Correntropy ICP SLAM (CICP-SLAM) using Lidar

Ashutosh Singandhupe, Hung Manh La

***Abstract*— This work introduces the idea of correntropy and its usage in the realm of Simultaneous Localization and Mapping (SLAM). The Advanced Robotics and Automation (ARA) Lab team of the University of Nevada, Reno (UNR), has applied correntropy with the Iterative Closest Point (ICP) algorithm to create the Correntropy Iterative Closest Point (CICP) algorithm for handling non-Gaussian noise in lidar data. This was then applied to estimate the pose of a robot. CICP is a concept which we introduce initially (our variant of Iterative Closest Point Algorithm) and later on extend it into an application in SLAM. The estimated pose of an autonomous system is optimized using pose-graph optimization or factor graphs. CICP can be applied to address the presence of non-Gaussian noise in a system and is less computationally expensive than other existing methods of addressing non-Gaussian noise such as statistical outlier removal (SOR). The reduction in computational expense allows for non-Gaussian noise filtering to be more accessible and applicable in fields such as statistical learning, machine learning pattern recognition, and in the design of robust filters. The ARA team evaluated CICP algorithm on the well known KITTI dataset with non-Gaussian noise and compared the results with the traditional ICP.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) has evolved gradually with numerous algorithms that use data from multiple sensors to estimate the state of a system. SLAM is a complex problem to solve and despite it's complexity, researchers across the community have proposed efficient algorithms with increasingly accurate prediction of the state. Modern SLAM algorithms are mostly based on visual data and geometric data of surrounding environment, which can be extracted from sensors like cameras, Lidar, etc. Navigating a robot or any autonomous system in a Global Positioning System (GPS) denied environment is the most important driving force for research in modern SLAM systems [2], [15]. Even though GPS can be used in SLAM frameworks to improve localization, most SLAM applications are targeted to improve the robots localization without the need for GPS.

Early approaches for development of a SLAM system include the use of Kalman Filters (KF). However, Kalman filters are limited to linear systems, where as the real time navigation heavily deals with non-linear environments. In order to deal with these non-linearities, variants of Kalman Filter were introduced like Extended Kalman Filters (EKF) and Unscented Kalman Filters (UKF), which produced promising results in real time environments. Another significant contribution to the SLAM community is the introduction of Particle Filters, which has shown significant improvement with respect to the earlier methods. These approaches represent filtering based methods, which have been tried and tested at various applications in the robotics community. A unique ground breaking approach, which grabbed serious attention is the Graph-based methodology, and is an essential component of our work. In this framework, the robot's pose is modeled as a node in a graph representation and the edges represent the errors in measurements from various sensors. Eventually, this process results in generating a pose graph structure in which the resulting error can be minimized by using mathematical optimization techniques like Gauss-Newton or Levenberg–Marquardt optimization. Popular SLAM techniques like Oriented fast and Rotated Briefs-SLAM (ORB2-SLAM) [10], [11] use the graph-based approach for localization.

Beyond traditional methods as mentioned before, deep learning has found a place in state estimation as well. Given deep learning's high success rate, it has opened a new door for research in SLAM systems. Convolutional Neural Networks (CNN), a field of deep learning, has produced interesting results and is especially known as CNN-SLAM in modern literature [18]. It is based on the understanding that by using a pair of images as the robot moves, the visual data can be fed into an appropriate CNN-based network to acquire the position of the robot. However, the complexity of training a CNN-based network increases drastically in the case of a complex and dynamic environment. This in turn requires a high-end GPU, which cannot be deployed at the required scale in modern robotic systems [13], [17]–[19].

