RMSE: 1.85.

Figure 5.4(b) and Figure 5.4(d)

The next step involves using the odometries obtained from the above-mentioned methods in our MCC-EKF framework. The combined odometries (Lidar odometry, Stereo odometry and the MCC-EKF odometry) for the example shown in Figure 5.4 is shown in Figure 5.5. In Table 1, this trajectory is referred to as Trajectory 1.

Our initial query, as mentioned earlier, was, can we improve the odometry by using these two odometries obtained from different SLAM algorithms? In our experiment

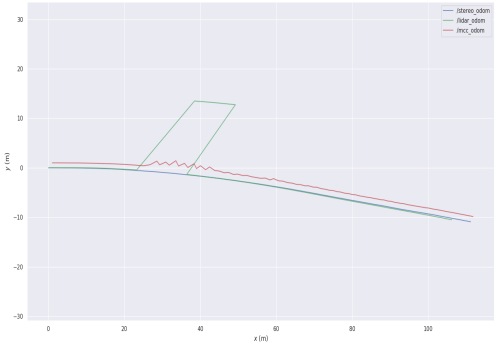


FIGURE 5.8: EKF response on KITTI dataset SEQ 01 when attack vector is introduced.

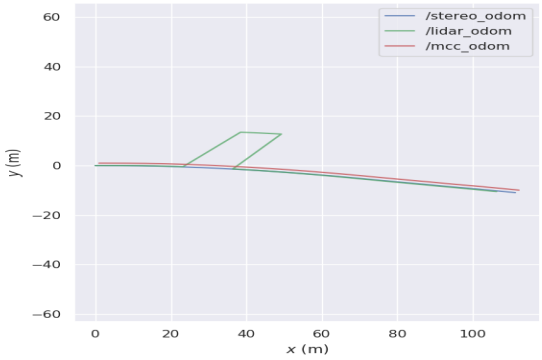


FIGURE 5.9: MCC-EKF response on KITTI dataset SEQ 01 when attack vector is introduced.

(Gazebo simulation), we compare the Root Mean Square Error (RMSE) values of individual trajectories with respect to the ground truth. Table 5.1 compares the RMSE values, which clearly shows that MCC-EKF performs better than the individual SLAM algorithms.

Another problem that we intended to solve was to avoid attacks (false injection of odometry values). This was the primary reason for our work presented here. We inject false values in the lidar odometry and test the response of our MCC-EKF approach. Table 5.2 indicates the RMSE values of the algorithms when false odometry is injected.

We inject constant false values at a random location in the lidar odometry, and we see that MCC-EKF can handle those attacks as compared to the use of the traditional EKF algorithm. The sigma parameter of the Gaussian kernel function G_σ plays an important role in MCC-EKF implementation. For our experiment, it is set to 10. Figure 5.6 shows the response of the MCC-EKF to the attacks on the Lidar data. In Figure 5.6, it is important to note that the trajectories are translated so that all the paths are displayed clearly. One can see that MCC-EKF does not get affected at places where the attacks were introduced.

TABLE 5.1: RMSE comparison

	Traj 1	Traj 2
Stereo-SLAM (RMSE)	0.861	1.840472
Lidar-SLAM (RMSE)	0.598076	0.505731
MCC-EKF-SLAM (RMSE)	0.419823	0.49761

TABLE 5.2: RMSE comparison after attack vectors

	Traj 1	Traj 2
Random attacks 1		
Normal EKF (RMSE)	3.624596	5.01472
MCC-EKF (RMSE)	0.420437	0.81037
Random attacks 2		
Normal EKF (RMSE)	4.624596	7.01472
MCC-EKF (RMSE)	0.43317	0.85132

As it is clear, MCC-EKF is a variant of the traditional EKF, and the MCC-EKF is robust to attacks or sudden outliers. So if we introduce attacks in the system (attacking Lidar odometry), the trajectory changes drastically. However, using the MCC-EKF approach, the attacks are rejected.

We have evaluated the approach on the KITTI dataset. For the KITTI dataset, we have shown our results in two sequences. We have compared the results of MCC-EKF with normal EKF with and without attacks. The comparison of both the systems is shown in Figure 5.8 and 5.9. Figure 5.8 shows the normal EKF response with attacks, while Figure 5.9 shows the MCC-EKF response with attacks. We can clearly see that the MCC-EKF can successfully eliminate the attacks, but the EKF can not. The source code is available on our ARA lab GitHub: `https://github.com/aralab-unr/MCC-EKF-SLAM`

Table 5.3: RMSE comparison on KITTI dataset.

Random Attacks 1				
	SEQ 01	SEQ 05	SEQ 11	SEQ 27
EKF	2.04	2.42	1.475	3.193625
MCC-EKF	0.0198	0.0073	0.110	0.142083
Random Attacks 2				
EKF	3.79	7.0685	2.1875	10.406
MCC-EKF	0.0198	0.092263	0.1842	1.85

5.4 Conclusions

In this chapter, we have attempted to provide a self-secure solution to an autonomous system using the MCC-EKF approach. From the results, we have shown how an autonomous system can be attacked by an attacker/hacker and change the system's naturally estimated trajectory and how even a simple injection of false positions can affect the overall trajectory of the autonomous system. We have also shown how the MCC-EKF approach can resolve the issue of sudden attacks/outliers to the system. In

addition, we have also proposed that we can also get a better estimate of the odometry by fusing the odometry data from two different SLAM algorithms to obtain a better odometry estimate of the autonomous system.

Our future work will focus on proposing a solution where we can secure the system if the attacker chooses to inject false data on the sensor's raw measurements. We also plan to extend this work to distributed MCC-EKF security for vehicle to vehicle network in which multi-robot system research [2, 61–75] can be utilized.

Chapter 6

Correntropy Registration

6.1 Motivation

This chapter focuses on the Registration or Alignment of 3D point sets. Although the Registration problem is a well-established problem, and it is solved using multiple variants of the Iterative Closest Point (ICP) Algorithm, most of the approaches in the current state-of-the-art still suffer from misalignment when the *Source* and the *Target* point sets are separated by large rotations and translation. In this work, we propose a variant of the Standard ICP algorithm, where we introduce a Correntropy Relationship Matrix in the computation of the rotation and translation component, which attempts to solve the significant rotation and translation problem between *Source* and *Target* point sets. This matrix is computed through the correntropy criterion, which is updated in every iteration. The correntropy criterion defined in

this approach maintains the relationship between the points in the *Source* dataset and the *Target* dataset. Through our experiments and validation, we verify that our approach has performed well under various rotations and translations compared to the other well-known state-of-the-art methods available in the Point Cloud Library (PCL) and other methods available as open source.

Registering two point clouds is an active area of research in the community. Registration is the process of aligning two point cloud datasets of the same scene, which are separated by a certain transformation (rotation and translation). The registration problem can be explained as finding the transformation between two point clouds such that the relative error between the 2 point sets is minimized. The problem of registration is a significant component in Simultaneous Localization and Mapping (SLAM) for robotics and other visual data matching algorithms [76]. The relative transformation computed from point clouds acquired at close time intervals from a 3D sensor can be used to compute the overall trajectory of a system [1, 76, 77].

Early approaches for 3D point cloud registration can be attributed to Besl and MacKay [78, 79] and Arun et.al [78], which consists of computing the centroid for both *Source* and *Target* datasets. Each point from both the datasets is subtracted from their respective centroids, and later on, Singular Value Decomposition (SVD) is used to compute the rotation component. This procedure is performed iteratively until the relative mean square error between the point cloud datasets is minimized,

and hence it is termed the Iterative Closest Point (ICP) Algorithm. Since then, variants of ICP have been introduced, and a point-to-plane variant of ICP operates on taking advantage of surface normal information [80]. Instead of minimizing the root mean square of the individual points as done in the generalized version of the ICP, the point to plane Algorithm minimizes error along the surface normal [19, 31, 81–85].

Despite the various advances in the registration methodologies, handling the situation where the data set is noisy is still a significant concern. If the noise is Gaussian, we can remove most of it from the data. However, when the noise is non-Gaussian (e.g., shot noises), it becomes difficult to resolve. Hence, it is essential to address the problem of handling non-Gaussian noise. The idea of Correntropy has seen widespread use in modern machine learning, signal processing, and other related fields [86, 87]. Correntropy is a measure of similarity between two random variables and has been implemented with the traditional Kalman filter [55]. Some recent work [87–90] has introduced the Correntropy concept to the registration problem to overcome the drawbacks of the ICP algorithm. In the traditional ICP algorithm, if any of the point cloud dataset (*Source* or *Target*) is affected by non-Gaussian noise, the ICP fails drastically and affects the overall transformation estimation of the point clouds. In this work, we show basic examples of how the addition of non-Gaussian noise can affect the traditional ICP and compare how our approach can be better than the traditional ICP algorithm and its variants. We also compare our approach with the well known Normal Distribution Transform (NDT) algorithm available in the PCL library.

In this work, we propose a Correntropy Similarity Matrix Iterative Closest Point (CoSM ICP) algorithm. In our approach, we introduce a matrix of size $N \times N$ which maintains a 'numeric' relationship between *Source* and *Target* dataset points (size of *Source* dataset and *Target* dataset is N, hence the matrix size is $N \times N$). This numeric relationship is calculated through the well-known correntropy criterion, and it is updated at every iteration. It is calculated only between the nearest points from *Source* dataset to the *Target* dataset, thereby making the Similarity Matrix a sparse matrix. Our approach not only provides a significant improvement in the alignment of *Source* and *Target* dataset under various rotations and translations, but it also proves quite effective against non-Gaussian outliers affecting the dataset. In addition to evaluating our approach in multiple datasets, we also evaluate the method where we inject the *Source* dataset with random false values. Through experimental comparison with the traditional ICP and other approaches, we evaluate how well our approach can estimate the transformation between the 'infected' *Source* and the *Target* dataset, and we see how our approach performs better than the well-known state-of-the-art methods. Alongside estimating accurate transformations, handling shot-noise or the non-Gaussian noise vectors is a critical component for better transformation estimation and is the driving force for our work.

Inspired by this Correntropy concept, we fuse it with the ICP algorithm, called Correntropy Similarity Matrix ICP or CoSM-ICP. Our main contribution to this work can be summarized as follows:

- We propose a new data registration algorithm in which the Correntropy Similarity Matrix ICP is introduced.
- We evaluate our approach on various randomly generated transformation matrices between *Source* and the *Target*.
- Through evaluation on these random transformation matrices, we see how well our approach performs better than the other well-known state-of-the-art methods.
- We also introduce outliers in the *Source* dataset and evaluate our approach on the 'infected' *Source* dataset.
- We compare our approach with other approaches on the 'infected' *Source* dataset and see through the results how well our approach performs better than the other state-of-the-art methods.

6.2 Correntropy Criterion

We begin by describing the mathematical representation of point cloud data and develop our Algorithm based on it. We denote *Source* point cloud as $\mathbf{P_s}$ and *Target* point cloud as $\mathbf{P_t}$. The points in the point cloud $\mathbf{P_s}$ contains N points in which each point can be referenced as $\mathbf{p}_j = \{x_j, y_j, z_j\}$, where x_j, y_j, z_j denotes the 3D coordinates of the point $\mathbf{p}_j$. Similarly, the points in the point cloud $\mathbf{P_t}$ contains N points in which each point can be represented as $\mathbf{q}_k = \{x_k, y_k, z_k\}$, where x_k, y_k, z_k

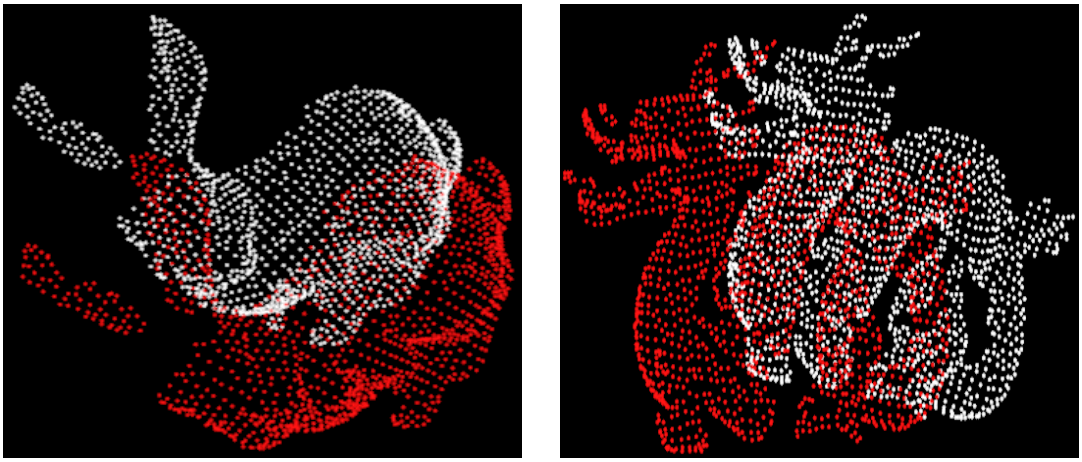
TABLE 6.1: Nomenclature

Variables	Usages
$\mathbf{P_s}$	*Source* Point Cloud Set.
$\mathbf{P_t}$	*Target* Point Cloud Set.
N	Total number of points in point clouds $\mathbf{P_s}$ and $\mathbf{P_t}$.
j, k	Index of every point in point cloud where $j, k = 1, .., N$.
$\mathbf{p}_j$	Individual 3D point (x_j, y_j, z_j) in a point cloud $\mathbf{P_s}$, or $\mathbf{p}_j \in \mathbf{P}_s$.
$\mathbf{q}_k$	Individual 3D point (x_k, y_k, z_k) in a point cloud $\mathbf{P_t}$, or $\mathbf{q}_k \in \mathbf{P}_t$.
G_σ	Gaussian kernel function with bandwidth σ.
$\mathbf{R}$	Rotation matrix.
$\mathbf{tr}$	Translation vector.
ε^2	Root mean square error.
s	Scaling factor.
$\mathbf{SM}$	Similarity Matrix.
$\mathbf{s_{cen}}$,$\mathbf{t_{cen}}$	Centroids computation of *Source* and *Target* datasets.

denotes the 3D coordinates of the point $\mathbf{q}_k$. Throughout the paper, we describe a matrix as bold and capitalized letters, a vector is represented as bold and lowercase letters and the rest of the variables (single dimension) are represented as lowercase variables. Figure 8.2 shows a brief description of the point clouds, and Table 6.1 denotes the mathematical notations used throughout this work.

We use the Correntropy concept between the *Source* point cloud and the *Target* point cloud. Correntropy has proved beneficial in removing large outliers [86]. Correntropy is essentially a similarity measure of two random variables. We use the Correntropy concept between *Source* and *Target* point clouds. In this scenario, we are measuring the similarity between points $\mathbf{p}_j$ and $\mathbf{q}_k$ from the point clouds $\mathbf{P_s}$ and $\mathbf{P_t}$, respectively. Then the Correntropy criterion between points $\mathbf{p}_j$ and $\mathbf{q}_k$ can be mathematically represented as :

$$V_\sigma(\mathbf{p}_j, \mathbf{q}_k) = E[\kappa_\sigma(\mathbf{p}_j - \mathbf{q}_k)]. \tag{6.1}$$



(A) (B)

FIGURE 6.1: (a) Red is the *Source* point cloud $\mathbf{P_s}$ (bunny rabbit). $\mathbf{P_s}$ contains N points $\mathbf{p}_j$ $(j = 1, .., N)$, each of which is a 3D representation x_j, y_j, z_j. White is the *Target* point cloud $\mathbf{P_t}$. $\mathbf{P_t}$ contains N points $\mathbf{q_k}$ $(k = 1, .., N)$, each of which is a 3D representation x_k, y_k, z_k. Same goes for the dragon dataset in (b).

Equ. (6.1) also refers to in common literature as a cross-Correntropy of two random variables [3, 54]. In Equ. (6.1), *E[.]* refers to the expectation of the variable, and κ_σ denotes the kernel function. We can manually choose the size of the kernel function (also referred to as bandwidth σ). In our approach, we use the Gaussian kernel function where we redefine the Correntropy function as:

$$V_\sigma(\mathbf{p}_j, \mathbf{q}_k) = \frac{1}{N} \sum_{j,k=1}^{N} G_\sigma(\mathbf{p}_j - \mathbf{q}_k). \tag{6.2}$$

From Equ.(6.2),

$$G_\sigma(\mathbf{p}_j - \mathbf{q}_k) = exp(-\frac{\|\mathbf{p}_j - \mathbf{q}_k\|^2}{2\sigma^2}), \tag{6.3}$$

where σ is the bandwidth or the kernel size of the Gaussian kernel. From Equ. (6.3) it becomes evident that if $\mathbf{p}_j = \mathbf{q}_k$ Gaussian Correntropy is maximum, and the Gaussian

Correntropy function is positive and bounded [3, 54].

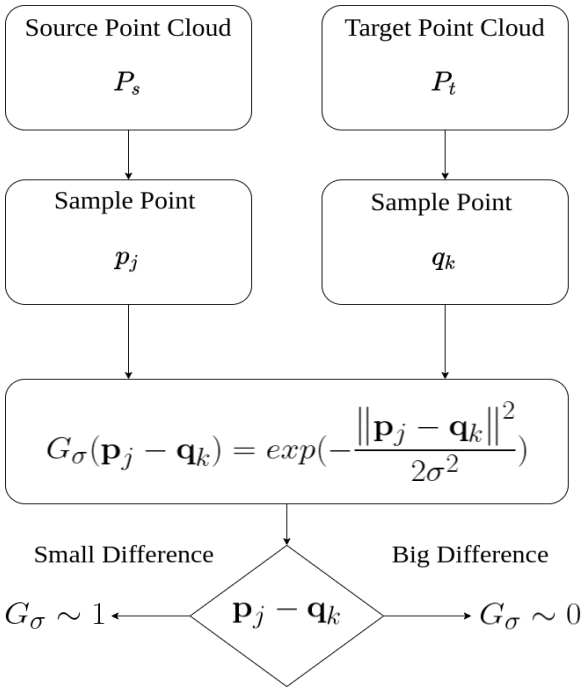


FIGURE 6.2: Correntropy Criterion. Here, when the difference between the points $(\mathbf{p}_j - \mathbf{q}_k)$ is small, i..e., when they are similar, and the Gaussian kernel function G_σ approaches to 1. On the contrary, when the difference is large, the Gaussian kernel function approaches to 0.

Fig. 6.2 shows the basic understanding of the Correntropy criterion. We see that when the difference between the points $(\mathbf{p}_j - \mathbf{q}_k)$ is small, *i.e.,* they are similar, and the Gaussian kernel function G_σ approaches to 1. On the contrary, when the difference is large, the Gaussian kernel function approaches 0. It essentially means that when a potentially changed data point (e.g., $\mathbf{q}_k$) is far varied from $\mathbf{p}_j$, the Gaussian kernel signifies that the points $\mathbf{p}_j$ and $\mathbf{q}_k$ are not similar. Intuitively, this similarity is dependent on the value of σ. This technique allows us to reject the faulty data point (shot noise or non-Gaussian noise) $\mathbf{q}_k$. This idea of calculating

the maximum similarity is given in Equ. (6.3) is called the maximum Correntropy criterion.

When the value of $\mathbf{q}_k$ approaches ∞ then the kernel function approaches 0, i.e.,

$$\lim_{\mathbf{q}_k \to \infty} G_\sigma = 0. \tag{6.4}$$

Intuitively, when the Gaussian Kernel shrinks to 0, the Correntropy approaches the value $p(\mathbf{p}_j = \mathbf{q}_k)$ [91].

6.3 Proposed Method

6.3.1 Correntropy Similarity Matrix with Iterative Closest Point Algorithm

Given the general description of the Correntropy concept in the previous section, we extend this idea of Correntropy to the well known ICP algorithm. Traditional ICP [78] describes the alignment of point clouds such that the Mean Square Error (MSE) between point sets is minimized. The MSE is the key criterion of convergence in ICP and its variants. ICP revolves around the problem of aligning point sets $\mathbf{P_s}$ and $\mathbf{P_t}$. We denote the point sets as $\mathbf{P}_s = \left\{[\mathbf{p}_j]_{j=1}^{N}\right\}$ (here $\{\}$ denotes a point set) and $\mathbf{P}_t = \left\{[\mathbf{q}_k]_{k=1}^{N}\right\}$, respectively. Our goal is to best align *Source* point set $\mathbf{P_s}$ to *Target* point set (or a model point set) $\mathbf{P_t}$. This leads to finding the appropriate rotation and translation between the two point sets, such that the Root Mean Square

Error (RMSE) error between the two point set is minimized (or is within a defined threshold). This process is commonly known as registration between the *Source* $\mathbf{P_s}$ and the *Target* $\mathbf{P_t}$.

Now, the problem can be written as: find the best $\mathbf{R}$ and $\mathbf{tr}$ such that the MSE, ε^2, between two point clouds ($\mathbf{P_s}$ and $\mathbf{P_t}$) is minimized. This problem can be mathematically written as,

$$\varepsilon^2 = \sum_{j,k=1}^{N} \|\mathbf{p}_j - (s\mathbf{R}\mathbf{q}_k + \mathbf{tr})\|^2 , \tag{6.5}$$

where s is the scaling component. $\mathbf{p}_j \in \mathbf{P_s}$ is the set of all the points in the *Source* point set, and $\mathbf{q}_k \in \mathbf{P}_t$ is the set of all the points in the *Target* point set ($j, k = 1, ..., N$).

Our work is built on the work of Arun et al.[78], and it is well known from their work that initially they compute the corresponding points between *Source* and *Target* point sets. The Corresponding points in this work are computed as shortest point from each point in the *Source* dataset to the *Target* dataset. This can be efficiently done using k-d tree data structure. We introduce an additional parameter using the Correntropy criterion to find the similarity metric between points. Now, we say that for a *Source* point index i the corresponding point index in the *Target* point set is $\mathbf{c}(i)$ ($\mathbf{c}$ is vector of corresponding points from *Source* to *Target*. Note: We do not perform reciprocal correspondence i.e., from *Target* to *Source*). From Equ.6.3 we compute the similarity

between the two corresponding points $\mathbf{P_s}$ and $\mathbf{P_t}$ as

$$\begin{aligned} \mathbf{d} &= \mathbf{P_s}(i) - \mathbf{P_t}(\mathbf{c}(i)), \\ G_\sigma(\mathbf{P_s}(i) - \mathbf{P_t}(\mathbf{c}(i))) &= \frac{1}{(2\pi\sigma)^{\frac{1}{D}}} exp(\frac{\mathbf{d}^T\mathbf{d}}{2\sigma^2}). \end{aligned} \tag{6.6}$$

Here, D is the dimensions and $D = 3$ in this case [92]. Now, we can create a similarity matrix between the *Source*, and the *Target* points based on the similarity metric computed in Equ. 6.6. Intuitively, the size of the similarity matrix is $N \times N$. In every iteration, we initialize the value of this similarity matrix as zeros and update them in every iteration based on the similarity metric computed above in Equ.6.6. It is updated as shown below:

$$\begin{aligned} \mathbf{SM}(i, \mathbf{c}(i)) &= G_\sigma(\mathbf{P_s}(i) - \mathbf{P_t}(\mathbf{c}(i))), \\ \mathbf{SM}(\mathbf{c}(i), i) &= G_\sigma(\mathbf{P_s}(i) - \mathbf{P_t}(\mathbf{c}(i))), \end{aligned} \tag{6.7}$$

where **SM** is the similarity matrix which we intend to use in the computation of the rotation component. The centroid of both the point clouds i.e., the *Source* and *Target* is computed as given by Equ.6.8

$$\begin{aligned} \mathbf{s_{cen}} &= \frac{1}{N}\sum_{i=1}^{N} \mathbf{p}_i, \\ \mathbf{t_{cen}} &= \frac{1}{N}\sum_{i=1}^{N} \mathbf{q}_i, \end{aligned} \tag{6.8}$$

where $\mathbf{s_{cen}}$ and $\mathbf{t_{cen}}$ are the computed centroid of *Source* and *Target* data set respectively. The difference between the computed centroid $\mathbf{s_{cen}}$ and the individual *Source* points is computed. The same procedure is followed for the *Target* points. It is given in Equ. 6.9

$$\begin{aligned} \mathbf{p}_i^{'} &= \mathbf{p}_i - \mathbf{s_{cen}}, \\ \mathbf{q}_j^{'} &= \mathbf{q}_j - \mathbf{t_{cen}}. \end{aligned} \tag{6.9}$$

The Singular Value decomposition (SVD) for finding the rotation component is based on first finding the 4×4 matrix as shown in Equ. 6.10 where we introduce our Similarity Matrix:

$$\mathbf{H} = \sum_{i=1}^{N} \mathbf{p}_i^{'} \mathbf{SM} \mathbf{q}_i^{'T}. \tag{6.10}$$

The size of the **SM** matrix is $N \times N$ where as the size of the **p**' matrix is $4 \times N$ (under homogeneous transformation) which results in $4 \times N$ matrix. The result when multiplied with $\mathbf{q}^{'T}$ (which is a $N \times 4$ matrix) will have the final result in a 4×4 matrix form which is the size of the Matrix **H**. The rest of the algorithm is the same as the traditional algorithm where we compute the SVD of **H** as given by Equ. 6.11

$$\mathbf{H} = \mathbf{U} \Lambda \mathbf{V}^T. \tag{6.11}$$

Now the Rotation component is calculated as given by Equ. 6.12.

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T. \tag{6.12}$$

The determinant of R must be a positive integer. The translation component is simply computed as the difference of the centroids computed in Equ.6.9. Algorithm 2 describes the entire procedure of our proposed method.

Algorithm 2 CoSM Algorithm

function ReadDataSets($\mathbf{P_s}$,$\mathbf{P_t}$) ▷ Read the *Source* and the *Target* datasets.
  while not converged **do**
    $\mathbf{c} \leftarrow$ **ComputeCorrespondence**$(\mathbf{P_s}, \mathbf{P_t})$. ▷ Compute Correspondence
    $\mathbf{SM} = zeros(N, N)$ ▷ Initialize $N \times N$ Similarity Matrix to all zeros.
    for $i \leftarrow 1$ to N **do**
      $\mathbf{d} = \mathbf{P_s}(i) - \mathbf{P_t}(\mathbf{c}(i))$.
      Compute G_σ as shown in Equ.6.6.
      $\mathbf{SM}(i, \mathbf{c}(i)) = G_\sigma(\mathbf{P_s}(i) - \mathbf{P_t}(\mathbf{c}(i)))$.
      $\mathbf{SM}(\mathbf{c}(i), i) = G_\sigma(\mathbf{P_s}(i) - \mathbf{P_t}(\mathbf{c}(i)))$.
    end for
    Compute centroid as given in Equ.6.8.
    Compute the difference as given in Equ.6.9.
    Compute the **H** Matrix as given in Equ.6.10.
    Compute SVD of **H** as given in Equ.6.11.
    Compute **R** as given in Equ.6.12.
    Compute **tr** as difference of centroids.
  end while
end function=0

6.3.2 Properties of the Similarity Matrix

6.3.2.1 SM Matrix is a sparse Matrix

Since at every iteration, we initialize the values to zeros and based on Equ. 6.7 we fill the Correntropy similarity values at a specified location, the matrix is sparse. Line

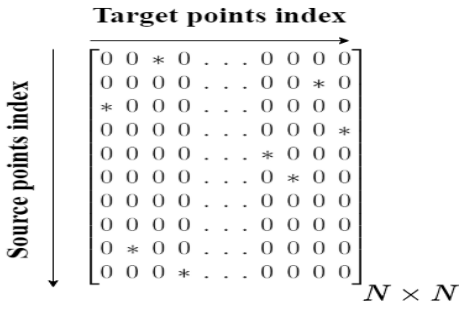


FIGURE 6.3: Similarity Matrix.'*' represent the Correntropy values.

8 and 9 in Algorithm 2 indicated that we update the indices in the *Source* and the *Target* point indexes based on the Correntropy criterion. Fig. 6.3 shows a simple structure of the similarity matrix.

6.3.2.2 The rows and columns are linearly independent

Every element in the **SM** matrix or $\mathbf{SM}(i, j)$ represents the Correntropy relationship between the ith *Source* point and the jth *Target* point. Since we are computing the similarity of the point from the *Source* index to the *Target* index, every other indices in that particular row/column is intuitively zero. This clearly means for every row index in the *Source* point set there is only one corresponding index point in the *Target* dataset with a similarity metric which assures one to one relationship between every *Source* point to every *Target* point. The points in the *Source* point set are transformed based on this rotation matrix after every iteration. This process is repeated every iteration and one can find that the number of points with closer similarity metric (~ 1) increases. Fig 6.4 shows that as the iterations increases the rank approaches N.

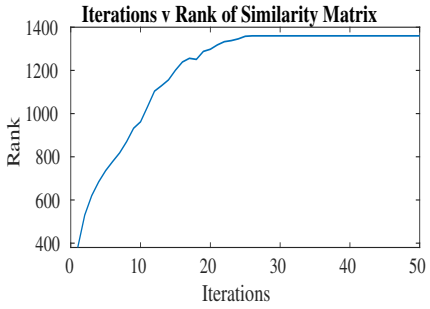


FIGURE 6.4: As the iterations increases, the rank of the matrix approaches N. Here N is 1360.

6.3.2.3 Similarity Matrix is a Mirror-Symmetric Matrix.

From Equ. 6.7, one can verify that if the matrix is diagonally separated, then it's a mirror image of each other (Mirror-Symmetric Matrices) which means $\mathbf{SM} = \mathbf{SM}^T$.

6.4 Results

6.4.1 Evaluation on Datasets with no outliers.

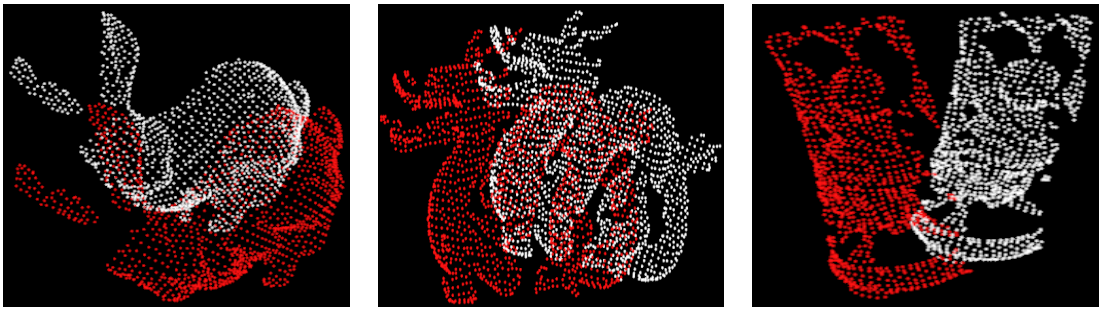
We validated our approach on three simple and well-known datasets: the Bunny Rabbit, the Happy Buddha, and the Dragon dataset. In addition, we have also performed our evaluation on a Lidar dataset (KITTI dataset) since it captures data on a large scale. We perform our experiment on an Intel i7 2.GHz CPU with 32 Gb of RAM. We implemented our approach in the PCL library (version 1.9) and released our code in the GitHub repository (`https://github.com/aralab-unr/CoSM-ICP`). In this section, we did not add any random outliers, and we introduced random rotation and translation between the *Source* and the *Target* point sets. First, we capture a sample pcd file and transform the acquired point cloud with a random

transformation matrix where all the individual components of the transformation matrix were randomly generated with different seeds (we assume throughout the paper that the units for angle are in radians). We name the original point cloud as read from the pcd file as *Target* and the randomly transformed point cloud as the *Source*. The problem statement is to find the relative transformation between the *Source* and the *Target* such that the *Source* is aligned with the *Target*. For the above steps, the Algorithm is described in Algorithm 3.

Algorithm 3 Transform the point cloud with a Random Transformation Matrix.

function ReadPCDFile('filename.pcd',$\mathbf{P_t}$). ▷ Read the PCD file and name it as a Target.
    a_{ax}= Random angular component along the X-axis (-6.28 to 6.28 radians).
    a_{ay}= Random angular component along the Y-axis (-6.28 to 6.28 radians).
    a_{az}= Random angular component along the Z-axis (-6.28 to 6.28 radians).
    tr_{tx}= Random translation along the X-axis.
    tr_{ty}= Random translation along the Y-axis.
    tr_{tz}= Random translation along the Z-axis.
    $\mathbf{M} \leftarrow$ **GenerateTransformationMatrix**$(a_{ax}, a_{ay}, a_{az}, tr_{tx}, tr_{ty}, tr_{tz})$
    $\mathbf{P_s} \leftarrow$ **TransformPointCloud**$(\mathbf{P_t}, \mathbf{M})$
    return $\mathbf{P_s}$ ▷ Return the transformed point cloud and name it as Source.
end function

To validate our approach on the datasets, we collected the results on a simple scenario where the *Source* is transformed manually with a simple transformation matrix which has less variation in rotation and translation ($(r, p, y, x, y, z) = (0.314, 0, 0, 0, 0, 0.05)$). We reduce the number of point clouds to decrease the computational time by performing Voxel grid filtering (leaf size 0.05) on the point clouds. The original number of points in the bunny point cloud sets is 40256 points, and after applying voxel grid filtering, we get the number of points reduced to 1360 points. Similarly, for the Dragon dataset, the original number of points is 43467 points, and after voxel grid

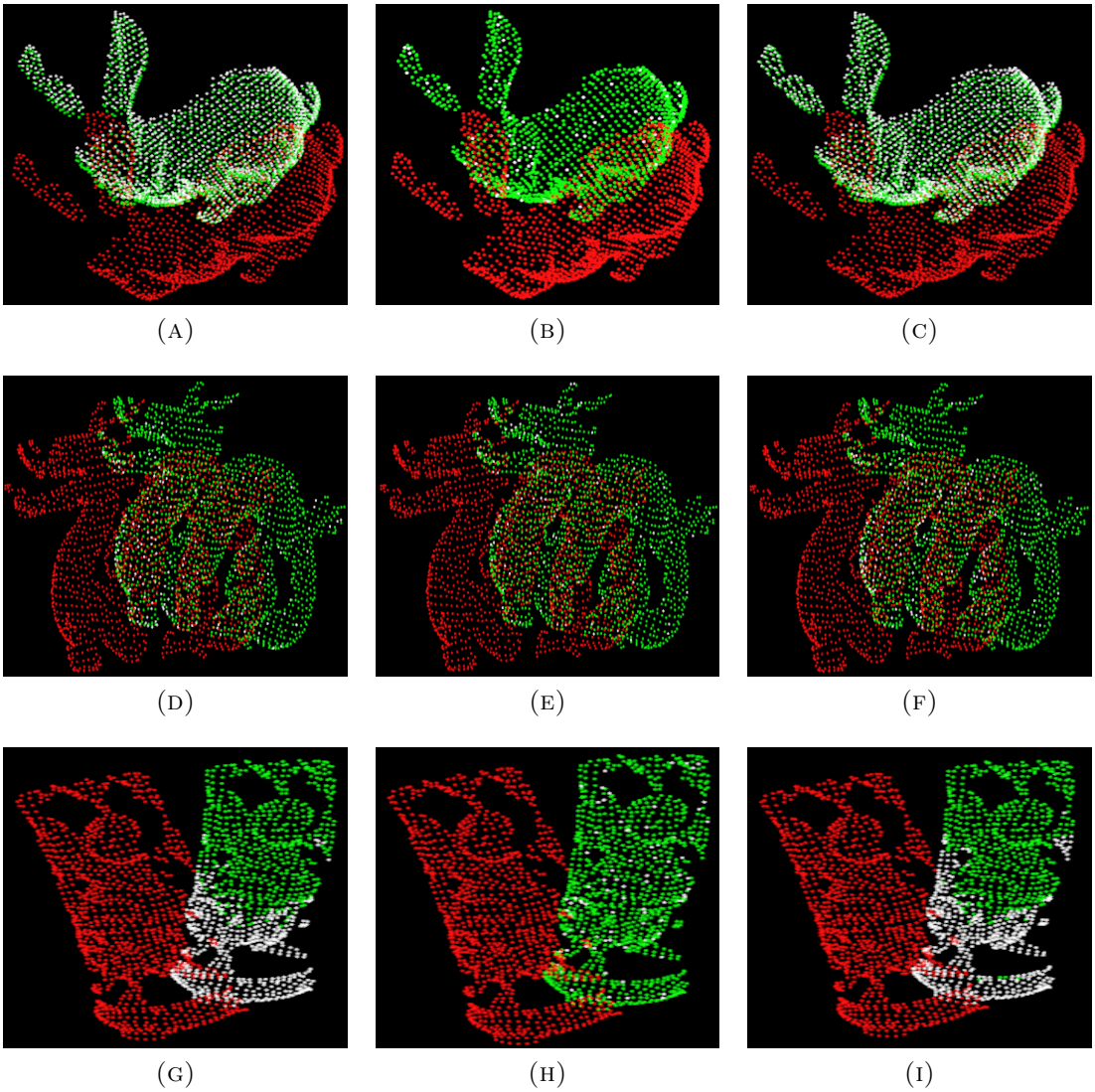


(A) (B) (C)

FIGURE 6.5: Rotated and translated point cloud (white is the original point cloud (*Target*), and Red is the transformed point cloud (*Source*)) with rotation of $(r, p, y) = (0.314, 0, 0)$ and translation of 0.05 unit in the z-axis. (a) shows the bunny rabbit point cloud down sampled using voxel grid filtering of leaf size 0.005. (b) shows the dragon point cloud down sampled using voxel grid filtering of leaf size 0.005. (c) shows the happy buddha point cloud down sampled using voxel grid filtering of leaf size 0.005.

filtering, the number of points is reduced to 1593 points. The original number of points for the Happy Buddha dataset is 76166 points, and after voxel grid filtering, we get 1090 points. The *Source* and the *Target* point clouds for the datasets are shown in Fig. 6.5.

After applying a few iterations, our approach aligns equally well (RMSE: 2.46336e-14) in comparison to the well-known methods like ICP with the standard SVD method (RMSE: 6.25494e-06) and ICP point-to-plane (RMSE: 2.77175e-16). The results for different datasets are shown in Fig. 6.6. We compared the Root Mean Square Error (RMSE) on this particular transformation, and we see that ICP point to plane converges faster than other methods, including ours (in 50 iterations). Fig. 6.7 shows the RMSE comparisons of different approaches on different datasets for this simple transformation. In this example, ICP point to plane seems to converge faster than



(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.6: Transformation from *Source* to *Target*: (r,p,y,x,y,z)=(0.314,0,0,0,0,0.05). White point cloud: Original point cloud (*Target*). Red point cloud: *Source* point cloud and Green point cloud : *Source* transformed point cloud after 10 iterations using (a) ICP Standard SVD (RMSE: 6.25494e-06) (b) ICP Point to Plane (RMSE: 2.77175e-16) and (c) CoSM ICP (RMSE: 3.34226e-06) on the Bunny Rabbit dataset. Similarly we perform 50 iterations on the Dragon dataset using (d) ICP Standard SVD (RMSE: 2.86089e-14) (e) ICP Point to Plane (RMSE: 3.93822e-16) and (f) CoSM ICP (RMSE: 2.72838e-15) and 35 iterations on the Happy Buddha dataset using (g) ICP Standard SVD (RMSE: 1.72851e-13) (h) ICP Point to Plane (RMSE: 3.91357e-16) and (i) CoSM ICP (RMSE: 2.46336e-14).

our method since the rotation and translation components are very small. The next step is to evaluate how well our method performs as compared to the other methods when the rotation is significant.

Fig. 6.9 shows the results of different approaches, including ours, when the rotation and the translation component of the transformation matrix is quite large. In this example, we use the Bunny Rabbit dataset where the *Source* is transformed from the *Target* as:

$$(r, p, y, x, y, z) = (-1.32811, -5.87854, 2.12814, -0.874, -0.433, 0.221).$$

The translation component is small compared to the rotation component. In this example, our method performs well compared to the other methods, with RMSE as 4.76074e-06 (in our method) after 20 iterations. It clearly shows that the *Source* is aligned very well with our approach, whereas in other methods, the *Source* failed to align with the *Target* point cloud. We compared the RSME results with other approaches like GICP, NDT, and ICP-nonlinear [93–95]. Fig 6.8 shows the RMSE results for various methods on the Bunny Rabbit dataset. Our method performs well in comparison to others. In addition, Table 6.2 shows the RMSE comparison of various methods applied to the Bunny Rabbit dataset with various rotations and translations.

In Fig. 6.10 we use the Dragon dataset where the *Source* is transformed from the *Target* as:

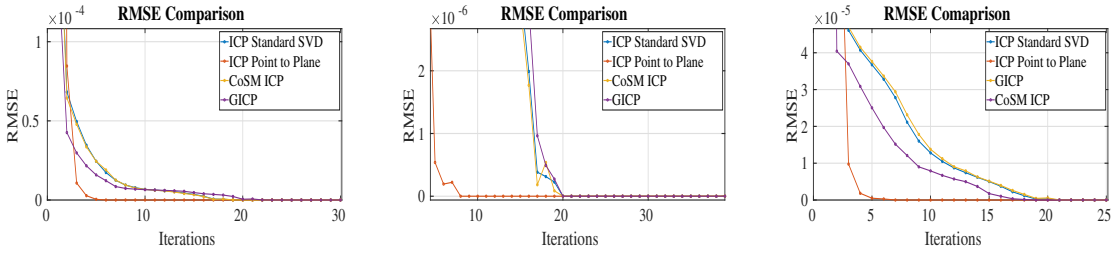
TABLE 6.2: RMSE comparison of different methods with different values of rotation and translation after 50 iterations on Bunny Rabbit dataset. Note: Rotation component is in radians.

Transformation(r,p,y,x,y,z)	ICP Standard SVD	ICP Point to Plane	GICP	ICP Non-Linear	NDT	CoSM ICP
(2.39384,-2.57132,4.66973,0.876204,-2.83931,2.68268)	0.000428388	88.0656	0.0011591	0.000206335	15.5141	1.81066e-13
(6.10518,-0.249119,2.41527,1.99458,8.99637,1.20097)	0.000274292	268.684	81.2352	0.000301596	81.2352	3.09602e-13
(1.17438,-5.95203,-4.13622,4.6532,6.28659,0.0542642)	0.000177495	147.987	58.9048	2.69103e-11	58.9048	1.75634e-13
(-0.866749,-2.6182,-0.318386,-2.1561,-1.25001,-4.8753)	0.000283007	859.114	28.7628	0.000244988	28.7628	1.98801e-13
(5.08434,-3.9644,-2.66895,2.45251,-6.82633,1.41512)	0.000471378	58.1505	55.996	1.54e-11	55.996	2.63875e-13

$(r, p, y, x, y, z) = (2.39318, -5.02554, -2.69076, 0.000, -0.003, 0.003)$.