Most SLAM applications deal with the assumption that noise is Gaussian, which is not always the case in real time scenarios. For example in Lidar data, the data can be affected by both Gaussian and non-Gaussian noise. Lidar data is used frequently to find the pose of a robot. By calculating the transformation between the two acquired point cloud data from the Lidar we can estimate the pose of the robot. The well known Iterative Closest Point (ICP) algorithm is used for this purpose. However, in cases where the data is affected by non-Gaussian noise, the estimated transformation can be wrong, which could result in poor trajectory of the robot. In order to remove non-Gaussian or shot noise, algorithms like statistical outlier removal (SOR) can be applied, but are also computationally expensive. Handling non-

Ashutosh Singandhupe and Dr. Hung La are with the Advanced Robotics and Automation (ARA) Laboratory, Department of Computer Science and Engineering, University of Nevada, Reno, NV 89557, USA. Corresponding author: Hung La, email: hla@unr.edu

This material is based upon work supported by 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 source code of the CICP in ROS is available at the ARA lab's github: https://github.com/aralab-unr/CICP-SLAM

Gaussian noise is a critical component for better trajectory estimation and is the driving force for our work and others. For handling non-Gausian noise, we initially introduce the concept of correntropy and further evolve it's application to the well-known ICP algorithm. The idea of correntropy has seen wide spread use in modern machine learning, signal processing and other related fields [16]. Correntropy is a similarity measure between 2 random variables and has been implemented with a Kalman filter [4]. Inspired from this correntropy concept, we introduce it into the ICP algorithm, called correntropy ICP or CICP. Our main contribution to this work can be summarized as:

- Introducing the idea of correntropy, which explains the similarity between 2 random variables and it's application to handling non-Gaussian noises.
- Incorporating correntropy into the ICP algorithm (CICP) where we explain how correntropy can be used in the ICP algorithm.
- Formulating CICP SLAM - where we provide a formulation of CICP algorithm as well as perform pose graph optimization for loop closure detection.

In this work, we propose a solution to handle non-Gaussian noise in Lidar data. The Advanced Robotics and Automation (ARA) Lab's team does this by introducing non-Gaussian noise or shot noise into Lidar data to evaluate how a basic SLAM framework (ICP with pose graph optimization) responds to the non-Gaussian noise. Our team concludes that the basic SLAM framework cannot handle non-Gaussian noise effectively and results in faulty trajectory estimation. In order to resolve this, we introduce the idea of correntropy in the well known ICP algorithm (CICP algorithm) and show through our evaluation on the well-known KITTI datsets that the application of correntropy is effective in handling non-Gaussian noise. CICP results in a better estimate as compared to the traditional ICP algorithm by utilizing the similarity in the Lidar data acquired over time. The ARA Lab uses the created CICP algorithm to estimate the relative pose of the robot, and also applies graph optimization to minimize the error in pose and detect loop closures.

The remaining paper is organized as follows: Section II introduces the idea of correntropy and its underlying concepts. Section III describes our implementation of the proposed methodology. The results and evaluation of our proposed methodology is discussed in Section IV. Conclusions are given in Section V.

II. Correntropy Criterion

1) Correntropy: We begin by describing the mathematical representation of a Lidar data and develop our algorithm based on it. We denote the Lidar point clouds acquired at time i as P_i where $i = 0, 1, 2, ..., t$, and t is the time period. We use the correntropy concept between the point clouds acquired at consecutive time intervals i and $i + 1$, since the geometrical features of the scene acquired at such close time intervals does not change significantly. Every point in the point cloud P_i contains N points in which each point can be referenced as p_j, where $j = 1, ..., N$ and $p_j = \{x_j, y_j, z_j\}$, where x_j, y_j, z_j denotes the 3D coordinates of the point p_j. Similarly, the point cloud P_{i+1}, contains N points in which each point can be represented as q_k where $k = 1, ..., N$ and $q_k = \{x_k, y_k, z_k\}$, where x_k, y_k, z_k denotes the 3D coordinates of the point q_k. Figure 1 shows a brief description of the point clouds and it's mathematical notations. The idea of correntropy has gained importance

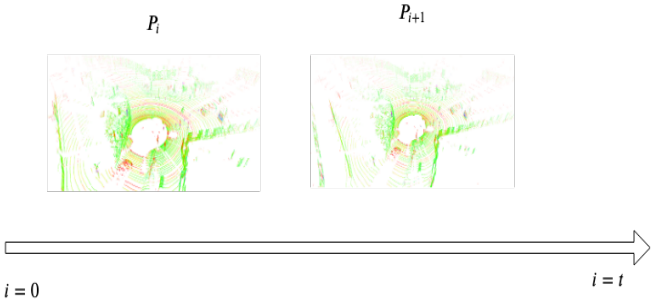


Fig. 1: Point Clouds acquired over time from Lidar (source: KITTI dataset sequence 05). P_i contains N points p_j, each of which is a 3D representation x_j, y_j, z_j. Similarly, P_{i+1} contains N points q_k, each of which is a 3D representation x_k, y_k, z_k.

in recent years in the fields of machine learning, pattern recognition and designing filters in order to remove non-Gaussian noise. It has proved beneficial to remove large outliers [16]. Correntropy is essentially a similarity measure of two scalar random variables. With reference to Lidar, we are measuring the similarity between 2 point clouds P_i and P_{i+1} acquired at time intervals i and $i + 1$. Then the correntropy criterion between the 2 point clouds P_i and P_{i+1} can be mathematically represented as :

$$V_\sigma(P_i, P_{i+1}) = E[\kappa_\sigma(P_i - P_{i+1})]. \tag{1}$$

This also refers to in common literature as a cross-correntropy of two scalar random variables [3], [4], [8].

In Equ. (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 as bandwidth σ). In our approach, we use the Gaussian kernel function where we define the correntropy function as:

$$V_\sigma(P_i, P_{i+1}) = \frac{1}{N}\sum_{i=1}^{N} G_\sigma(p_j - q_k), \tag{2}$$

where p_j and q_k are N points from point clouds P_i and P_{i+1} respectively. Equ. (2) plays a key role in our work where,

$$G_\sigma(p_j - q_k) = exp(-\frac{\|p_j - q_k\|^2}{2 * \sigma^2}), \tag{3}$$

and σ is the bandwidth or the kernel size of the Gaussian kernel. From Equ. (3) it becomes evident that if $P_i =$

P_{i+1} Gaussian correntropy is maximum, and the Gaussian correntropy function is positive and bounded [3], [4], [8]. Fig. 2 shows the basic understanding of correntropy criterion. In addition, the maximum of correntropy of error in Equ. (3) is called the maximum correntropy criterion.

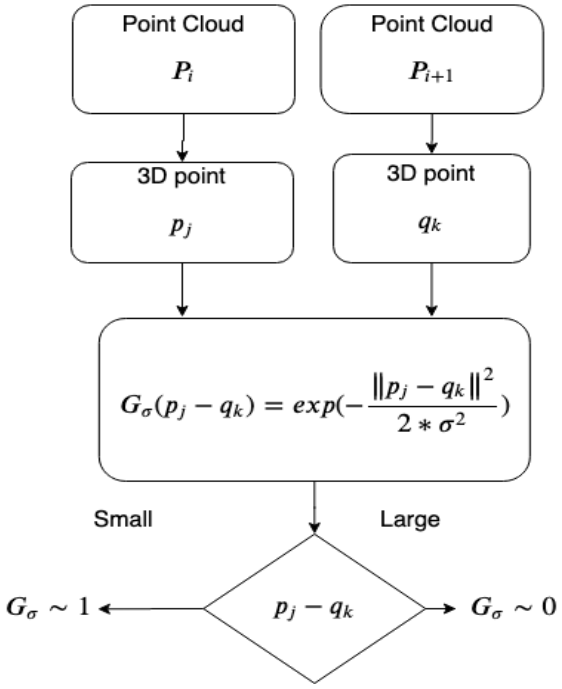


Fig. 2: Correntropy Criterion.