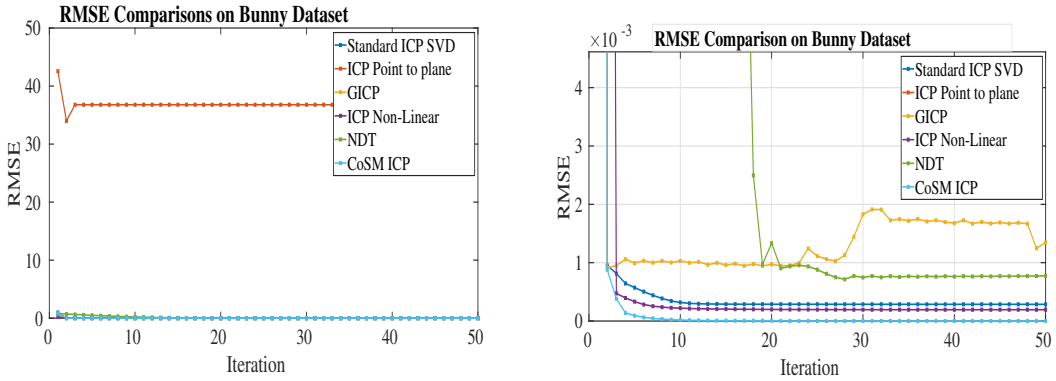
Again, our method performs well compared to other methods with RMSE as 8.66664e-08 (in our method) after 25 iterations. We arrive at a similar conclusion that the *Source* is aligned very well with our approach, whereas in other methods, the *Source* failed to align with the *Target* point cloud. Fig. 6.11 shows the RMSE results for various methods on the Dragon dataset. In addition, Table 6.3 shows the RMSE comparison of various methods applied to the Dragon dataset with various rotations and translations.

In Fig. 6.12 we use the Happy Buddha dataset where the *Source* is transformed from the *Target* as $(r, p, y, x, y, z) = (-4.50504, 1.31677, 4.83251, -0.023, -0.019, -0.008)$. Again, our method performs well in comparison to the rest of the methods with RMSE as 2.26675e-06 in our method after 25 iterations. We arrive at a similar conclusion that the *Source* is aligned very well with our approach where as in other methods the *Source* failed to align with the *Target* point cloud. Fig 6.13 shows the RMSE results for various methods on the Happy Buddha dataset. In addition, Table 6.4 shows the RMSE comparison of various methods applied on the Happy Buddha dataset with various rotation and translation.



(A) (B) (C)

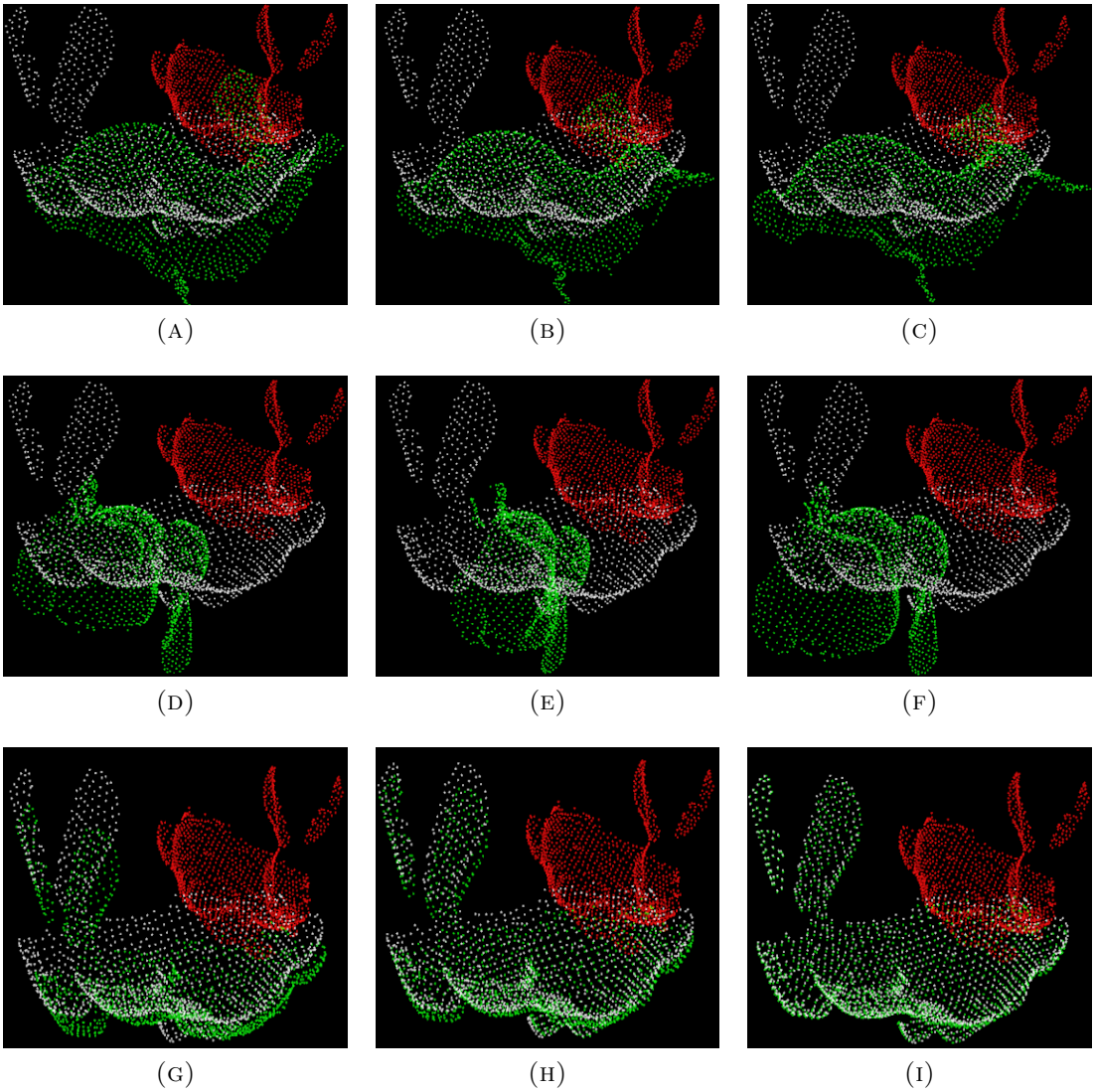
FIGURE 6.7: Convergence of different Registration Methods (ICP Standard SVD, ICP point to plane, GICP and CoSM ICP) on different datasets on the simple transformation $((r, p, y, x, y, z) = (pi/10, 0, 0, 0, 0, 0.05))$. (a) Shows RMSE comparison on Bunny Rabbit dataset. (b) Shows RMSE comparison on the Dragon dataset. (c) RMSE comparison on the Happy Buddha dataset.



(A) (B)

FIGURE 6.8: RMSE comaprison of various methods on the Bunny Rabbit dataset. (b) shows the zoomed in version of (a).

For evaluating on the Lidar dataset, we applied a voxel grid filtering of leaf size 1.5. The original point cloud size is 115150 points, and after voxel grid filtering, we get 1497 points. In Fig. 6.14 we use the Lidar dataset where the *Source* is transformed from the *Target* as $(r, p, y, x, y, z) = (-4.27108, -0.505914, 0.0988647, 10.938, -10.532, 17.832)$. Again, our method performs well compared to the rest of the methods with RMSE as 2.26675e-06 in our method after 25 iterations. We arrive at a similar conclusion that the *Source* is aligned very well with our approach, whereas in other methods, the



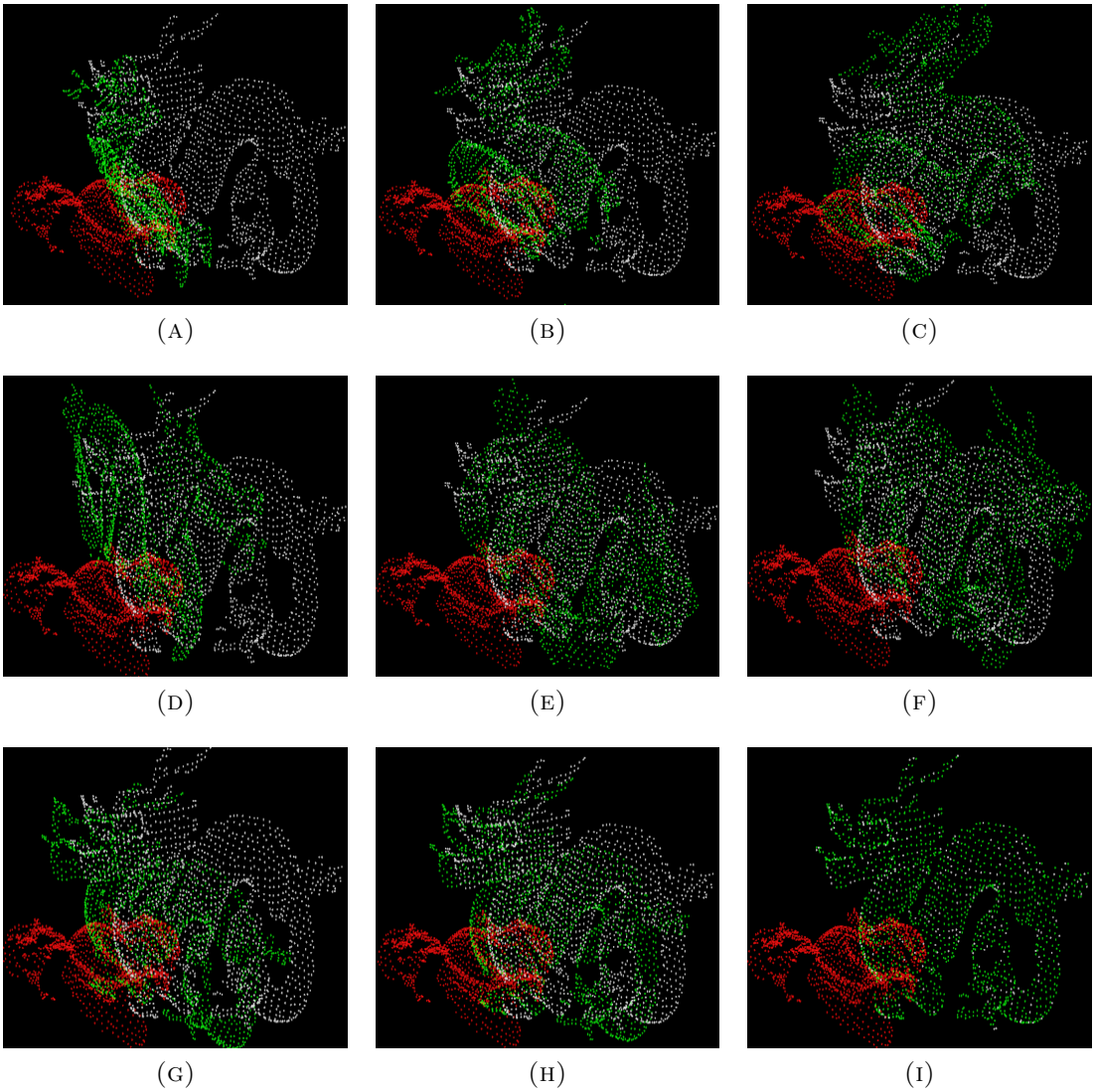
(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.9: White Point Cloud: Original Point Cloud (*Target*). Red Point Cloud: *Source* point cloud. Green Point Cloud: *Source* transformed point cloud after applying iterations. *Source* is transformed from the *Target* as ((r,p,y,x,y,z)=(-1.32811,-5.87854,2.12814,-0.874,-0.433,0.221)). (a)-(c) shows the convergence of the *Source* point clouds after 5, 10 and 20 iterations using the standard ICP, respectively (RMSE after 20 iterations is 0.000358474). (d)-(f) shows the same using ICP Point to Plane (RMSE after 20 iterations is 0.00216325). (g)-(i) shows the same using CoSM ICP (RMSE after 20 iterations is 4.76074e-06).

Source failed to align with the *Target* point cloud. Fig 6.15 shows the RMSE results for various methods on the Lidar dataset. In addition, Table 6.5 shows the RMSE comparison of various methods applied on the Lidar dataset with various rotation



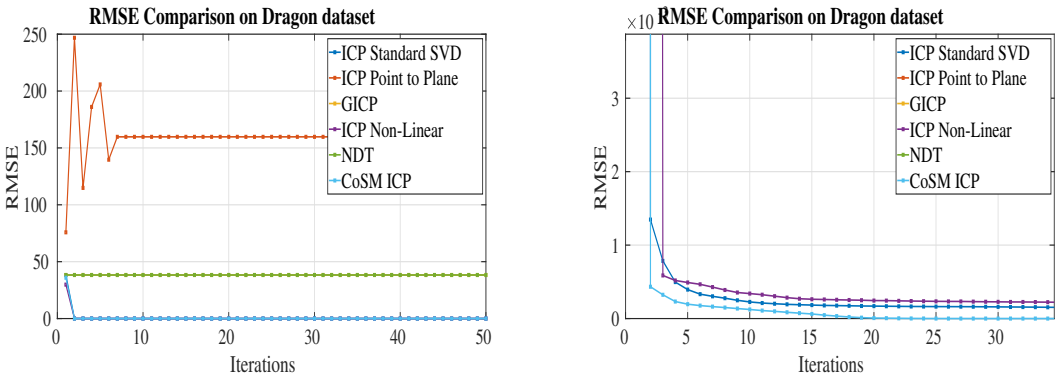
(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.10: White point cloud: Original Point Cloud (*Target*). Red point cloud: *Source* point cloud. Green point cloud: shows the *Source* transformed point cloud after applying different registration methods. *Source* is transformed from the *Target* as ((r,p,y,x,y,z)=(2.39,-5.025,-2.69,0.00,-0.003,0.003)). (a)-(c) shows the convergence of the *Source* point clouds after 5, 10 and 25 iterations using the standard ICP, respectively (RMSE after 25 iterations is 0.000198014). (d)-(f) shows the same using ICP Point to Plane (RMSE after 25 iterations is 0.000136451). (g)-(i) shows the same using CoSM ICP (RMSE after 25 iterations is 8.66664e-08).

and translations.



(A) (B)

FIGURE 6.11: RMSE comparison of various methods on Dragon dataset. (b) shows the zoomed in version of (a).

TABLE 6.3: RMSE comparison of different methods with different values of rotation and translation (from *Source* to *Target*) after 50 iterations on Dragon dataset. Note: Rotation component is in radians

Transformation(r,p,y,x,y,z)	ICP Standard SVD	ICP Point to Plane	GICP	ICP Non-Linear	NDT	CoSM ICP
(-2.8152,-2.09474,-2.48844,5.66472,6.31125,-9.44663)	0.000167091	638.278	156.91	0.000159398	156.9	4.19748e-13
(0.429847,2.60943,3.61045,3.6405,-1.63246,7.60966)	0.00016037	325.136	73.1995	0.000224256	73.1995	1.25329e-13
(-2.78018,2.15172,-4.43629,-4.52232,-9.42573,-2.54953)	0.000142098	3015.76	115.142	0.000162551	115.142	3.78298e-13
(1.75465,-1.56088,-1.70889,2.33321,6.13018,3.79368)	0.000142188	26.6913	56.4895	0.000224482	56.4895	1.94386e-13
(-3.72204,1.05945,2.34622,-4.65481,9.74086,3.64636)	0.000159037	115.65	127.159	126.91	127.159	4.19583e-13

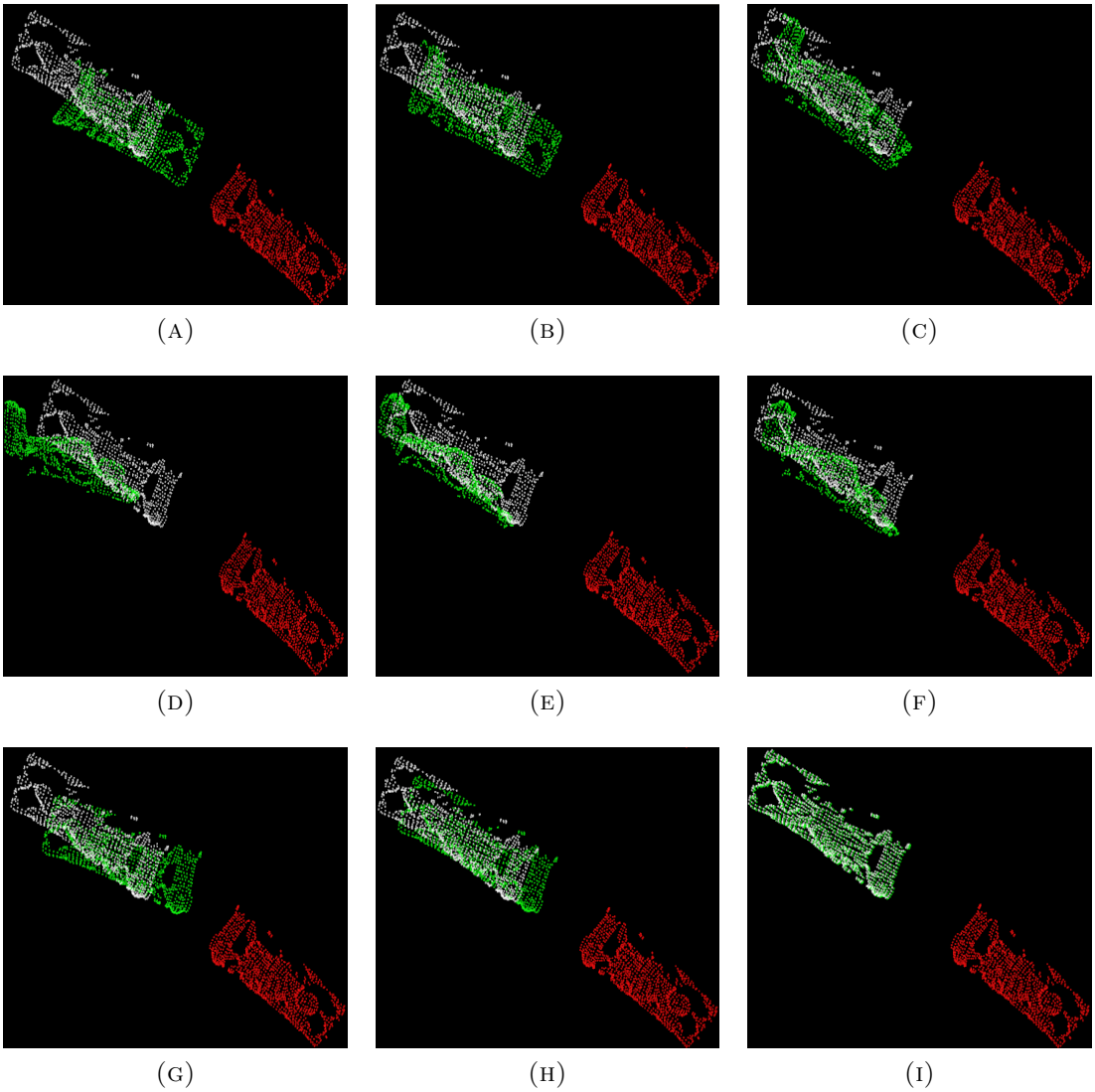
TABLE 6.4: RMSE comparison of different methods with different values of rotation and translation (from *Source* to *Target*)after 50 iterations on the Happy Buddha dataset. Note: Rotation component is in radians

Transformation(r,p,y,x,y,z)	ICP Standard SVD	ICP Point to Plane	GICP	ICP Non-Linear	NDT	CoSM ICP
(0.429847,2.60943,3.61045,3.6405,-1.63246,7.60966)	0.00016037	325.136	73.1995	0.000224256	73.1995	1.25329e-13
(-1.58692,-5.66055,-4.80461,-7.03304,0.718292,2.77415)	0.000142378	469.843	55.5392	0.00016041	55.5392	1.41924e-13
(-2.79112,4.31946,-2.15715,2.44105,2.03992,-1.87023)	0.000121835	39.1843	0.0018519	0.000133238	11.981	7.99904e-14
(-0.744191,-1.86187,-1.78262,-4.61717,1.79701,-7.04776)	0.000192252	81.0697	71.9632	0.000192185	71.9632	1.80283e-13
(5.17263,1.57234,1.9238,-0.263428,5.05962,-2.27954)	0.000157366	116.57	27.6866	0.000182242	27.6866	1.38844e-13

6.4.2 Evaluation on datasets with outliers.

In this section, we evaluated our approach in cases where the data has outliers. We inject random data values at random indexes in the *Source* point set. The algorithm for injecting random outliers is shown in Algorithm 4.

In this experiment, we selected the percentage of points in the *Source* dataset to be



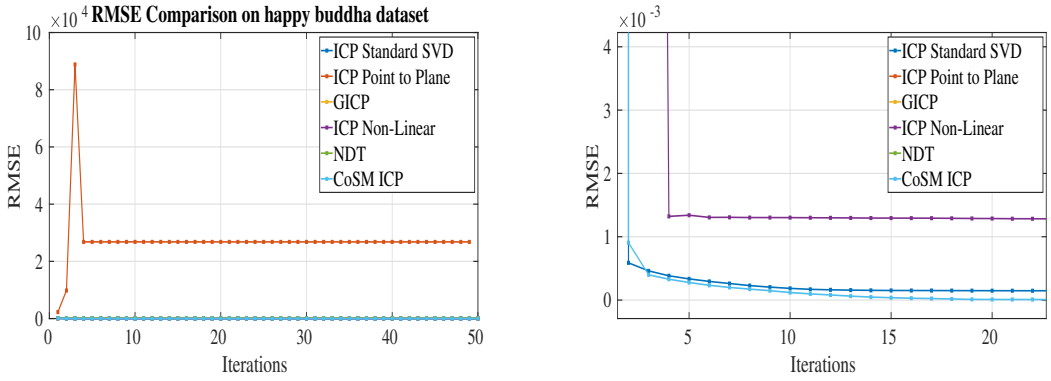
(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.12: White point cloud: Original point cloud (*Target*). Red point cloud: *Source* point cloud. Green: *Source* transformed point cloud after applying different registration methods. Source is transformed from the *Target* as ((r,p,y,x,y,z)=(-4.50504,1.31677,4.83251,-0.023,-0.019,-0.008)). (a)-(c) shows the convergence of the *Source* point clouds after 5, 10 and 25 iterations using the standard ICP, respectively (RMSE after 25 iterations is 0.000128905). (d)-(f) shows the same using ICP Point to Plane (RMSE after 25 iterations is 0.000124717). (g)-(i) shows the same using CoSM ICP (RMSE after 25 iterations is 2.26675e-06).

affected by outliers. We added the outliers after performing voxel grid filtering of leaf size 0.05 since one can still encounter outliers after pre-processing the data. The *Source* point cloud is transformed from the *Target* point cloud with the transformation



(A) (B)

FIGURE 6.13: RMSE comparison of various methods on the Happy Buddha dataset. (b) shows the zoomed in version of (a).

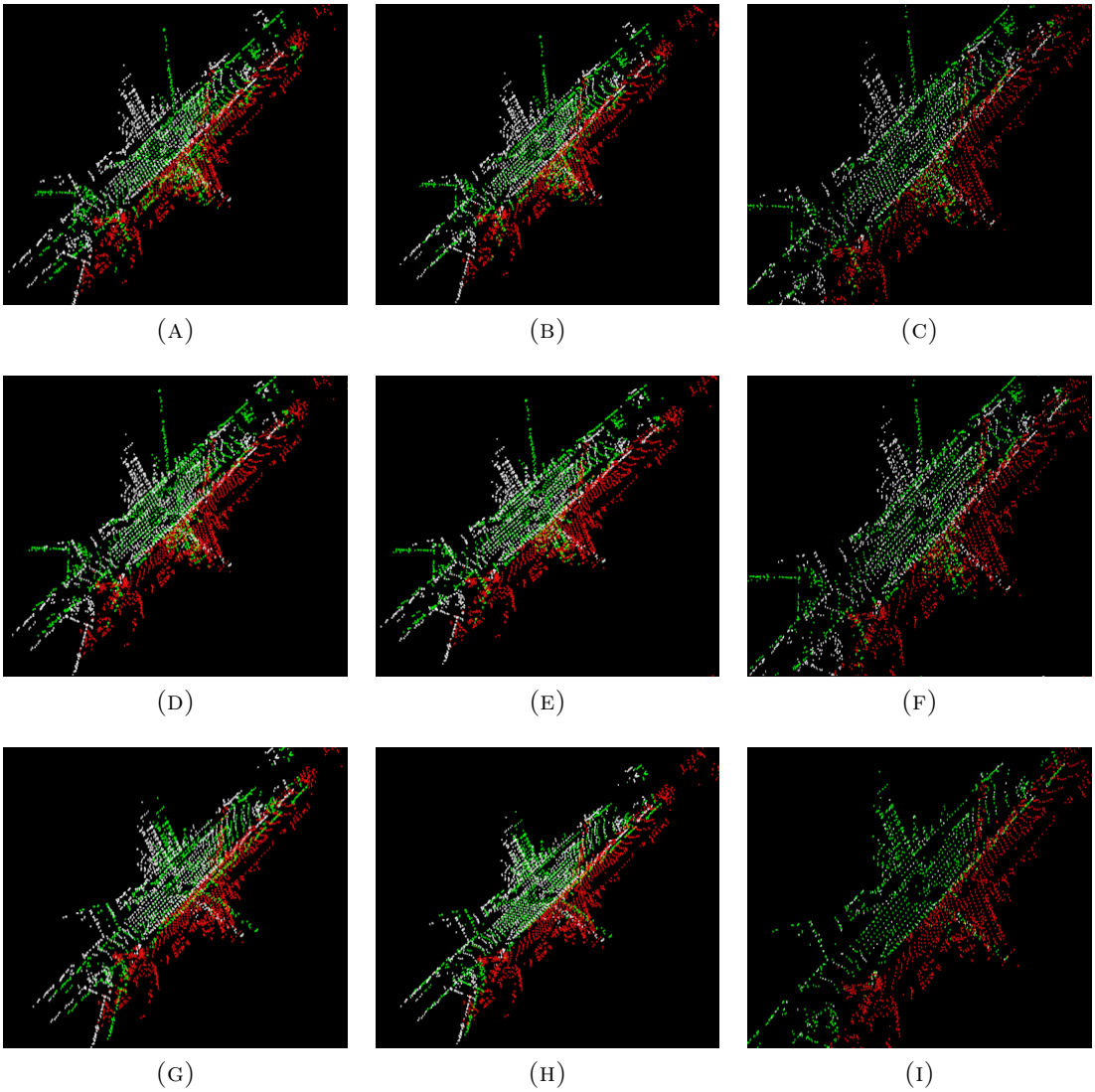
TABLE 6.5: RMSE comparison of different methods with different values of rotation and translation (from *Source* to *Target*)after 50 iterations on the KITTI Lidar dataset. Note: Rotation component is in radians

Transformation(r,p,y,x,y,z)	ICP Standard SVD	ICP Point to Plane	GICP	ICP Non-Linear	NDT	CoSM ICP
(4.23443,-5.64022,-3.13665,-2.2485,-1.19326,-9.12248)	16.1754	371.296	38.1373	16.1762	274.547	3.4779e-08
(3.64853,-3.29044,-5.13981,0.991908,-5.06315,4.75733)	16.9795	480.78	321.312	23.8551	432.754	3.20148e-08
(-5.78829,0.576843,1.91183,0.640607,6.7615,3.82172)	194.469	513.454	225.474	194.581	257.164	4.47064e-08
(-1.0146,-0.427581,-3.42779,7.22804,-7.6078,-0.806072)	10.2524	286.673	18.9515	10.266	24.5575	2.83469e-08
(0.149154,4.33846,1.82095,-2.5633,-0.377752,-1.20834)	16.8181	1010.43	248.244	17.2776	258.734	4.46591e-08

Algorithm 4 Add random data values at random data indexes.

function InjectOutliers($\mathbf{P_s}$). ▷ Read the Source point cloud data.
  $ridx$= Pick random index in the Source point set.
  tr_{tx}= Random translation along the X-axis.
  tr_{ty}= Random translation along the Y-axis.
  tr_{tz}= Random translation along the Z-axis.
  $\mathbf{P_s}[ridx].x = \mathbf{P_s}[ridx].x + tr_{tx}$
  $\mathbf{P_s}[ridx].y = \mathbf{P_s}[ridx].y + tr_{ty}$
  $\mathbf{P_s}[ridx].z = \mathbf{P_s}[ridx].z + tr_{tz}$
  return $\mathbf{P}_s$ ▷ Return the 'infected' point cloud and name it as Source.
end function

$(r, p, y, x, y, z) = (-1.32811, -5.87854, 2.12814, -0.874, -0.433, 0.221)$. The *Source* point cloud is indeed affected by outliers, and we compare the results with different approaches on different datasets when 10% ,25%, and 50% of the *Source* data points are affected.



(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.14: (From KITTI dataset) White point cloud: Original Point Cloud(*Target*). Red point cloud: *Source* point cloud and Green point cloud: *Source* transformed point cloud after applying different registration methods. Source is transformed from the *Target* as ((r,p,y,x,y,z)=(-4.2,-0.5,0.098,10.9, -10.5, 17.8)). (a)-(c) shows the convergence of the *Source* point clouds after 7, 15 and 33 iterations using the standard ICP, respectively (RMSE after 33 iterations is 13.7695). (d)-(f) shows the same using ICP Point to Plane (RMSE after 33 iterations is 12.6385). (g)-(i) shows the same using CoSM ICP (RMSE after 25 iterations is 1.01689e-08).

We compared our approach with other methods as well. Table 6.6 shows the RMSE comparison with other datasets where 10% of the *Source* dataset (after voxel grid

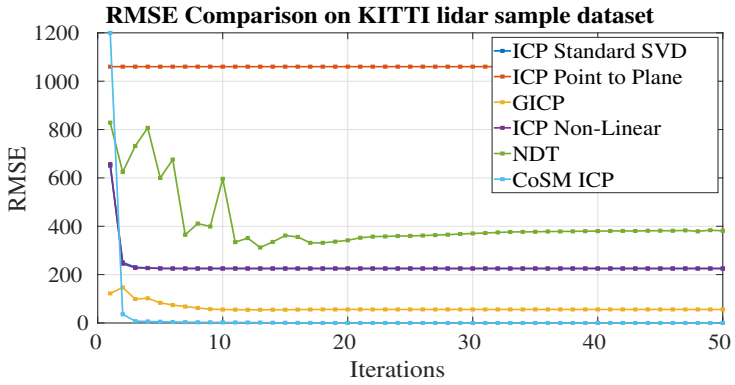


FIGURE 6.15: RMSE comparison of various methods on the KITTI Lidar dataset.

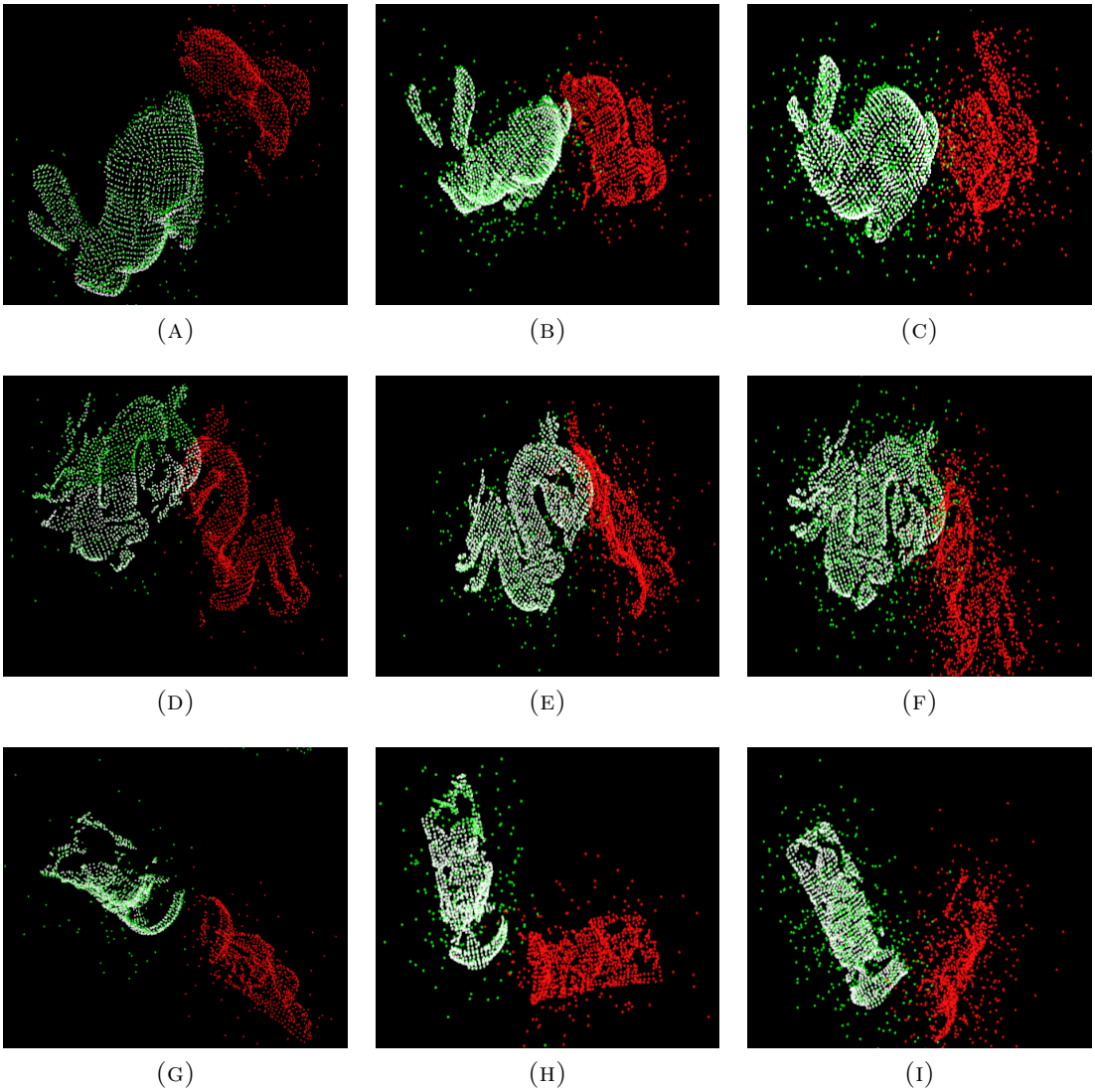
filtering) is is affected. Our methods outperform other approaches under various rotations and translations. Our results are consistent and keeps performing better even when 25% and 50% of the *Source* dataset are affected as shown in Table 6.7 and Table 6.8.

TABLE 6.6: (10% of the *Source* data contains outliers) RMSE comparison of different methods with different values of rotation and translation (from *Source* to *Target*) after 50 iterations on different datasets. Note: Rotation component is in radians

Bunny Dataset						
Transformation(r,p,y,x,y,z)	ICP Standard SVD	ICP Point to Plane	GICP	ICP Non-Linear	NDT	CoSM ICP
(-2.36822,-0.726873,-5.69612,7.08856,-2.68581,7.63531)	0.000411328	711.375	114.581	0.000269638	114.581	6.15845e-05
(-2.67216,3.73756,-0.902484,-9.27928,-4.0282,4.09878)	0.000257389	374.895	0.000460177	23.8551	119.48	7.70252e-05
(-1.56059,4.39878,-2.80893,-3.94166,-3.10092,-3.34176)	0.000527419	304.884	37.13	0.000487805	37.13	8.26112e-05
Dragon Dataset						
(0.177274,-3.62568,5.47677,0.852604,-5.3915,-9.91804)	0.000220957	76.301	128.455	0.000246111	128.455	6.49243e-05
(-0.620232,-5.82563,3.78543,4.85066,2.74563,-8.42101)	0.000270745	45.484	101.049	23.8551	101.049	7.94277e-05
(-1.53521,4.48718,-0.388112,-7.74233,5.42215,-6.44992)	0.00021795	241.345	125.922	0.000223478	125.922	7.67165e-05
Happy Buddha Dataset						
(-2.83367,-1.24896,2.98587,2.69059,3.34046,-4.41227)	0.000176202	19.1229	36.8922	0.000212469	36.8922	7.10533e-05
(-2.47527,-5.65213,-3.30425,-3.8167,8.95128,-9.03962)	0.000210153	598.839	174.049	0.000217757	174.049	8.03656e-05
(3.05982,-0.897597,-1.43556,1.60636,1.22125,7.01582)	0.000171509	146.972	52.0521	0.000202291	52.0521	7.24014e-05

6.4.3 Effect of σ.

As shown in the previous section, the registration works well when the value of σ is 100. With lower values of σ like 0.01, it aligned well for small rotation and translation.



(A) (B) (C)

(D) (E) (F)

(G) (H) (I)

FIGURE 6.16: CoSM Results when *Source* is infected with noise. White point cloud: Original Point Cloud (*Target*). Red point cloud: Infected *Source* point cloud. Green point cloud: *Source* transformed point cloud after applying iterations. Transformation from *Source* to *Target*: ((r,p,y,x,y,z)=(1.658, 0.607, -1.204, 0.00, -0.003, 0.003)). (a),(b) and (c) show the result of CoSM ICP on the Bunny dataset when 10%, 25% and 50%of the data is affected with outliers. Their respective RMSE's are $8.64002e-05$,0.000203088 and 0.000384025. (d),(e) and (f) show the result of CoSM ICP on the Dragon dataset when 10%, 25% and 50%of the data is affected with outliers. Their respective RMSE's are $6.91792e-05$,0.000163833 and 0.000368516. (g),(h) and (i) show the result of CoSM ICP on the Happy Buddha dataset when 10%, 25% and 50%of the data is affected with outliers. Their respective RMSE's are $9.29047e-05$,0.000205528 and 0.000386966. For each case, we performed around 30 iterations.

TABLE 6.7: (25% of the *Source* data contains outliers) RMSE comparison of different methods with different values of rotation and translation (from *Source* to *Target*) after 50 iterations on different datasets. Note: Rotation component is in radians

Bunny Dataset						
Transformation(r,p,y,x,y,z)	ICP Standard SVD	ICP Point to Plane	GICP	ICP Non-Linear	NDT	CoSM ICP
(1.55693,-1.11205,4.17676,-6.05252,1.66971,5.82581)	0.000426784	1175.52	71.583	0.000365769	71.583	0.000187812
(5.65505,4.15868,-3.27751,9.31612,-5.23313,-5.55587)	0.000346638	3286.84	0.000410574	144.696	144.696	0.000167037
(-2.55584,-4.76118,3.16486,-7.20725,3.99654,-7.30791)	0.000447408	1343.79	117.165	0.000552115	117.165	0.000216027
Dragon Dataset						
(6.25758,2.3638,-2.88282,-7.36544,-6.20769,-7.92445)	0.000294534	67.9046	156.303	0.000286822	156.303	0.000168995
(3.76055,2.28707,4.29734,-8.8644,-6.40219,1.19919)	0.000255092	115.102	120.245	0.000292728	120.245	0.000158364
(4.63246,-4.08457,-1.65193,-9.97667,-4.37054,1.77417)	0.000354324	174.437	119.75	0.000263999	119.75	0.000162741
Happy Buddha Dataset						
(-1.20164,-2.47146,4.2161,1.29076,7.69533,-6.64823)	0.000295019	116.015	98.0711	0.000298207	98.0711	0.000194422
(-1.14973,-4.81181,3.44981,9.95025,9.69451,8.84972)	0.000266775	103.766	269.094	0.000262239	269.094	0.000177326
(-2.78376,3.84551,-5.11847,-0.469235,-7.93148,-7.98029)	0.000269701	116.321	127.049	0.000269828	127.049	0.0001856

TABLE 6.8: (50% of the *Source* data contains outliers) RMSE comparison of different methods with different values of rotation and translation (from *Source* to *Target*) after 50 iterations on different datasets. Note: Rotation component is in radians

Bunny Dataset						
Transformation(r,p,y,x,y,z)	ICP Standard SVD	ICP Point to Plane	GICP	ICP Non-Linear	NDT	CoSM ICP
(4.21267,-5.1913,0.510754,4.78028,-3.81009,-8.99842)	0.000555246	176.793	118.74	0.000705962	118.74	0.000367226
(-1.90501,2.94602,-4.15732,-4.38121,-8.43926,0.951326)	0.000533201	1644.03	91.374	0.000535398	91.374	0.000364791
(-3.88617,5.23854,-4.81381,2.95017,-6.77983,-2.26068)	0.000615907	83.6865	61.5916	0.000610313	61.5916	0.00036124
Dragon Dataset						
(-4.41725,-2.40238,-0.642351,3.74842,-9.32348,-9.14091)	0.000487516	188.167	185.885	0.000510906	185.885	0.000372615
(3.29757,5.37519,4.68352,5.59739,-1.76298,-7.59656)	0.000362955	487.439	90.1547	0.000516225	90.1547	0.000327693
(-2.27986,-2.66004,-5.61896,-9.89552,-2.83092,4.9319)	0.000520534	91.5809	127.933	0.000472107	127.933	0.000380396
Happy Buddha Dataset						
(5.4771,1.17243,-0.67022,-8.42324,-5.29389,-3.70151)	0.000462191	510.054	111.689	0.000489964	111.689	0.000384029
(1.77436,-2.70017,-4.70393,1.9514,4.74639,-8.0765)	0.000478957	128.078	90.493	0.000489046	90.493	0.000383038
(-3.01961,2.91297,-5.51066,6.01917,4.29907,5.78239)	0.000486108	155.204	86.8654	0.00173299	86.8654	0.000399139

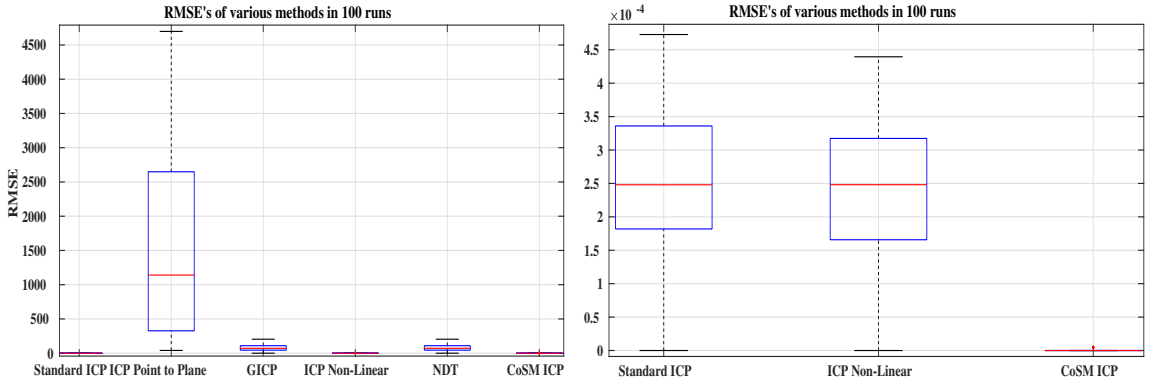
However for large rotation and translation between the *Source* and the *Target* it took a large number of iterations to converge. For even smaller values of σ like 0.001, and for larger rotation and translation, alignment did take place. Moreover, the iterations were larger (~ 500), and the computed transformation was prone to errors.