III. Proposed Methodology

1) Correntropy 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. By exploring the idea of correntropy in relation to the ICP algorithm, we see how the correntropy criterion can be used to handle non-Gaussian noise in dataset. Traditional ICP [1] describes the aligning of 2 point clouds so that the Mean Square Error (MSE) between the 2 point sets is minimized. The MSE is the key criterion in ICP and its variants. The idea of ICP revolves around the problem of aligning 2 points sets P_i and P_{i+1}. We denote the 2 point sets as $P_i = \left\{[p_j]_{j=1}^N\right\}$ (here $\{\}$ denotes a point set) and $P_{i+1} = \left\{[p_k]_{k=1}^N\right\}$, respectively. We define P_{i+1} as the source point set, which is to be aligned with the target point set (or a model point set) P_i. This results in finding the appropriate rotation and translation between the two point sets, which essentially means how much units the source data set needs to be translated (tr) and rotated (R) such that source point set P_{i+1} is aligned well with P_i given that it satisfies a particular criterion (Root Mean Square Error (RMSE) threshold). This process is commonly known as the registration. So, the registration between the source P_{i+1} and the target P_i is defined as finding the translation tr and the rotation R component so that P_{i+1} is in best alignment with P_i. Now, the problem can be written as: find the best R and tr such that the MSE ε^2 between 2 point clouds is minimized. It can be mathematically written as,

$$\varepsilon^2 = \sum_{m=1}^{N} \|p_m - (sRq_m + tr)\|^2 . \tag{4}$$

where, N is again the number of data points, which is generally equal in both point sets, and s is the scaling component. $p_m \in P_{i+1}$ is the collection of all the points in the source point set, and $q_m \in P_i$ is the collection of all the points in the target point set. In relation to correntropy, the problem can be equivalently formulated as:

$$\max_{s,R,tr} \sum_{m=1}^{N} exp(-\|p_m - (sRq_m + tr)\|^2 /(2\sigma^2)), \tag{5}$$

where σ is the bandwidth of the given Gaussian kernel, which can be user defined. We iteratively perform this procedure in order to minimize the error between the 2 point clouds. For every iteration n in CICP, firstly, we use the nearest neighbour search to compute $c_n(u)$ and $d_n(v)$, which stores the closest point in one set to every other point in another set. It is computed as follows:

$$c_n(u) = arg \min_{u,v \in \{1,2,\ldots,N\}} (R_{n-1}p_u + tr_{n-1} - q_v) \tag{6}$$

$$d_n(v) = arg \min_{u,v \in \{1,2,\ldots,N\}} (R_{n-1}p_u + tr_{n-1} - q_v). \tag{7}$$

Equ.(6) and Equ.(7) can be written as another set of point correspondences as,

$$a_r = \begin{Bmatrix} p_u & 1 \le u \le N \\ p_{d_n(v)} & 1 \le v \le N \end{Bmatrix} \tag{8}$$

$$b_r = \begin{Bmatrix} q_{c_n(u)} & 1 \le u \le N \\ q_v & 1 \le v \le N \end{Bmatrix} \tag{9}$$

Here a_r and b_r represent the point correspondences in both the point data sets, respectively. Incorporating correntropy, the rotation and the translation components can be computed as,

$$F(R,tr) = (1/N)\sum_{r=1}^{N} exp(-\|b_r - (sRa_r + tr)\|^2)/(2\sigma^2)), \tag{10}$$

Taking derivative of the Equ.(10) and equating it to 0 we can calculate the translation vector tr as,

$$tr = \frac{\sum_{r=1}^{N} G_\sigma(e_r)(b_i - Ra_r)}{\sum_{i=1}^{N} G_\sigma(e_r)}, \tag{11}$$

where, $e_r = Ra_i + tr - b_i$. The centroid can be computed as,

$$p_{cen_r} \sim a_r - \frac{\sum_{r=1}^{N} G_\sigma(e_r)a_r}{\sum_{i=r}^{N} G_\sigma(e_r)}, \tag{12}$$

$$q_{cen_r} \sim b_r - \frac{\sum_{r=1}^{N} G_\sigma(e_r)b_r}{\sum_{r=1}^{N} G_\sigma(e_r)}. \tag{13}$$