6.5 Discussion

We have found that using a Correntropy Relationship matrix proved to be very effective in registration. Through experiments, we have evaluated that our approach is robust to large rotation and translation as well as robust to outliers. We have compared our approach to other state-of-the-art methods like GICP, NDT, ICP Non-Linear, etc., and proved it's efficiency under large transformation matrices. It is quite important to note that the size of the Correntropy Matrix is dependent on the size of the dataset, which means that if we have larger datasets, our approach would be computationally very expensive for each iteration ($\sim 7ms$ for a dataset of size 10000 points for each iteration). In addition to our previously mentioned results, we have also evaluated on different datasets using multiple runs. We define a run such that in each run, we generate a random transformation matrix, and we compute a transformation matrix from randomly generated translation and rotation components and transform the original point cloud (which we call as *Target*) to another point cloud (which we call it as *Source*). We then employ several state-of-the-art methods like GICP, NDT, ICP Non-Linear NDT, and CoSM ICP iteratively on the *Source* point cloud. In this case, we perform the iteration 50 times during each run and collect the final RMSE's from each of the methods. The components of the random transformation matrix (translation and rotations (x, y, z, r, p, y)) are generated independently with different seeds (the system's clock defines seeds). The mean for all the components is zero. The standard deviation for the translation components

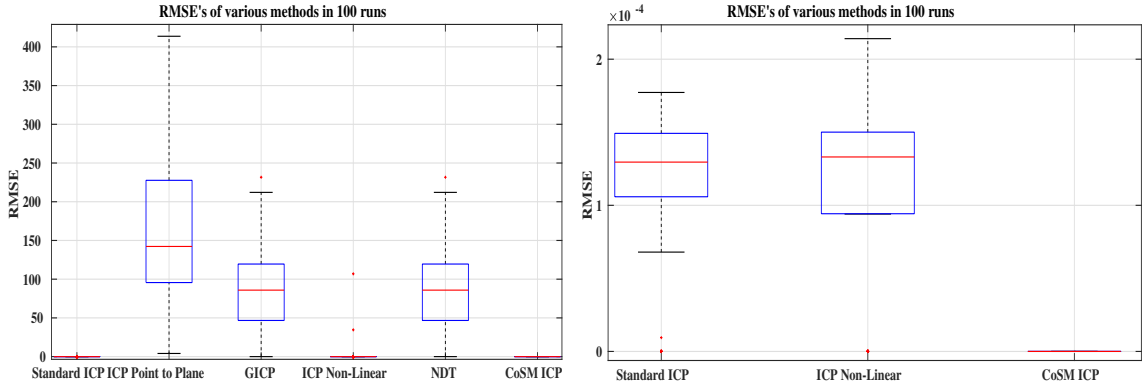
along x, y, z is 10, whereas the angular components are 6.28317 radians (360 degrees) along each axes. On the Bunny dataset, we found that ICP Point to Plane had a very large RMSE (mean RMSE for ICP Point to Plane is 1.9668e+03 in 100 runs) which means the *Source* dataset was not at all aligned to the *Target* dataset. However, GICP and NDT are better than ICP Point to Plane in most cases (the mean RMSE of GICP in 100 runs is 75.6727, and the mean RMSE of NDT in 100 runs is 76.2536). However, in certain transformations, GICP and NDT perform well with RMSE of 8.8582e-04 and 5.6122e-04. On the other hand, if we compare the average RMSE of Standard ICP, ICP Non-linear and CoSM ICP, their average RMSE's in 100 runs is 2.4156e-04, 2.3781e-04 and 3.2489e-07, respectively. CoSM ICP is better than the other approaches (by more than $\sim$ 100%). Fig 6.17(a) shows the distribution of errors (RMSE's) of various methods in 100 runs on the Bunny Rabbit dataset. Fig. 6.17(b) shows the RMSE deviation of Standard ICP, ICP Non-Linear, and CoSM ICP. In the Dragon dataset, the average RMSE's of Standard ICP, ICP Point to Plane, GICP, ICP Non-Linear, NDT and CoSM ICP is 1.1372e-04, 177.0397, 86.8183, 1.4154, 87.5327 and 2.6989e-13, respectively. In this case as well, CoSM outperformed the other approaches by more than $\sim$ 100%. It is again evident from Fig. 6.18(a) and Fig. 6.18(b). On the Happy buddha dataset, the average RMSE's for Standard ICP, ICP Point to Plane, GICP, ICP Non-Linear, NDT and CoSM ICP is 9.4830e-05, 414.0633, 93.0712, 1.1289e-04, 93.3621 and 5.6055e-13 respectively in 100 runs. Again it is validated that CoSM ICP performs better than the rest of the approaches by more than $\sim$ 100%. Fig. 6.19(a) and Fig. 6.19(b) clearly validates our

claim for the happy buddha dataset. Readers can note that the top and the bottom of the box plot shown represent the 25^{th} and 75^{th} percentiles, respectively, and the middle red line shows the median of the error (average RMSE's) across 100 runs.



(A) (B)

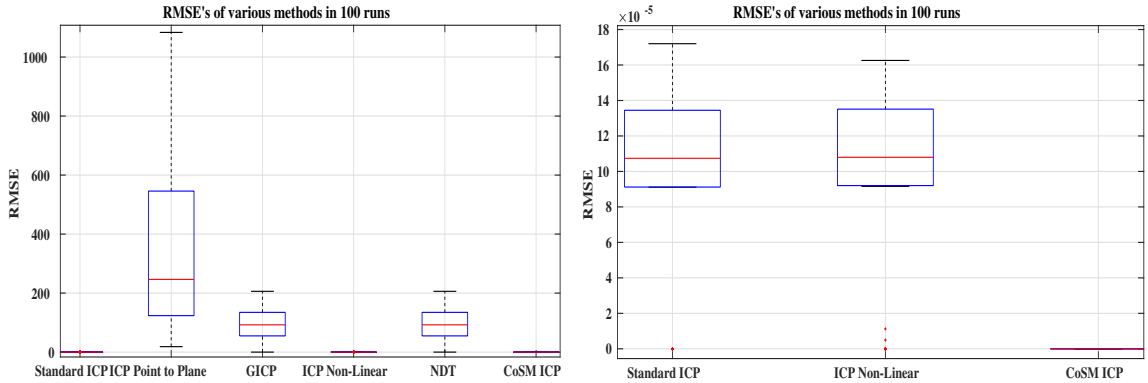
FIGURE 6.17: Average RMSE's of various methods in 100 runs (Bunny Rabbit dataset). Each run consists of random rotation and translation between the *Source* and the *Target*.



(A) (B)

FIGURE 6.18: Average RMSE's of various methods in 100 runs (Dragon dataset). Each run consist of random rotation and translation between the *Source* and the *Target*.

It is significant to note that our approach has proved beneficial in solving the registration problem under various rotations and translations as compared to other well-known methods. We encourage the community to validate our approach on their own



(A) (B)

FIGURE 6.19: Average RMSE's of different methods in 100 runs (Happy Buddha dataset). Each run consist of random rotation and translation between the *Source* and the *Target*.

datasets.

6.6 Conclusions

One of the critical things we have addressed is to find an accurate estimate of the transformation between the *Source* and the *Target* point clouds under various rotations and translations. In this work, we have verified our approach in various datasets, where the *Source* is transformed from the *Target* using various randomly generated transformation matrices. We can see through the obtained results that our method has performed better than the other state-of-the-art approaches. In addition, we also evaluate our approach when the *Source* dataset is affected by noise generated with random intensity and affecting random data points. In this scenario as well, we have verified that our proposed approach has outperformed most of the other state-of-the-art approaches where the 'infected' *Source* dataset is transformed randomly from the

Target dataset. We firmly insist that our readers evaluate our approach located in our Github repository(`https://github.com/aralab-unr/CoSM-ICP`).

Chapter 7

Single Frame Lidar Stereo Camera Calibration Using Registration of 3D planes

7.1 Motivation

This chapter focuses on finding the extrinsic parameters (rotation and translation) between the Lidar and the Stereo sensor setups. We use a planar checkerboard and place it inside the Field-of-View (FOV) of both the sensors, where we extract the 3D plane information of the checkerboard acquired from the sensor's data. The planes extracted from the sensor's data are used as reference data sets to find the relative transformation between the two sensors. We use our proposed method Correntropy

Similarity Matrix Iterative Closest Point (CoSM-ICP) Algorithm, as mentioned in the previous chapter, to estimate the relative transformation. In this work, we use a single frame of the point cloud data acquired from the Lidar sensor and a single frame from a calibrated Stereo camera point cloud to perform this operation. We evaluate our approach on a simulated dataset since it has the freedom to evaluate under multiple configurations. Through results, we verify our approach under various configurations.

7.2 Background

Primitive approaches for an autonomous multi-sensor system involve a predefined setup where the sensors are placed at known locations relative to each other. This setup does not necessarily involve any calibration methodology since the sensors' relative translation and rotation components are known. However, with the advent of unique designs of autonomous systems in the market and the research community, it has become essential for automatic and efficient calibration methods for multi-sensor setups. Calibration in a multi-sensor system is essentially finding the relative transformation between the sensors. With our focus on autonomous navigation, Lidar and stereo camera configurations are explicitly designed to the requirement of the autonomous system. These sensors, however, need to be calibrated (finding the relative rotation and translation between the two sensors) for efficient autonomous navigation [96].

Fig. 7.1 shows a sample setup of Lidar-Stereo Calibration. Early methods that involved extrinsic calibration between Lidar and a camera (or a stereo camera) made use of a calibration card or some well-defined calibration objects [96–103]. A planar checkerboard is one of the most widely used calibration objects since it can easily extract features from both the Lidar and the camera data. Most of these approaches have highlighted the importance of placing the calibration objects in the Field-of-View (FOV) of both sensors. Finding the correspondences between the Lidar and the stereo data points is crucial for efficient Lidar and stereo camera calibration. The correspondences are calculated either manually or automatically using feature extraction algorithms. The accurate estimation of the correspondences is essential for efficient calibration.

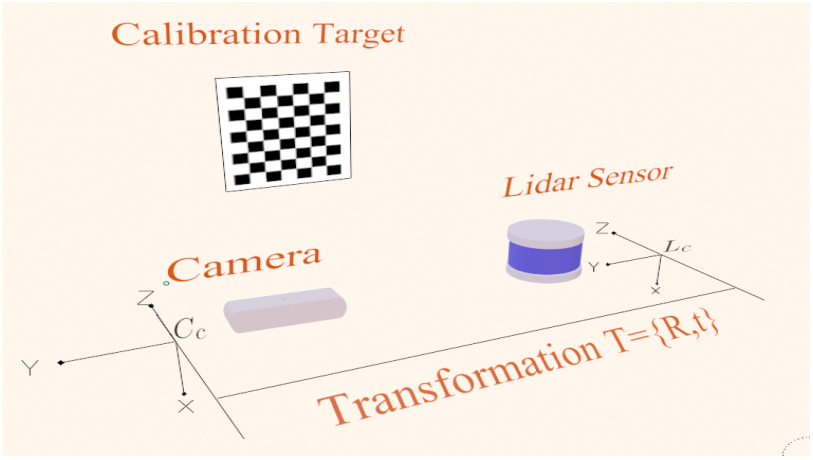


FIGURE 7.1: Sample Lidar and stereo camera configuration setup. C_c is the stereo camera coordinate frame (here we consider the left camera center as stereo camera's coordinate frame), and L_c is the Lidar coordinate frame. The objective here is to compute the transformation T between L_c and C_c.

The method proposed by Zhang and Pless [97] has used a planar checkerboard pattern as a calibration target to calibrate a 2D laser scanner and a monocular camera.

The checkerboard was placed with different poses in front of the 2D laser and the camera. The relative transformation in this method is calculated by minimizing the re-projection error of features of the checkerboard as viewed by the 2D Lidar and the camera. Their algorithm required at least five poses, whereas, in theory, the extrinsic calibration could be achieved in 3 poses. This method was later extended to a 3D laser by Unnikrishnan, and Hebert [104], in which they estimated the plane parameters from the 3D Lidar and the camera data. They calculate the initial rotation and translation using plane-plane correspondence and later refine it by minimizing the point to plane distance. Another work by Geiger et al. [105] placed several checkerboards on the scene where its algorithm detects the checkerboards and matches them with the planes from the laser data. In their approach, a simple one shot of the scene is enough since there are several checkerboards. Most of these methods involve the use of plane information of the checkerboard and do not consider the boundary of the checkerboard. Work by [106] has shown accurate results even with fewer poses of the calibration target. Since the laser point can approximate the boundary more accurately when the checkerboard is placed closer to the sensor, it shows high accuracy. The authors introduced a similar transformation between the Lidar and the camera, which showed better results than rigid transformation.

Other calibration objects were also used for the calibration method. Work proposed in [100] used a polygonal board, and using the vertices of the board, they estimated the calibration parameters. However, it involved manual labelling of the correspondences

in the camera frame. These correspondences are used to estimate the calibration parameters using a genetic algorithm-based approach. Other approaches [101–103] have used planar boards with rectangular holes, planar boards with circular holes, and other 3d objects. One of the drawbacks of using all the above methods is that it is laborious and time-consuming.

Along with the approaches mentioned above, several target-less or no object-based calibration approaches have been proposed as well [107, 108]. Initial work by Scaramuzza [109] has introduced a technique for calibrating a 3D laser and an omnidirectional camera from natural scenes. The algorithm automatically extracts some features from the sensor data and manually selects correspondence between the selected features. Another work by Moghadam [110] exploits the linear features present in the indoor environment. It uses the 3D line features from the point cloud and the corresponding 2D line segment from the camera to estimate the rigid body transformation between the two. Boughorbal [111] uses a chi-square test to maximize the correlation between the sensor data. This technique exploits the statistical dependence of the data acquired from the two sensors. Levinson and Thrun [112] uses a series of corresponding laser scans and camera images to estimate the calibration parameters. Work by [113] uses similar mutual information metrics between the two sensors' data by measuring the statistical dependence between the data [114].

From Pandey, [113], a theoretical derivation is proposed, which estimates the kernel density of the probability distribution of the sensor data. Scott [114] has proposed

an approach for automatic calibration using diligently selected natural scenes. This algorithm searches over the selected scenes to extract models and yields better results. One advantage of this approach is that it requires no knowledge of the physical world and continuously finds scenes that constraints the optimization parameters. One recent and exciting approach by Jeong [115] exploits known features such as road markings. These features are captured by both the Lidar and the stereo sensors and use a multiple cost function for robust optimization even with rough initial values. Work by Jeong [115] uses the road as a reference for computing transformation, and it is required to be done in a controlled environment. It is also time-consuming and requires significant effort to obtain the transformation parameters.

We propose an easy and efficient process to perform the Lidar-stereo camera calibration using a checkerboard calibration target. Our work is similar to the work proposed by [116], where we compute the plane coefficients using the data from both the Lidar and the camera data. Later on, these coefficients are used to construct a well-structured set of 3D points residing in that plane. Our work differs from the above-mentioned works, where we use our own proposed Correntropy Similarity Matrix(CoSM ICP) approach for aligning the points in the plane and computing the relative transformation between the sensors. In essence, the significant contributions of our work are outlined as follows:

- We develop our algorithm based on finding planes acquired from both the Lidar and the camera sensor's data.

- We compute the plane coefficients separately from both the sensor's data.
- From the plane coefficients acquired from both the sensors, we determine the plane's location and populate the plane with structured data points.
- Our proposed algorithm only needs to populate a limited number of points for the plane to plane matching.
- We use our CoSM ICP approach to find the relative transformation between the points present in the plane.

7.3 Proposed Methodology

In this section, we describe the steps involved in our Lidar-stereo calibration. We perform different steps corresponding to the data acquired from the 3D Lidar sensor and the stereo camera data. The overall procedure involved in our work is shown in Fig. 7.2. In this work, use only a single frame of the Lidar's 3D data and the stereo camera's 3D data, which contains the 3D points of the calibration target. For the 3D Lidar sensor, we manually select the region containing the 3D points corresponding to the calibration target (checkerboard). We do the same for the stereo camera data, and we assume that the camera is calibrated and we know the intrinsic parameters of both the cameras in the stereo sensor setups. It was convenient for us to use the same calibration target for camera calibration as well. Again, the key point is to determine the plane coefficients acquired from both the Lidar's 3D data and the

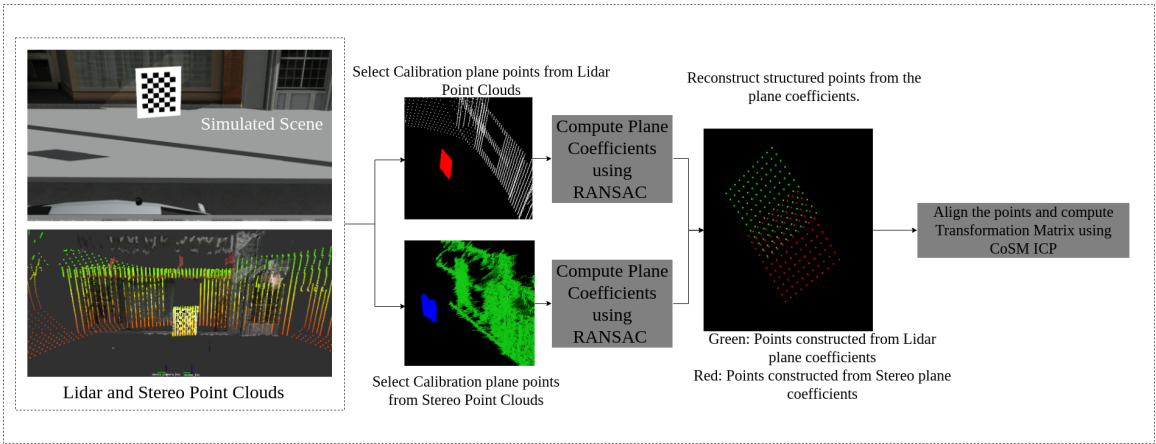


FIGURE 7.2: Flowchart of our approach.

stereo camera's computed 3D data. Fig. 7.2, as read from left to right, initially shows the simulated scene setup that includes a Lidar and a stereo camera separated by a certain transformation. The left camera centre defines the stereo camera's frame of reference. We directly show the 3D computed points from the stereo sensor setup for simplicity (3D points are computed using the well-known Stereo Global Block Matching(SGBM) of the image pairs). We manually select the region which contains the points corresponding to the calibration target from both the Lidar 3D point clouds and the stereo computed 3D point cloud. We compute the plane coefficients for each of these selected regions using Random Sample Consensus (RANSAC). We construct a well-structured set of points from the computed plane coefficients (from the Lidar and the stereo sensor data). We use our CoSM ICP approach to compute the transformation between these point sets. The transformation computed by the CoSM ICP returns the relative transformation between the Lidar and the stereo sensors.

7.3.1 Lidar data processing

Using a single calibration target is relatively easy and intuitive for an extrinsic calibration between multiple sensors. As mentioned earlier, the first step of our process is to capture the 3D points (from the Lidar sensor) and the 3D points (from the stereo camera) that contain the calibration target. From the 3D data, we manually pick the region containing the points corresponding to the calibration target and ignore the rest. This selected data contains points of the calibration target, which is effectively a plane. Other automatic approaches, like distance filtering[117], can be used for this process, but we let the user pick the region for complete control. Now, from this 'selected' point set, we intend to get the corresponding plane coefficients. RANSAC is our choice for this process, which determines the best-fit plane of 3D points using inliers. We use the following steps for our work.

- Randomly selects 3 points from the 'selected' points.
- Compute the plane equation using these 3 points.
- Compute inliers using the computed plane with all other 3D points.
- Repeat the process with the highest inlier ratio.

For this setup, we set the maximum iterations for our RANSAC algorithm as 1000. 3D points that are within 10mm from the plane are considered inliers. The inlier ratio, which crosses 90% or more, is considered as a best-fit plane. The plane equation

computed from the Lidar points is given as $a_l x + b_l y + c_l z + d_l = 0$, where a_l,b_l,c_l and d_l represent the coefficients of the plane.

7.3.2 Stereo camera data processing

This section details the process of computing the 3D plane equations of the calibration target using the point clouds generated from the stereo camera setup. In this work, we assume that the camera is already calibrated, and we know the intrinsic parameters. For our experiments, we perform the calibration steps that are implemented in Robot Operating System (ROS). For camera calibration, we use the same checkerboard calibration target that we have used to perform the Lidar-camera calibration as mentioned in this work. To compute the point clouds from a pair of stereo images, we use the popular Semi-Global Block Matching (SBGM) algorithm [118] to generate the depth map from the stereo image pairs. Based on the quality of the images (resolution, frame rate), we can tune multiple parameters of the SGBM algorithm (block size, speckleRange, speckleWIndowSize, etc.) to get the desired quality of the depth map for further point cloud generation. In our framework we use the well known ***StereoSGBM()*** function provided by OpenCV [119]. The reader is suggested to refer to [118] and [119] to explore further options in point cloud generation from stereo images. We manually select the 3D points computed from the stereo camera and compute the plane equation corresponding to the 3D points using RANSAC. We follow the same steps as mentioned in the Lidar data plane computation:

- Randomly selects 3 points from the 3D point clouds computed from the stereo data.
- Compute the plane equation using these 3 points.
- Compute inliers using the computed plane with all other 3D points.
- Repeat the process with the highest inlier ratio.