Now the rotation matrix can be calculated by computing the

singular value decomposition (SVD) of a matrix H where,

$$H = \sum_{r=1}^{N} p_{cen_r} G_\sigma(e_i) q_{cen_r}^T. \quad (14)$$

where T is the transpose in Equ.(14) and it is very similar to the well known algorithm given by [1]. The optimal rotation matrix is given as $R^* = VDU^T$, where V is a $m \times m$ real or complex matrix, D is a $m \times n$ diagonal matrix, and U is an $n \times n$ real or complex matrix [1]. The CICP algorithm can be described in steps as mentioned in Algorithm 1.

Algorithm 1: CICP Algorithm

1: Source point cloud $(P_{i+1}) \rightarrow \{p_j\}_{j=1}^{N}$
2: Target Point Cloud $(P_i) \rightarrow \{q_k\}_{k=1}^{N}$
3: Initialize $R_0 = I$ and $t_0 = 0$ for $k = 0$.
4: **for** $n = 1, 2, ...K$ **do**
5: Compute $c_n(u)$ and $d_n(v)$ from Equ.(6) and Equ.(7).
6: Compute correspondence points a_r and b_r from Equ.(8) and Equ.(9).
7: Compute kernel $G_\sigma(e_n)$ by R_{n-1} and tr_{n-1}.
8: Compute the Rotation matrix R^* from Equ.(14).
9: Compute the Mean Square Error(MSE) e_n as given by Equ.(4).
10: break when $\|e_n - e_{n-1}\| < threshold$
11: **end for**
12: Return pairwise transformations $T_0, T_1, ..., T_n$.

In Algorithm 1, $threshold$ is a mean square error as defined by the user. We have implemented this algorithm in the well known PCL library and can be used by calling the function pcl::registration::TransformationEstimationCorrentropySVD. It is available in our github repository as our own extension of PCL [14].

Fig. 3 gives the basic system architecture of our proposed approach.

We get the point clouds acquired from the Lidar and perform the initial preprocessing step of removing the NaN values. Let the acquired point clouds over a period of time be denoted as $P_0, P_1, ..., P_t$. We do not perform any form of filtering like statistical outlier removal since our idea is to work on the data with noise (or non-Gaussian noise). We remove the number of point vlouds using voxel grid filtering for reducing the computational time however the noise in the data still persists after reduction. Let point clouds received after voxel grid filtering be $\hat{P}_0, \hat{P}_1, ..., \hat{P}_t$. We perform pairwise alignment on the incoming point clouds in order to calculate the pose using our pcl::registration::TransformationEstimationCorrentropySVD implemented in the pcl library [14]. We use the first acquired point cloud, P_0, which we initialize as target and align the next acquired point cloud P_1, which we initialize it as source. The aligned point clouds gives the rotation and translation component between these 2 point clouds (denoted as T_0), and the steps is repeated for the next incoming point clouds (which results in transformations $T_1, T_2, ...T_t$). This results in the over all estimated pose of the robot as it is navigating in an environment.

It is quite clear that the CICP outputs the transformation matrix between 2 point clouds. It is composed of keyframes K_i (point clouds) acquired at the robot position pos_i. The keyframes are stored and can be used later in order to detect a previously visited environment. The result of the CICP algorithm is the relative transformation computations between the acquired point clouds, i.e., $T_0, T_1, ..., T_n$. However it also results in accumulation of errors. A pose graph representation is essential for reducing the errors as well as detecting a loop. A pose graph representation is a node-edge based graph representation where every node represents the robot poses, and the edges are the relative error between the pose. It is also called a factor graph, which is a bipartite graph $\xi = (F, X, E)$. Here $f_i \in F$ is the factor nodes (previous robot pose) and current robot pose $pos_j \in X$. Edge $edge_{ij} \in E$ represents the relative transformation $(T_0, T_1, ..., T_n)$ in this case. The pose graph optimization can be described in Algorithm 2.