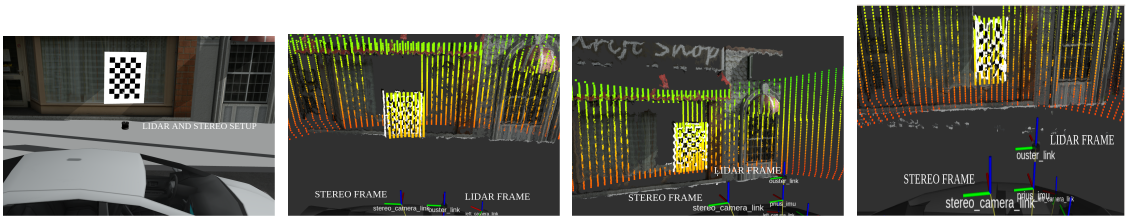
The plane equation computed from the camera points is given as $a_s x + b_s y + c_s z + d_s = 0$, where a_s, b_s, c_s and d_s represent the coefficients of the plane.

7.3.3 Transformation estimation

We compute the transformation between these sensors from the plane equations calculated from both the Lidar and the camera data. For this step, we use the computed plane equations to populate the planes with a fixed number of points separated by a known distance (e.g., 100 points). The point sets generated from this process can be 'aligned'. The resultant transformation gives us the transformation between the Lidar and the camera sensor. The primary reason for this process is that the number of points in the Lidar point set is different from the number of points in the computed 3D points from the stereo camera. Our CoSM ICP needs to have an equal number of points for alignment. We describe our CoSM ICP approach by first describing the Correntropy criterion and then using it in our approach to compute the transformation between the Lidar and the stereo data points.

7.4 Results

We perform our initial evaluation in a simulated environment provided by Open Robotics[120]. To demonstrate our results, we start with a basic simulated dataset containing simulated Lidar data and a simulated stereo camera setup (mounted on a simulated Prius model car) established in a simulated environment in Gazebo, which is shown in Fig.7.3 (a) and (b) [121]. The primary reason for selecting this simulation setup is that we can compare our results since we know the ground truth, and this setup contains other complexities in the environment, like buildings and cars (beyond the calibration card). Our experiments test the results with a linear transformation ranging from $0.05m$ to $2.5m$ (with x, y, z axes). We set up a simple test case of a Lidar-stereo setup where the stereo camera faces the calibration target and is placed near the Lidar sensor under various configurations. (e.g., 5cm along the y axis of the Lidar sensor or $\mathbf{t} = [0, 0.05, 0.0]$). We intend to calculate the transformation between the stereo's left camera (C_c) with respect to the Lidar's frame L_c. The stereo camera has a baseline of $7cm$ between the left and the right cameras. Both the cameras have the resolution of 1280×720, and since we use a simulated setup, we ignore the radial and tangential distortion (both were set to 0). In this scenario, we evaluate our approach on multiple configurations where the ground truth is already known, as it can be seen in Fig. 7.3. So, in this case, our problem statement is defined as finding the transformation between the Lidar(L_c) and the Stereo camera(S_c) setup using our proposed methodology. This setup provides a base test case to verify our algorithm.



(A) (B) (C) (D)

FIGURE 7.3: Simulation setup for evaluating Lidar-stereo calibration under multiple configurations: (a) denotes a gazebo simulation of Prius car model with Lidar and stereo camera; (b) denotes the TF-frames of the Lidar and the stereo camera. The link *ouster_link* is the reference frame for Lidar; (c) and (d) denote TF-frames of multiple configurations of Lidar-stereo setup. $\mathbf{t} = [t_x, t_y, t_z]$ denotes the translation component along x, y, z. It essentially denotes how the stereo camera is transformed with respect to the Lidar sensor along x, y, z. For (c) the translation between the Lidar and the stereo sensor is $\mathbf{t} = [0, 0.5, 0.0]$, and for (d) it is $\mathbf{t} = [-0.5, 0.5, 0]$.

Thus it can be extended to complex real-time test cases.

7.4.1 Evaluation on Simulated data

The various configurations under which we performed our experiments are shown in Table 7.1. It shows the original ground truth transformations between the Lidar and the stereo sensor. We place the calibration target at an average distance of 2m to 3m from the calibration target throughout our experiments (this distance is with respect to the Lidar coordinate frame (L_c)).

After performing the steps mentioned in our approach, we collect the computed transformations returned by our approach. Table 7.2 shows the individual estimated transformation computed using our method. The reason for separating the components is to evaluate each component separately.

TABLE 7.1: Configuration setups used in our experiments (or Ground Truth).

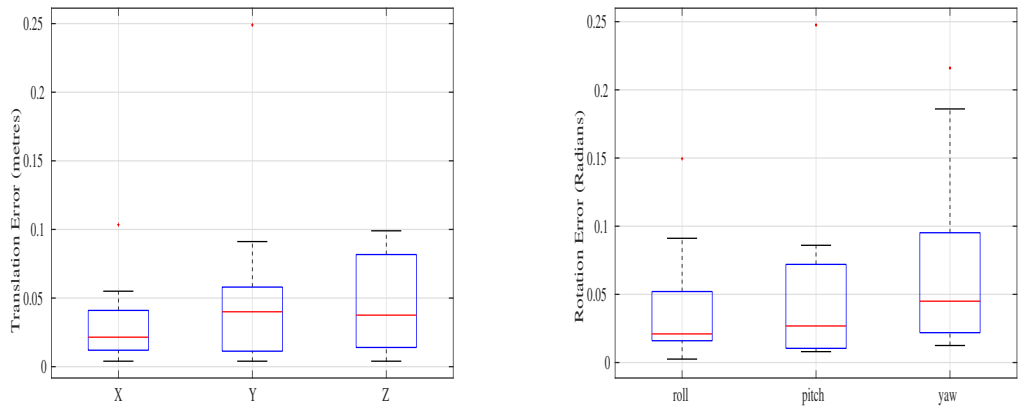
Setting	$t_x(m)$	$t_y(m)$	$t_z(m)$	roll (rad)	pitch(rad)	yaw(rad)
1	0	0.4	0.0	0.0	0.0	0.0
2	0	0.1	0.0	0.0	0.0	0.0
3	0	0.05	0.0	0.0	0.0	0.0
4	0	0.6	0.0	0.0	0.0	0.0
5	0	1.2	0.0	0.0	0.0	0.0
6	0	2.2	0.0	0.0	0.0	0.0
7	0.5	0.5	0.0	0.0	0.0	0.0
8	−0.5	1.5	0.3	0.0	0.0	0.0
9	−0.5	0.5	0	0.0	0.0	−0.0
10	0.5	0.5	0	0	0	−0.523599
11	0.5	0.5	0.3	0	0.349066	−0.523599
12	0.3	0.6	0.4	0	0.349066	−0.523599
13	0.2	0.3	0.2	0.261799	0	−0.523599
14	0.7	0.2	0.9	0	0.349066	0

TABLE 7.2: Individually computed transformation based on our approach.

Setting	$t_x(m)$	$t_y(m)$	$t_z(m)$	roll (rad)	pitch(rad)	yaw(rad)
1	0.019	0.342	−0.074	−0.108	−0.032	−0.0090
2	−0.044	−0.091	0.031	−0.0280	0.01700	0.0089
3	0.004	0.046	0.009	−0.0129	0.0069	0.00999
4	0.004	−0.655	0.082	0.046	0.0430	0.0100
5	−0.015	−1.147	0.036	−0.016	0.0039	0.0109
6	0.025	−2.165	0.013	0.0169	0.014	0.0099
7	−0.504	−0.468	0.004	−0.0360	0.0040	0.0059
8	0.541	−0.494	0.091	0.033	0.039	0.0090
9	−0.476	0.549	0.401	−0.093	−0.0049	−0.001993
10	0.488	0.545	0.045	−0.532	0.009	−0.00099
11	0.445	0.567	0.267	−0.5341	0.2085	−0.166
12	0.345	0.571	0.431	−0.2740	0.1023	−0.0663
13	0.163	0.291	0.201	−0.174	0.0209	−0.166
14	0.661	0.242	1.03	−0.331	0.0989	−0.123

For further evaluation, we show the average error in multiple configurations as done in the experiments. Fig.7.4 shows the average error of the individual components of the rotation and translation components under various configurations. From the data as given in Table 7.2, we can see that under simple translation along x, y, z axes, our algorithm performs well (with individual RMSE's ~ 0.01 along with all the individual components). Even under small rotations, our approach returns relatively close values (RMSE ~ 0.05) compared to the ground truth. However, when the transformation between C_c and L_c is significant (>60 degrees or 1.0472 radians), the overall RMSE

increases, and the computed transformation is quite far off from the ground truth values (RMSE >1.3541).



(A) (B)

FIGURE 7.4: Translation and Rotation errors of individual components(x, y, z and roll,pitch and yaw) under various configurations.

7.4.2 Effect of σ

For all the experiments performed in this work, σ was set to 100. With lower values of σ like 0.01, the computed transformation was equally well for slight rotation and translation. However for significant rotation and translation between the *Source* and the *Target* it took a large number of iterations to converge. For even smaller values of σ like 0.001, the computed transformation was relatively well for large rotation and translation (RMSE $\sim$ 0.01). However, it took $\sim$ 500 iterations to converge.

7.5 Discussion

One of the critical observations here was to estimate 3D points computed from the stereo camera setup accurately. Throughout the experiments, we collected datasets for all the individual configurations as mentioned in Table 7.1. We hand-picked each dataset that had a good collection of 3D point sets with less visible outliers. However, we hope that with further advancements in 3D depth estimation, we can get more accurate and robust results.

Since it was a simulated setup, it could have been simple to have a Lidar sensor, a stereo camera, and a calibration target. However, one of our primary goals was to have a calibration technique in various environments. We plan to evaluate our approach in real-time; however, evaluating a Lidar and a stereo camera setup under various configurations is time-consuming. So, in our simulated environment, we add environmental complexities and evaluate our approach. We let the user select the region in the 3D point sets corresponding to the calibration plane. Since it is essential for the algorithm to have accurate plane coefficients, manual selection can provide complete control.

7.6 Conclusions

This work proposes a novel algorithm for an efficient Lidar-stereo calibration using a single frame of Lidar data and the stereo camera data (3D points estimated from the stereo camera). In this work, we propose estimating the plane coefficients from

both the Lidar and the stereo camera data. From the computed plane coefficients (from both the sensor's data), we construct a well-spaced 3D point structure. Later, we propose to use our methodology called CoSM ICP to compute the transformation between the 'structured' points, thereby accomplishing the purpose of Lidar-stereo calibration. CoSM ICP maintains one-to-one relationship between each point in the *Source* dataset and the *Target* dataset. CoSM ICP is also robust to large rotations and translations, which makes it the right choice for this approach to be implemented in this work. One of the primary challenges we still face is the efficient detection of plane coefficients from the 3D points of the stereo data. We still face failure in estimation if we have inadequate data from the stereo point cloud.

Chapter 8

Single Frame Lidar-Camera Calibration Using Registration of 3D planes

8.1 Motivation

This work focuses on finding the extrinsic parameters (rotation and translation) between the Lidar and an RGB camera sensor. We use a planar checkerboard and place it inside the Field-of-View (FOV) of both the sensors, where we extract the 3D plane information of the checkerboard acquired from the sensor's data. The plane coefficients extracted from the sensor's data are used to construct a well-structured set of

3D points. These 3D points are then 'aligned,' which gives the relative transformation between the two sensors. We use our proposed method Correntropy Similarity Matrix Iterative Closest Point (CoSM-ICP) Algorithm, to estimate the relative transformation. This work uses a single frame of the point cloud data acquired from the Lidar sensor and a single frame from a calibrated camera data to perform this operation. From the camera image, we use projection of the calibration target's corner points to compute the 3D points, and along the process, we calculate the 3D plane equation using the corner points. We evaluate our approach on a simulated dataset with complex environment settings since it has the freedom to assess under multiple configurations. Through results, we verify our method under various configurations.

Lidar and camera calibration is a well-studied problem, and most of the previous approaches require calibration cards or some well-defined calibration objects [96–103]. Fig. 8.1 shows a simple Lidar camera configuration setup. A calibration card is placed in the Field-of-View (FOV) of both sensors. Extracting features from a checkerboard is convenient, making a planar checkerboard one of the most widely used calibration objects. Most of these approaches have emphasized placing the calibration objects in the FOV of both sensors. Finding the correspondences between the Lidar and the camera data points is crucial for efficient Lidar and camera calibration[122]. The correspondences are calculated either manually or automatically using feature extraction algorithms. The accurate estimation of the correspondences is essential for efficient calibration.

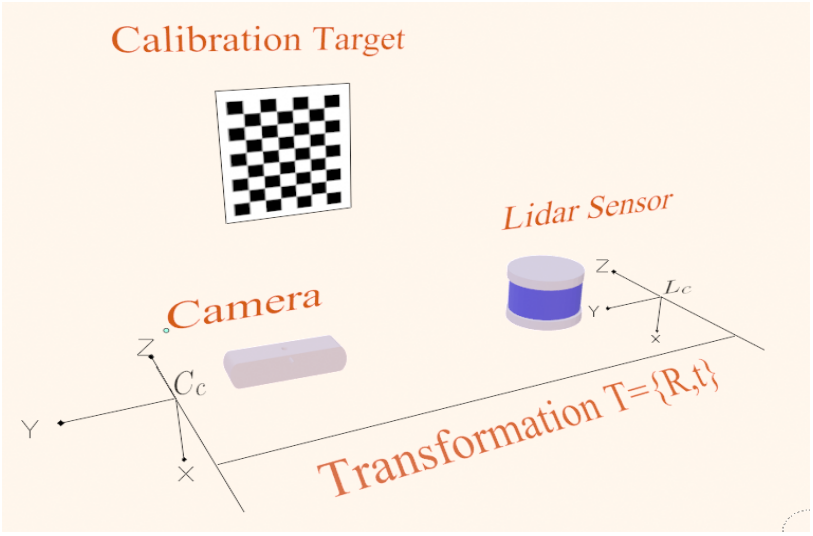


FIGURE 8.1: Sample Lidar and camera configuration setup (It is a stereo camera setup, however we use only the images from the left camera for our work). C_c is the camera coordinate frame (here we consider the left camera center of the stereo camera's coordinate frame), and L_c is the Lidar coordinate frame. The objective here is to compute the transformation T between L_c and C_c.

8.2 Background

The earlier method proposed by Zhang and Pless [97] aimed to calibrate a 2D laser and a monocular camera, and they used a planar checkerboard pattern to achieve that. Their approach involved placing the checkerboard in front of the 2D laser and the camera with different poses. The relative transformation is calculated by minimizing the re-projection error of features of the checkerboard as viewed by the 2D Lidar and the camera. Their algorithm required at least five poses, whereas, in theory, the extrinsic calibration could be achieved in 3 poses. Unnikrishnan and Hebert [104] extended this method to a 3D laser, in which they estimated the plane parameters from the 3D Lidar and the camera data. In their approach, the initial rotation and translation were calculated using plane-plane correspondence and later refined by minimizing the point to plane distance. Another work proposed by Geiger et al.

[105] placed several checkerboards on the scene where their proposed algorithm detects the checkerboards and matches them with the planes from the laser data. Since there were several checkerboards in the scene, one shot of the scene is enough. Most of these methods involve the use of plane information of the checkerboard and do not consider the boundary of the checkerboard. Work by [106] requires fewer poses of the calibration target and has shown accurate results. In their work, the checkerboard is placed closer to the sensor, allowing the laser point to approximate the boundary more accurately, and hence it showed high accuracy. The authors introduced a similarity transformation between the Lidar and the camera, which showed better results than rigid transformation.

Apart from the planar checkerboard, researchers also used several other calibration objects for the calibration method [123–126]. Work proposed in [100] used a polygonal board, and the calibration parameters were estimated using the vertices of the board. However, the camera frame involved manual labelling of the correspondences. These correspondences were used to estimate the calibration parameters using a genetic algorithm-based approach. Other approaches [101–103] have used planar boards with rectangular holes, planar boards with circular holes, and other 3D objects. One of the drawbacks of using all the above methods is that it is laborious and time-consuming.

In addition to the approaches mentioned above, several target-less or no object-based calibration approaches have been proposed as well [107, 108, 127]. Initial work by

Scaramuzza [109] has introduced a technique of using natural scenes to calibrate a 3D laser and an omnidirectional camera. The features were extracted automatically from the sensor data based on their algorithm, and they manually selected the correspondence between the features. Another work by Moghadam [110] exploits the linear features present in the indoor environment. It uses the 3D line features from the point cloud and the corresponding 2D line segment from the camera to estimate the rigid body transformation between the two. Boughorbal [111] proposed to maximize the correlation between the sensor data using a chi-square test. This technique exploits the statistical dependence of the data acquired from the two sensors. Levinson and Thrun [112] uses a series of corresponding laser scans and camera images to estimate the calibration parameters. Work by [113] uses similar mutual information metrics between the two sensors' data by measuring the statistical dependence between the data [114].

Pandey, [113], proposed a theoretical derivation that estimates the kernel density of the probability distribution of the sensor data. Scott [114] proposed an approach for automatic calibration using diligently selected natural scenes. This algorithm searches over the chosen locations to extract models and yields better results. One advantage of this approach is that it requires no knowledge of the physical world and continuously finds scenes that constraints the optimization parameters. One recent and exciting approach by Jeong [115] exploits known features such as road markings. These features are captured by both the Lidar and the stereo sensors and use a multiple cost function for robust optimization even with rough initial values. Work

by Jeong [115] uses the road as a reference for computing transformation. However, it is required to be done in a controlled environment. It is also time-consuming and requires significant effort to obtain the transformation parameters.

In this work, we propose a simple process to perform the Lidar and camera calibration using a checkerboard calibration target. Our work is similar to the work presented by [116], where we compute the plane coefficients using the data from both the Lidar and the camera data. Later on, these coefficients are used to construct a well-structured set of 3D points residing in that plane. We use our own proposed Correntropy Similarity Matrix (CoSM ICP) approach to align the constructed set of 3D points and compute the relative transformation between the sensors. In essence, the significant contributions of our work are outlined as follows:

- We develop our algorithm to find planes of the checkerboard pattern acquired from both the Lidar and the camera sensor's data.
- We propose to compute the plane coefficients separately from both the sensor's data.
- From the plane coefficients acquired from both the sensors, we determine the plane's location and populate the plane with structured data points.
- Our proposed algorithm only needs to populate a limited number of points for the plane to plane matching.

- We leverage our CoSM ICP approach to perform the alignment and find the relative transformation.

8.3 Proposed Methodology

In this section, we describe the steps involved in our Lidar-camera calibration. We perform different steps corresponding to the data acquired from the 3D Lidar sensor and the camera data. The overall procedure involved in our work is similar to the work as mentioned in the previous chapter in Fig. 7.2. However, we only use the data from the left camera to generate the 3D points. We acquire only a single frame of the Lidar's 3D data and the camera's 2D image data, which contains the calibration target in its view. For the 3D Lidar sensor, we manually select the region containing the 3D points corresponding to the calibration target (checkerboard). For the camera's 2D data, we assume that the camera is calibrated, which means that the camera's intrinsic parameters are already known. We project the 2D corner points to 3D points in the world frame. We discuss this process more in subsection 8.3.2. It was convenient for us to use the same calibration target for performing the camera calibration as well. Again, the key point is to determine the plane coefficients acquired from Lidar's 3D data and the camera's projected 3D points. We compute the plane coefficients for the manually selected region of the Lidar data using Random Sample Consensus (RANSAC). Using the camera's data, we compute the corner points of the checkerboard. We project the corner points in 3D space in the world frame using the

intrinsic and extrinsic parameters, and we compute the plane coefficients of these 3D projected corner points. We construct a well-structured set of points using the plane coefficients computed independently from the Lidar and the camera data. We use our CoSM ICP approach to compute the transformation between these constructed point sets. The transformation computed by the CoSM ICP returns the relative transformation between the Lidar and the camera sensors.

8.3.1 Lidar data processing

Using a single calibration target is relatively easy and intuitive for an extrinsic calibration between multiple sensors. As mentioned earlier, the first step of our process is to capture the 3D points of the Lidar data. From the 3D data, we manually pick the region containing the points corresponding to the calibration target and ignore the rest. This selected data contains points of the calibration target, which is effectively a plane. Other automatic approaches, like distance filtering, can be used for this process, but we let the user pick the region for complete control. Now, from this 'selected' point set, we intend to get the corresponding plane coefficients. RANSAC is our choice for this process, which determines the best-fit plane of 3D points using inliers. We use the following steps for our work.

- Randomly selects 3 points from the 'selected' 3D Lidar points.
- Compute the plane equation using these 3 points.
- Compute inliers using the computed plane with all other 3D points.

- Repeat the process with the highest inlier ratio.

For this setup, we set the maximum iterations for our RANSAC algorithm as 1000. 3D points that are within 10mm from the plane are considered inliers. The inlier ratio, which crosses 90% or more, is considered as a best-fit plane. The plane equation computed from the Lidar points is given as $a_l x + b_l y + c_l z + d_l = 0$, where a_l,b_l,c_l and d_l represent the coefficients of the plane.

8.3.2 Camera data processing

This section details the process of computing the 3D plane equations of the calibration target using the 2D image data acquired from the camera. In this work, we assume that the camera is already calibrated. We know the intrinsic parameters $\alpha_x, \alpha_y, x_0, y_0$, where α_x,α_y represents the camera's focal length in terms of pixel dimensions and x_0, y_0 represent the principal point in the pixel dimensions. The parameters are described in the matrix form and are denoted as K. We ignore the skew, radial, and tangential distortion since we operate in a simulated environment. The chessboard corners are computed using OpenCV $findChessboardCorners$ function, which is based on the algorithm by Duda and Frese [128]. After the corner point detection, we use the well-known PnP algorithm to compute the pose of the calibration object from 3D-2D point correspondences. The result of PnP gives us the rotation and the translation components that transform a 3D point expressed in the object coordinate frame to the camera coordinate frame. For this method, we use an iterative method which is based on the Levenberg-Marquardt optimization method that minimizes the

reprojection error. From the set of 3D computed corner points of the checkerboard pattern, we select three random points and transform them from the object frame to the camera frame using the rotation and the translation components computed from the PnP method. Let the 3 randomly selected be $p_1 = \{x_1, y_1, z_1\}$,$p_2 = \{x_2, y_2, z_2\}$ and $p_3 = \{x_3, y_3, z_3\}$. We compute the normal vector $\vec{n}$ as,

$$\vec{n} = (p_2 - p_1) \times (p_3 - p_1) \tag{8.1}$$

where, $\times$ is the cross product in Equ.(8.1). The coefficients of the plane is then computed as,

$$\begin{aligned} a_c &= n.x, \\ b_c &= n.y, \\ c_c &= n.z, \\ d_c &= -(a * p_1.x + b * p_1.y + c * p_1.z). \end{aligned} \tag{8.2}$$

In Equ.(8.2), $*$ denotes multiplication and . represents the individual components in a vector. The plane equation computed from the camera points is given as $a_c x + b_c y + c_c z + d_c = 0$, where a_c, b_c, c_c and d_c represent the coefficients of the plane.

8.3.3 Transformation estimation

We compute the transformation between these sensors from the plane equations calculated from both the Lidar and the camera data. For this step, we use the calculated

plane equations to populate the planes with a fixed number of points separated by a known distance (e.g., 100 points). The point sets generated from this process can be 'aligned.' Fig.8.2 shows the structured set of points. The computed transformation gives us the transformation between the Lidar and the camera sensor. The primary reason for this process is that the number of points in the Lidar point set is different from the number of points in the computed 3D points from the camera. Our CoSM ICP needs to have an equal number of points for alignment. We describe our CoSM ICP approach to compute the transformation between the Lidar and the camera data points.

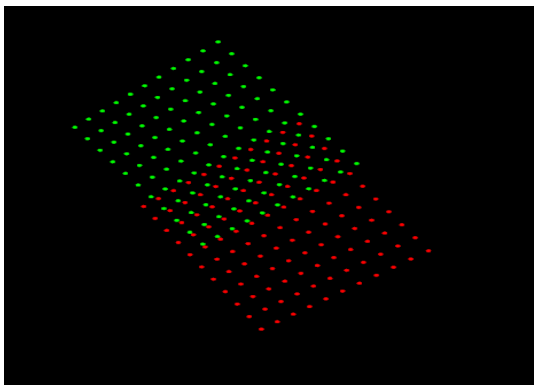


FIGURE 8.2: Green is the *Source* point cloud $\mathbf{P_s}$ (as computed from Lidar points). $\mathbf{P_s}$ contains N points $\mathbf{p}_j$ $(j = 1, .., N)$, each of which is a 3D representation x_j, y_j, z_j. White is the *Target* point cloud $\mathbf{P_t}$ (as computed from camera data). $\mathbf{P_t}$ contains N points $\mathbf{q_k}$ $(k = 1, .., N)$, each of which is a 3D representation x_k, y_k, z_k.

8.4 Results

We performed our initial evaluation in a simulated environment provided by Open Robotics[120]. To demonstrate our results, we start with a basic simulated dataset

containing simulated Lidar data and a simulated camera setup (mounted on a simulated Prius model car) established in a simulated environment in Gazebo, which is similar to our work in our previous chapter (Fig.7.3 (a) and (b) [121]). The primary reason for selecting this simulation setup is to compare our results since we know the ground truth. This setup contains other environmental complexities like buildings and cars (apart from the calibration card) and simulated lighting conditions. Our experiments test the results with a linear transformation ranging from $0.05m$ to $2.5m$ (along x, y, z axes). We set up a simple test case of Lidar-camera setup where the stereo camera faces the calibration target and is placed near the Lidar sensor under various configurations. (e.g., 5cm along the y axis of the Lidar sensor or $\mathbf{t} = [0, 0.05, 0.0]$). Here, we intend to calculate the transformation between the camera (C_c) with respect to the Lidar's frame L_c. The camera has a resolution of 1280×720, and since we use a simulated setup, we ignore the radial and tangential distortion (both were set to 0). In this scenario, we can evaluate our approach on multiple configurations where the ground truth is already known. So, in this case, our problem statement is defined as finding the transformation between the Lidar(L_c) and the camera(C_c) setup using our proposed methodology. This setup provides a base test case to verify our algorithm. Thus it can extend to complex real-time test cases.

8.4.1 Evaluation on Simulated data

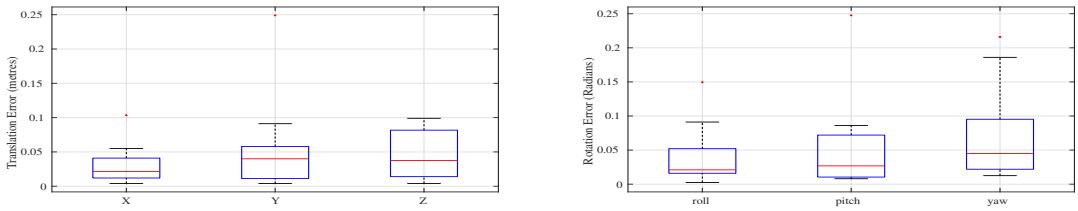
The various configurations under which we performed our experiments are shown in Table 8.1. It shows the original ground truth transformations between the Lidar and the camera sensor. We place the calibration target at an average distance of 2m to 3m from the calibration target throughout our experiments (this distance is with respect to the Lidar coordinate frame (L_c)).

TABLE 8.1: Configuration setups used in our experiments (or Ground Truth).

Setting	$t_x(m)$	$t_y(m)$	$t_z(m)$	roll (rad)	pitch(rad)	yaw(rad)
1	0	0.4	0.0	0.0	0.0	0.0
2	0	0.1	0.0	0.0	0.0	0.0
3	0	0.05	0.0	0.0	0.0	0.0
4	0	0.6	0.0	0.0	0.0	0.0
5	0	1.2	0.0	0.0	0.0	0.0
6	0	2.2	0.0	0.0	0.0	0.0
7	0.5	0.5	0.0	0.0	0.0	0.0
8	−0.5	1.5	0.3	0.0	0.0	0.0
9	−0.5	0.5	0	0.0	0.0	−0.0
10	0.5	0.5	0	0	0	−0.523599
11	0.5	0.5	0.3	0	0.349066	−0.523599
12	0.3	0.6	0.4	0	0.349066	−0.523599
13	0.2	0.3	0.2	0.261799	0	−0.523599
14	0.7	0.2	0.9	0	0.349066	0

For further evaluation, we show the average error in multiple configurations as done in the experiments. Fig.8.3 shows the average error of the individual components of the rotation and translation components under various configurations. From the experiments performed and from Fig. 8.3, we can see that under simple translation along x, y, z axes, our algorithm performs well (with individual RMSE's ~ 0.01 along with all the individual components). Even under small rotations, our approach returns relatively close values (RMSE ~ 0.02) compared to the ground truth. However, when the transformation between C_c and L_c is significant (> 60 degrees or 1.0472

radians), the overall RMSE increases, and the computed transformation is quite far off from the ground truth values (RMSE $>$ 6.891).



(A) (B)

FIGURE 8.3: Translation and Rotation errors of individual components (x, y, z and roll,pitch and yaw) under various configurations compared to ground truth.

8.5 Summary

One of the critical observations here was to estimate 3D points computed from the camera setup accurately. Throughout the experiments, we collected datasets for all the individual configurations as mentioned in Table 8.1. We hand-picked each dataset that had a good collection of 3D point sets with less visible outliers. However, we hope that with further advancements in 3D depth estimation, we can get more accurate and robust results.

Since it was a simulated setup, it could have been simple to have a Lidar sensor, a camera setup, and a calibration target. However, one of our primary goals was to have a calibration technique in various environments. We plan to evaluate our approach in real-time; however, valuating a Lidar and a camera set up under different configurations is time-consuming. So, in our simulated environment, we add environmental complexities and evaluate our approach. We let the user select the region in

the 3D point sets corresponding to the calibration plane. Since it is essential for the algorithm to have accurate plane coefficients, manual selection can provide complete control.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The goal of this work is to explore the concepts of Correntropy in various applications of SLAM applied in modern robotics. Initially, we briefly described various SLAM techniques and described the challenges faced by the modern SLAM framework when the sensor data gets affected by non-Gaussian noise. We have attempted to classify the SLAM techniques in both Lidar-based odometry and Stereo-based odometry. Handling non-Gaussian outliers have always been a challenge with reference to modern applications in robotics. We provide the conceptual background of Correntropy and then explore its properties of Correntropy. Later on, we describe the simple usage of Correntropy in KF, and through the results, we verify the effectiveness of

Correntropy in handling non-Gaussian noise. We extend that to EKF with application in fusing Lidar and stereo sensor odometry and verifying the robustness of the system when one of the odometry returned is affected by a third-party attacker. The highlight of the work is the application of using Correntropy in addressing the problem of Registration. We verify through results how well our application of using the Correntropy matrix is useful in handling non-Gaussian noise as well as handling large rotations and translation between the *Source* and *Target* datasets. This resulted in the production of the CoSM ICP approach, which we later used in Lidar-Stereo and Lidar Camera Calibration. In brief, the highlight of our work is described below.

- Review of SLAM techniques
  - We reviewed various SLAM techniques that were tried and tested on autonomous driving cars based on the KITTI dataset [57]. We risked an attempt to classify various SLAM techniques, which are based on sensors used for localization and the ability of the SLAM algorithms to detect a loop closure.
  - we have also attempted to describe the security vulnerabilities in the autonomous driving systems as well as describe various types of attacks that have been introduced and demonstrated by various researchers
- Correntropy Concepts

  - We describe the concepts of Correntropy and also give a brief description of the properties of Correntropy.

  - We explore the probabilistic concepts and geometric meaning of Correntropy with reference to connection with M-estimation, ITL and kernel methods.

  - As a measure of similarity, we use Correntropy that directs the probability density of how close are two random variables as well as show the advantage of using Correntropy in non-gaussian signal processing.

- Correntropy with KF and EKF

  - we extensively reviewed and discussed the KF to better understand how it works and how it can be modified to take into account higher-order signal statistics.

  - This addition of Correntropy lets us use higher-order statistics to improve state estimation. In a preliminary phase, we run a total of five simulations in which, each time, we introduce a random shot noise to the lidar measurements to prove that the standard formulation of the KF does not behave as well as in the presence of Gaussian noise

  - we have also attempted to provide a self-secure solution to an autonomous system using the MCC-EKF approach. From the results, we have shown how an autonomous system can be attacked by an attacker/hacker and

change the system's naturally estimated trajectory and how even a simple injection of false positions can affect the overall trajectory of the autonomous system. In addition, we have also proposed that we can also get a better estimate of the odometry by fusing the odometry data from two different SLAM algorithms to obtain a better odometry estimate of the autonomous system.

- Correntropy with Registration
  - We have addressed is to find an accurate estimate of the transformation between the *Source* and the *Target* point clouds under various rotations and translations.
  - We have verified our approach in various datasets, where the *Source* is transformed from the *Target* using various randomly generated transformation matrices.
  - We can see through the obtained results that our method has performed better than the other state-of-the-art approaches. In addition, we also evaluate our approach when the *Source* dataset is affected by noise generated with random intensity and affecting random data points.
  - We have verified that our proposed approach has outperformed most of the other state-of-the-art approaches where the 'infected' *Source* dataset is transformed randomly from the *Target* dataset. We firmly insist that

our readers evaluate our approach located in our Github repository under the name of CoSM-ICP.

- Lidar-Stereo and Lidar-Camera calibration
  - We have proposed a novel algorithm for an efficient lidar-stereo calibration using a single frame of lidar data and the stereo camera data (3D points estimated from the stereo camera).
  - We estimate the plane coefficients from both the Lidar and the stereo camera data, and from the computed plane coefficients (from both the sensor's data), we construct a well-spaced 3D point structure.
  - We also propose to use our methodology CoSM ICP as described earlier to compute the transformation between the 'structured' points, thereby accomplishing the purpose of lidar-stereo calibration. CoSM ICP maintains one-to-one relationship between each point in the *Source* dataset and the *Target* dataset. CoSM ICP is also robust to large rotations and translations, which makes it the right choice for this approach to be implemented in this work.

9.2 Future Work

Regarding the security aspect of SLAM, our future work will focus on proposing a solution where we can secure the system if the attacker chooses to inject false data on

the sensor's raw measurements. We also plan to extend this work to distributed MCC-EKF security for vehicle to vehicle network in which multi-robot system research [2, 61–75] can be utilized.

Regarding the calibration techniques we proposed, we intend to improve our approach in Single frame Lidar-Camera calibration since the current results require further tuning in terms of accuracy. In addition to evaluating our results in the simulated datasets, we also focus on extending our work to real-time datasets. We further plan to validate our approach for calibration under various lighting conditions in real-time.

Regarding Lidar Based SLAM, we also intend to extend our future research on improving it using the CoSM ICP approach and then the graph-based SLAM approach to optimize the map. Our work is focused on expanding to the well-known KITTI dataset and other challenging datasets (both simulated and real-time). The graph-based approach in SLAM provides better accuracy than the traditional Filtering based strategies in SLAM at the cost of high computation. In addition to it, we also intend to explore the idea of correntropy in the graph-based approaches and validate the performance of including and validating the deep learning approach to SLAM since it provides promising results for data analysis.

List of Authored Publications

- A. Singandhupe, H. M. La, D. Feil-Seifer, P. Huang, L. Guo, and M. Li. Securinga uav using individual characteristics from an eeg signal. InIEEE Intern. Conf.on Systems, Man, and Cybernetics (SMC), 2018.

- A. Singandhupe and H. M. La. A review of slam techniques and security inautonomous driving. In2019 Third IEEE International Conference on Robotic-Computing (IRC), pages 602–607, Feb 2019. doi: 10.1109/IRC.2019.00122.

- Adarsh Sehgal, Ashutosh Singandhupe, Hung Manh La, Alireza Tavakkoli, and-Sushil J. Louis. Lidar-monocular visual odometry with genetic algorithm for pa-rameter optimization. In George Bebis, Richard Boyle, Bahram Parvin, DarkoKoracin, Daniela Ushizima, Sek Chai, Shinjiro Sueda, Xin Lin, Aidong Lu,Daniel Thalmann, Chaoli Wang, and Panpan Xu, editors,Advances in Visual Computing, pages 358–370, Cham, 2019. Springer International Publishing.ISBN 978-3-030-33723-0.

- A. Singandhupe and H. M. La. Mcc-ekf for autonomous car security.Proceedingsof the 4th IEEE International Conference on Robotic Computing (IRC), Nov2020.

- Ashutosh Singandhupe, Hung M. La, Trung Dung Ngo, and Van Anh Ho.Registration of 3d point sets using correntropy similarity matrix.CoRR,abs/2107.09725, 2021. URLhttps://arxiv.org/abs/2107.09725

- K. Shrestha, R. Dubey, A. Singandhupe, S. Louis and H. La, "Multi Objective UAV Network Deployment for Dynamic Fire Coverage," 2021 IEEE Congress on Evolutionary Computation (CEC), 2021, pp. 1280-1287, doi: 10.1109/CEC45853.2021.9504947.

Bibliography

[1] 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 Trans. on Robotics*, 32(6): 1309–1332, Dec 2016. ISSN 1552-3098. doi: 10.1109/TRO.2016.2624754.

[2] Devin Connell and Hung Manh La. Extended rapidly exploring random tree based dynamic path planning and replanning for mobile robots. *Intern. J. of Advanced Robotic Systems*, 15(3):1729881418773874, 2018. doi: 10.1177/1729881418773874. URL `https://doi.org/10.1177/1729881418773874`.

[3] Badong Chen, Xi Liu, Haiquan Zhao, and Jose C. Principe. Maximum correntropy kalman filter. *Automatica*, 76:70–77, 2017. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2016.10.004. URL `http://www.sciencedirect.com/science/article/pii/S000510981630396X`.

[4] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Trans. on Intel. Transportation Systems Magazine*, 2:31–43, 12 2010. doi: 10.1109/MITS.2010.939925.

[5] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Trans. on Robotics*, 33(5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103.

[6] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Trans. on Robotics*, 31(5):1147–1163, 2015. doi: 10.1109/TRO.2015.2463671.

[7] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. CNN-SLAM: real-time dense monocular SLAM with learned depth prediction. abs/1704.03489, 2017. URL `http://arxiv.org/abs/1704.03489`.

[8] J. Morris. The kalman filter: A robust estimator for some classes of linear quadratic problems. *IEEE Transactions on Information Theory*, 22(5):526–534, 1976. doi: 10.1109/TIT.1976.1055611.

[9] Zongze Wu, Jiahao Shi, Xie Zhang, Wentao Ma, and Badong Chen. Kernel recursive maximum correntropy. *Signal Processing*, 117:11–16, 2015. ISSN 0165-1684. doi: https://doi.org/10.1016/j.sigpro.2015.04.024. URL `https://www.sciencedirect.com/science/article/pii/S0165168415001590`.

[10] Weifeng Liu, Puskal P. Pokharel, and Jose C. Principe. Correntropy: Properties and applications in non-gaussian signal processing. *IEEE Transactions on Signal Processing*, 55(11):5286–5298, 2007. doi: 10.1109/TSP.2007.896065.

[11] Badong Chen, Lei Xing, Junli Liang, Nanning Zheng, and José C. Príncipe. Steady-state mean-square error analysis for adaptive filtering under the maximum correntropy criterion. *IEEE Signal Processing Letters*, 21(7):880–884, 2014. doi: 10.1109/LSP.2014.2319308.

[12] H.W. Sorenson and D.L. Alspach. Recursive bayesian estimation using gaussian sums. *Automatica*, 7(4):465–479, 1971. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(71)90097-5. URL `https://www.sciencedirect.com/science/article/pii/0005109871900975`.

[13] Andrew Harvey and Alessandra Luati. Filtering with heavy tails. *Journal of the American Statistical Association*, 109(507):1112–1122, 2014. doi: 10.1080/01621459.2014.887011. URL `https://doi.org/10.1080/01621459.2014.887011`.

[14] D. Magill. Optimal adaptive estimation of sampled stochastic processes. *IEEE Transactions on Automatic Control*, 10(4):434–439, 1965. doi: 10.1109/TAC.1965.1098191.

[15] Igal Bilik and Joseph Tabrikian. Mmse-based filtering in presence of non-gaussian system and measurement noise. *IEEE Transactions on Aerospace and Electronic Systems*, 46(3):1153–1170, 2010. doi: 10.1109/TAES.2010.5545180.

[16] Arnaud Doucet, N. Freitas, and N.J Gordon. *Sequential Monte-Carlo Methods in Practice*, volume 1. 01 2001. ISBN 978-1-4419-2887-0. doi: 10.1007/978-1-4757-3437-9.

[17] Gabriel Agamennoni, Juan I. Nieto, and Eduardo M. Nebot. An outlier-robust kalman filter. In *2011 IEEE International Conference on Robotics and Automation*, pages 1551–1558, 2011. doi: 10.1109/ICRA.2011.5979605.

[18] J. Deschaud. Imls-slam: Scan-to-model matching based on 3d data. In *2018 IEEE Intern. Conf. on Robotics and Automation (ICRA)*, pages 2480–2485, May 2018. doi: 10.1109/ICRA.2018.8460653.

[19] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low drift, robust, and fast. In *IEEE Intern. Conf. on Robotics and Automation (ICRA)*, Seattle, WA, May 2015.

[20] Ji Zhang and Sanjiv Singh. LOAM: Lidar odometry and mapping in real- time. In *Robotics: Science and Systems Conf.*, Berkeley, CA, July 2014.

[21] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[22] Johannes Graeter, Alexander Wilczynski, and Martin Lauer. Limo: Lidar-monocular visual odometry. *arXiv preprint arXiv:1807.07524*, 2018.

[23] Ji Zhang, Michael Kaess, and Sanjiv Singh. Real-time depth enhanced monocular odometry. In *IEEE/RSJ Intern. Conf. on Intel. Robots and Systems (IROS)*, Chicago, IL, Sept. 2014.

[24] Jianke Zhu. Image gradient-based joint direct visual odometry for stereo camera. In *Intern. Joint Conf. on Artificial Intelligence, IJCAI*, 2017.

[25] Tim Y Tang, David J Yoon, and Timothy D Barfoot. A white-noise-on-jerk motion prior for continuous-time trajectory estimation on se (3). *arXiv preprint arXiv:1809.06518*, 2018.

[26] Kaijin Ji, Huiyan Chen, Huijun Di, Jianwei Gong, Guangming Xiong, Jianyong Qi, and Tao Yi. Cpfg-slam:a robust simultaneous localization and mapping based on lidar in off-road environment. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 650–655, 2018. doi: 10.1109/IVS.2018.8500599.

[27] N. Yang, R. Wang, J. Stueckler, and D. Cremers. Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry. In *European Conf. on Computer Vision*, September 2018.

[28] Frank Neuhaus, Tilmann Koss, Robert Kohnen, and Dietrich Paulus. Mc2slam: Real-time inertial lidar odometry using two-scan motion compensation. In *German Conf. on Pattern Recognition*. Springer, 2018. To appear.

[29] Martin Buczko and Volker Willert. How to distinguish inliers from outliers in visual odometry for high-speed automotive applications. In *IEEE Intel. Vehicles Symposium*, 2016. URL `https://www.researchgate.net/publication/305709493_How_to_Distinguish_Inliers_from_Outliers_in_Visual_Odometry_for_High-speed_Automotive_Applications`.

[30] Igor Cvišić and Ivan Petrović. Stereo odometry based on careful feature selection and tracking. In *European Conf. on Mobile Robots (ECMR)*, 2015.

[31] Tim Y Tang, David J Yoon, François Pomerleau, and Timothy D Barfoot. Learning a Bias Correction for Lidar- only Motion Estimation. *15th Conf. on Computer and Robot Vision (CRV)*, 2018.

[32] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. *Inter. J. of Computer Vision*, 9(3):137–154, 1991.

[33] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing ICP Variants on Real-World Data Sets. *Autonomous Robots*, 34(3): 133–148, February 2013.

[34] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *J. of Software Engineering for Robotics (JOSER)*, 3(1):2–12, 2012. ISSN 2035-3928.

[35] J. Behley and C. Stachniss. Efficient surfel-based slam using 3d laser range data in urban environments. In *Robotics: Science and Sys.*, 2018.

[36] Igor Cvišić, Josip Ćesić, Ivan Marković, and Ivan Petrović. Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for

autonomous unmanned aerial vehicles. *Journal of Field Robotics*, 35(4):578–595, 2018. doi: 10.1002/rob.21762. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21762`.

[37] J. Engel, J. Stückler, and D. Cremers. Large-scale direct SLAM with stereo cameras. In *Int. Conf. on Intel. Robot Systems (IROS)*, 2015.

[38] Kruno Lenac, Josip Ćesić, Ivan Marković, and Ivan Petrović. Exactly sparse delayed state filter on lie groups for long-term pose graph slam. *The Intern. J. of Robotics Research*, 37(6):585–610, 2018. doi: 10.1177/0278364918767756.

[39] Martin Buczko, Volker Willert, Julian Schwehr, and Jürgen Adamy. Self-validation for automotive visual odometry. In *IEEE Intel. Vehicles Symp.*, 2018.

[40] Fabio Bellavia, Marco Fanfani, Fabio Pazzaglia, and Carlo Colombo. *Robust Selective Stereo SLAM without Loop Closure and Bundle Adjustment*. Springer, 2013.

[41] D. Schlegel, M. Colosi, and G. Grisetti. ProSLAM: Graph SLAM from a Programmer's Perspective. In *2018 IEEE Intern. Conf. on Robotics and Automation (ICRA)*, pages 1–9, 2018.

[42] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles. S-PTAM: Stereo Parallel Tracking and Mapping. *Robotics and Autonomous Systems (RAS)*, 93:27 – 42, 2017. ISSN 0921-8890. doi: 10.1016/j.robot.2017.03.019.

[43] Taihú Pire, Thomas Fischer, Javier Civera, Pablo De Cristóforis, and Julio Jacobo berlles. Stereo Parallel Tracking and Mapping for robot localization. In *Proc. of the Intern. Conf. on Intel. Robots and Systems (IROS)*, pages 1373–1378, September 2015. doi: 10.1109/IROS.2015.7353546.

[44] Shiyu Song and Manmohan Chandraker. Robust scale estimation in real-time monocular sfm for autonomous driving. In *CVPR*, Columbus, Ohio, USA, 24-27, 2014.

[45] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intel. Vehicles Symposium (IV)*, 2011.

[46] Bernd Kitt, Andreas Geiger, and Henning Lategahn. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In *Intel. Vehicles Symposium (IV)*, 2010.

[47] Shiyu Song, Manmohan Chandraker, and Clark C. Guest. Parallel, real-time monocular visual odometry. In *ICRA*, Karlsruhe, Germany, May6-10, 2013.

[48] Daniel J Mankowitz and Ehud Rivlin. CFORB: Circular FREAK-ORB Visual Odometry. *arXiv preprint arXiv:1506.05257*, 2015.

[49] Eray Yağdereli, Cemal Gemci, and A Ziya Aktaş. A study on cyber-security of autonomous and unmanned vehicles. *The J. of Defense Modeling and Simulation*, 12(4):369–381, 2015. doi: 10.1177/1548512915575803.

[50] A. Singandhupe, H. M. La, and D. Feil-Seifer. Reliable security algorithm for drones using individual characteristics from an eeg signal. *IEEE Access*, 6(1): 2–12, 2018.

[51] A. Singandhupe, H. M. La, D. Feil-Seifer, P. Huang, L. Guo, and M. Li. Securing a uav using individual characteristics from an eeg signal. In *IEEE Intern. Conf. on Systems, Man, and Cybernetics (SMC)*, 2018.

[52] Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77: 167–181, 2015. ISSN 0965-8564.

[53] A. M. Wyglinski, X. Huang, T. Padir, L. Lai, T. R. Eisenbarth, and K. Venkatasubramanian. Security of autonomous systems employing embedded computing and sensors. *IEEE Micro*, 33(1):80–86, Jan 2013. ISSN 0272-1732. doi: 10.1109/MM.2013.18.

[54] Reza Izanloo, Seyed Abolfazl Fakoorian, Hadi Sadoghi Yazdi, and Dan Simon. Kalman filtering based on the maximum correntropy criterion in the presence of non-gaussian noise. In *2016 Annual Conference on Information Science and Systems (CISS)*, pages 500–505, 2016. doi: 10.1109/CISS.2016.7460553.

[55] Seyed Fakoorian, Alireza Mohammadi, Vahid Azimi, and Dan Simon. Robust Kalman-Type Filter for Non-Gaussian Noise: Performance Analysis With Unknown Noise Covariances. *Journal of Dynamic Systems, Measurement, and Control*, 141(9), 05 2019. ISSN 0022-0434. doi: 10.1115/1.4043054. URL `https://doi.org/10.1115/1.4043054`.

[56] David Brown. Here's how hackers are making your tesla, gm and chrysler less vulnerable to attack. https://www.usatoday.com/story/tech/2019/07/05/tesla-gm-hire-hackers-make-your-connected-car-safer/1594717001/, 2019.

[57] KITTI Dataset, available at. `http://www.cvlibs.net/datasets/kitti/`. Accessed: 2019-01-10.

[58] OSRF. car_demo. `https://github.com/osrf/car_demo`, 2019.

[59] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[60] Michael Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017.

[61] H. D. Pham, H. M. La, and T. D. Ngo. Adaptive hierarchical distributed control with cooperative task allocation for robot swarms. In *The 12th IEEE/SICE International Symposium on System Integration (SII), Hawaii, USA.*, pages 1–6, Jan 2020.

[62] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In *2007 46th IEEE Conference on Decision and Control*, pages 5492–5498, Dec 2007. doi: 10.1109/CDC.2007.4434303.

[63] Long Jin, Shuai Li, Hung Manh La, Xin Zhang, and Bin Hu. Dynamic task allocation in multi-robot coordination for moving target tracking: A distributed approach. *Automatica*, 100:75–81, 2019. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2018.11.001. URL `http://www.sciencedirect.com/science/article/pii/S0005109818305338`.

[64] Anh Duc Dang, Hung Manh La, Thang Nguyen, and Joachim Horn. Formation control for autonomous robots with collision and obstacle avoidance using a rotational and repulsive force based approach. *International Journal of Advanced Robotic Systems*, 16(3):1729881419847897, 2019. doi: 10.1177/1729881419847897. URL `https://doi.org/10.1177/1729881419847897`.

[65] M. T. Nguyen, H. M. La, and K. A. Teague. Collaborative and compressed mobile sensing for data collection in distributed robotic networks. *IEEE Transactions on Control of Network Systems*, 5(4):1729–1740, Dec 2018. ISSN 2372-2533. doi: 10.1109/TCNS.2017.2754364.

[66] T. Nguyen, H. M. La, T. D. Le, and M. Jafari. Formation control and obstacle avoidance of multiple rectangular agents with limited communication ranges. *IEEE Transactions on Control of Network Systems*, 4(4):680–691, Dec 2017. ISSN 2372-2533. doi: 10.1109/TCNS.2016.2542978.

[67] T. Nguyen and H. M. La. Distributed formation control of nonholonomic mobile robots by bounded feedback in the presence of obstacles. In *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 206–211, July 2017. doi: 10.1109/RCAR.2017.8311861.

[68] T. Nguyen, T. Han, and H. M. La. Distributed flocking control of mobile robots by bounded feedback. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 563–568, Sep. 2016. doi: 10.1109/ALLERTON.2016.7852281.

[69] H. M. La, W. Sheng, and J. Chen. Cooperative and active sensing in mobile sensor networks for scalar field mapping. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):1–12, Jan 2015. ISSN 2168-2232. doi: 10.1109/TSMC.2014.2318282.

[70] H. M. La, R. Lim, and W. Sheng. Multirobot cooperative learning for predator avoidance. *IEEE Transactions on Control Systems Technology*, 23(1):52–63, Jan 2015. ISSN 2374-0159. doi: 10.1109/TCST.2014.2312392.

[71] H. M. La and W. Sheng. Distributed sensor fusion for scalar field mapping using mobile sensor networks. *IEEE Transactions on Cybernetics*, 43(2):766–778, April 2013. ISSN 2168-2275. doi: 10.1109/TSMCB.2012.2215919.

[72] H. M. La and W. Sheng. Flocking control of multiple agents in noisy environments. In *2010 IEEE International Conference on Robotics and Automation*, pages 4964–4969, May 2010. doi: 10.1109/ROBOT.2010.5509668.

[73] A. D. Dang, H. M. La, and J. Horn. Distributed formation control for autonomous robots following desired shapes in noisy environment. In *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 285–290, Sep. 2016. doi: 10.1109/MFI.2016.7849502.

[74] H. X. Pham, H. M. La, D. Feil-Seifer, and M. C. Deans. A distributed control framework of multiple unmanned aerial vehicles for dynamic wildfire tracking. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–12, 2018. ISSN 2168-2232. doi: 10.1109/TSMC.2018.2815988.

[75] H. M. La and W. Sheng. Flocking control of a mobile sensor network to track and observe a moving target. In *2009 IEEE International Conference on Robotics and Automation*, pages 3129–3134, May 2009. doi: 10.1109/ROBOT.2009.5152747.

[76] A. Singandhupe and H. M. La. A review of slam techniques and security in autonomous driving. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 602–607, Feb 2019. doi: 10.1109/IRC.2019.00122.

[77] Ashutosh Singandhupe, Hung M. La, Trung Dung Ngo, and Van Anh Ho. Registration of 3d point sets using correntropy similarity matrix. *CoRR*, abs/2107.09725, 2021. URL `https://arxiv.org/abs/2107.09725`.

[78] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, May 1987.

ISSN 0162-8828. doi: 10.1109/TPAMI.1987.4767965. URL https://doi.org/10.1109/TPAMI.1987.4767965.

[79] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[80] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration. 01 2004.

[81] Keisuke Tateno, Federico Tombari, Iro Laina, and N. Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6565–6574, 2017.

[82] Adarsh Sehgal, Ashutosh Singandhupe, Hung Manh La, Alireza Tavakkoli, and Sushil J. Louis. Lidar-monocular visual odometry with genetic algorithm for parameter optimization. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Daniela Ushizima, Sek Chai, Shinjiro Sueda, Xin Lin, Aidong Lu, Daniel Thalmann, Chaoli Wang, and Panpan Xu, editors, *Advances in Visual Computing*, pages 358–370, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33723-0.

[83] Christopher Choy, Wei Dong, and Vladlen Koltun. Deep global registration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[84] Heng Yang, J. Shi, and L. Carlone. Teaser: Fast and certifiable point cloud registration. *ArXiv*, abs/2001.07715, 2020.

[85] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 766–782, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46475-6.

[86] A. Singandhupe and H. M. La. Mcc-ekf for autonomous car security. *Proceedings of the 4th IEEE International Conference on Robotic Computing (IRC)*, Nov 2020.

[87] Zongze Wu, Hongchen Chen, Shaoyi Du, Minyue Fu, Nan Zhou, and Nanning Zheng. Correntropy based scale icp algorithm for robust point set registration. *Pattern Recognition*, 93:14–24, 2019. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2019.03.013. URL `http://www.sciencedirect.com/science/article/pii/S0031320319301116`.

[88] Xuetao Zhang, Libo Jian, and Meifeng Xu. Robust 3d point cloud registration based on bidirectional maximum correntropy criterion. *PloS one*, 13(5): e0197542, 2018.

[89] Guanglin Xu, Shaoyi Du, Dixiao Cui, Sirui Zhang, Badong Chen, Xuetao Zhang, Jianru Xue, and Yue Gao. Precise point set registration using point-to-plane distance and correntropy for lidar based localization. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 734–739. IEEE, 2018.

[90] G. Xu, S. Du, D. Cui, S. Zhang, B. Chen, X. Zhang, J. Xue, and Y. Gao. Precise point set registration using point-to-plane distance and correntropy for lidar based localization. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 734–739, 2018. doi: 10.1109/IVS.2018.8500525.

[91] Jose C. Principe. *Information Theoretic Learning: Renyi's Entropy and Kernel Perspectives*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1441915699.

[92] Jose Principe and John Iii. Information-theoretic learning. *Advances in unsupervised adaptive filtering*, 09 2000.

[93] Aleksandr Segal and Dirk Hähnel. Generalized-icp. 06 2009. doi: 10.15607/RSS.2009.V.021.

[94] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2743–2748 vol.3, 2003. doi: 10.1109/IROS.2003.1249285.

[95] Andrew W Fitzgibbon. Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13):1145–1153, 2003. ISSN 0262-8856. doi: https://doi.org/10.1016/j.imavis.2003.09.004. URL `https://www.sciencedirect.com/science/article/pii/S0262885603001835`. British Machine Vision Computing 2001.

[96] Lili Huang and Matthew Barth. A novel multi-planar lidar and computer vision calibration procedure using 2d patterns for automated navigation. In *2009 IEEE Intelligent Vehicles Symposium*, pages 117–122. IEEE, 2009.

[97] Qilong Zhang and R. Pless. Extrinsic calibration of a camera and laser range finder (improves camera calibration). In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2301–2306 vol.3, 2004. doi: 10.1109/IROS.2004.1389752.

[98] O. Naroditsky, A. Patterson, and K. Daniilidis. Automatic alignment of a camera with a line scan lidar system. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3429–3434, 2011. doi: 10.1109/ICRA.2011.5980513.

[99] S. Verma, J. S. Berrio, S. Worrall, and E. Nebot. Automatic extrinsic calibration between a camera and a 3d lidar using 3d point and plane correspondences. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3906–3912, 2019. doi: 10.1109/ITSC.2019.8917108.

[100] Ankit Dhall, Kunal Chelani, Vishnu Radhakrishnan, and K Madhava Krishna. Lidar-camera calibration using 3d-3d point correspondences. *arXiv preprint arXiv:1705.09785*, 2017.

[101] C. Guindel, J. Beltrán, D. Martín, and F. García. Automatic extrinsic calibration for lidar-stereo vehicle sensor setups. In *2017 IEEE 20th International*

Conference on Intelligent Transportation Systems (ITSC), pages 1–6, 2017. doi: 10.1109/ITSC.2017.8317829.

[102] Yoonsu Park, Seokmin Yun, Chee Won, Kyungeun Cho, Kyhyun Um, and Sungdae Sim. Calibration between color camera and 3d lidar instruments with a polygonal planar board. *Sensors (Basel, Switzerland)*, 14:5333–5353, 03 2014. doi: 10.3390/s140305333.

[103] S. Debattisti, L. Mazzei, and M. Panciroli. Automated extrinsic laser and camera inter-calibration using triangular targets. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 696–701, 2013. doi: 10.1109/IVS.2013.6629548.

[104] Ranjith Unnikrishnan and Martial Hebert. Fast extrinsic calibration of a laser rangefinder to a camera. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-09*, 01 2005.

[105] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster. Automatic camera and range sensor calibration using a single shot. In *2012 IEEE International Conference on Robotics and Automation*, pages 3936–3943, 2012. doi: 10.1109/ICRA.2012.6224570.

[106] Lipu Zhou, Zimo Li, and Michael Kaess. Automatic extrinsic calibration of a camera and a 3d lidar using line and plane correspondences. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5562–5569. IEEE, 2018.

[107] Lionel Heng, Mathias Bürki, Gim Lee, Paul Furgale, Roland Siegwart, and Marc Pollefeys. Infrastructure-based calibration of a multi-camera rig. pages 4912–4919, 05 2014. doi: 10.1109/ICRA.2014.6907579.

[108] Christian Häne, Lionel Heng, Gim Lee, Friedrich Fraundorfer, Paul Furgale, Torsten Sattler, and Marc Pollefeys. 3d visual perception for self-driving cars using a multi-camera system: Calibration, mapping, localization, and obstacle detection. *Image and Vision Computing*, 08 2017. doi: 10.1016/j.imavis.2017.07.003.

[109] D. Scaramuzza, A. Harati, and R. Siegwart. Extrinsic self calibration of a camera and a 3d laser range finder from natural scenes. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4164–4169, 2007. doi: 10.1109/IROS.2007.4399276.

[110] Peyman Moghadam, Michael Bosse, and Robert Zlot. Line-based extrinsic calibration of range and image sensors. volume 2, 05 2013. doi: 10.1109/ICRA.2013.6631095.

[111] Faysal Boughorbel, David Page, Christophe Dumont, and Mongi Abidi. Registration and integration of multisensor data for photorealistic scene reconstruction. *Proc SPIE*, 03 2001. doi: 10.1117/12.384860.

[112] Jesse Levinson and Sebastian Thrun. *Unsupervised Calibration for Multi-beam Lasers*, pages 179–193. 01 2014. doi: 10.1007/978-3-642-28572-1_13.

[113] Gaurav Pandey, James McBride, Silvio Savarese, and Ryan Eustice. Automatic extrinsic calibration of vision and lidar by maximizing mutual information. *Journal of Field Robotics*, 32, 09 2014. doi: 10.1002/rob.21542.

[114] T. Scott, A. A. Morye, P. Piniés, L. M. Paz, I. Posner, and P. Newman. Choosing a time and place for calibration of lidar-camera systems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4349–4356, 2016. doi: 10.1109/ICRA.2016.7487634.

[115] J. Jeong, Y. Cho, and A. Kim. The road is enough! extrinsic calibration of non-overlapping stereo camera and lidar using road information. *IEEE Robotics and Automation Letters*, 4(3):2831–2838, 2019. doi: 10.1109/LRA.2019.2921648.

[116] Eung-su Kim and Soon-Yong Park. Extrinsic calibration between camera and lidar sensors by matching multiple 3d planes. *Sensors*, 20(1), 2020. ISSN 1424-8220. doi: 10.3390/s20010052. URL `https://www.mdpi.com/1424-8220/20/1/52`.

[117] R.B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4, May 2011. doi: 10.1109/ICRA.2011.5980567. URL `http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5980567&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5980567`.

[118] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30 (2):328–341, 2008. doi: 10.1109/TPAMI.2007.1166.

[119] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[120] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[121] Open Source Robotics Foundation. *Vehicle and city Simulation*, 2017-10-26. URL `http://gazebosim.org/blog/car_sim`.

[122] Faraz M. Mirzaei, Dimitrios G. Kottas, and S. Roumeliotis. 3d lidar–camera intrinsic and extrinsic calibration: Identifiability and analytical least-squares-based initialization. *The International Journal of Robotics Research*, 31:452–467, 2012.

[123] Zoltan Pusztai and Levente Hajder. Accurate calibration of lidar-camera systems using ordinary boxes. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 394–402, 2017. doi: 10.1109/ICCVW.2017.53.

[124] Tekla Tóth, Zoltán Pusztai, and Levente Hajder. Automatic lidar-camera calibration of extrinsic parameters using a spherical target. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8580–8586, 2020.

doi: 10.1109/ICRA40945.2020.9197316.

[125] Zixuan Bai, Guang Jiang, and Ailing Xu. Lidar-camera calibration using line correspondences. *Sensors*, 20:6319, 11 2020. doi: 10.3390/s20216319.

[126] Martin Velas, M. Spanel, Zdenek Materna, and A. Herout. Calibration of rgb camera with velodyne lidar. 2014.

[127] Kiyoshi Irie, Masashi Sugiyama, and Masahiro Tomono. Target-less camera-lidar extrinsic calibration using a bagged dependence estimator. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1340–1347, 2016. doi: 10.1109/COASE.2016.7743564.

[128] Alexander Duda and Udo Frese. Accurate detection and localization of checkerboard corners for calibration. 09 2018.