Algorithm 2: Pose Graph Optimization Algorithm

1: Data $\rightarrow \{data_i\}$,
2: where $data_i$=P_i point clouds
3: Graph Initialization g.
4: **for** all P_i **do**
5: g = BuildFrontEnd (g,P_i).
6: g = LoopClosure (g).
7: g = BackEndOptmization (g).
8: **end for**
9: Return g

The FrontEnd optimization in Algorithm 2 can be described in Algorithm 3.

The relative pose calculated from our CICP algorithm is used as input to $AddEdge2Graph$ function in Algorithm 3. The BackEndOptimization can find the state vector that minimizes the error from the constraints accumulated throughout the data using Gauss-Newton Method [7]. We use the well known $g2o$ library for our pose graph optimization [9].

IV. Results

The ARA Lab team evaluated CICP on the famous KITTI dataset and compared the root mean square error (RMSE) of both the ICP and CICP algorithms using the available ground truth data. Non-Gaussian noise was added to the Lidar data at random locations and then put through ICP and CICP algorithms to approximate the vehicle's trajectory. As mentioned earlier, the purpose of adding non-Gaussian noise is to evaluate how well CICP performs in comparison to the traditional ICP algorithm. In our methodology we solely use Voxel Grid Filtering [12] to reduce the number of point clouds for computation, which still does not fully remove

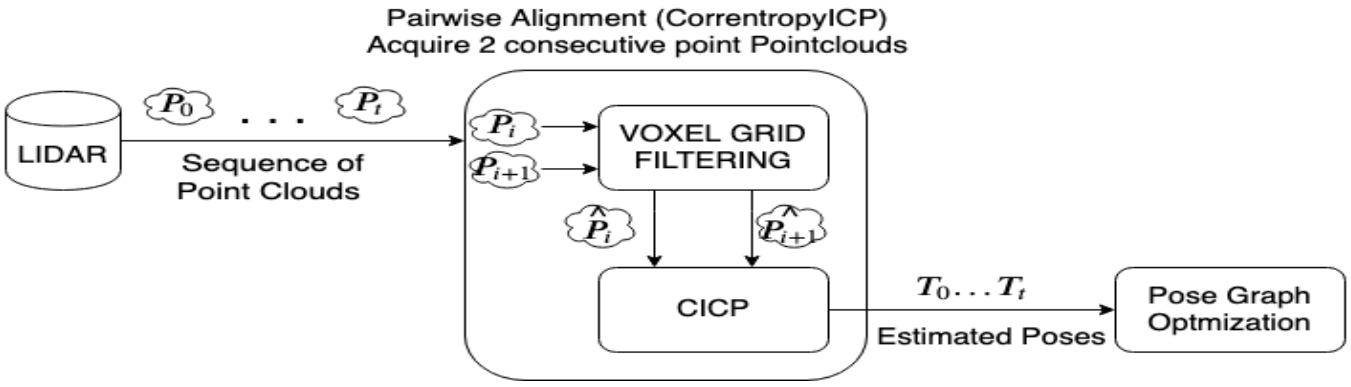


Fig. 3: System Architecture.

Algorithm 3: Front-End Algorithm

Given: Data $\rightarrow\{P_i\}$,g
if $GetNewData(P_i) = true$ **then**
  $NewNode = CreateNode(P_i)$
  $g = AddNode2Graph(g, NewNode)$
  $edge_{CICP} = CreateEdge(P_i, NewNode)$
  $g = AddEdge2Graph(g, edge_{CICP})$
  $Reg =$
  $RegistrationCICP(NewNode, PreviousNode(g))$
  $edge =$
  $CreateEdge(Reg, NewNode, PreviousNode(g))$
  $g = AddEdge2Graph(g, Edge)$
  $LoopClosureNodes =$
  $FindLoopClosureNodes(g, NewNode)$
  for all ($node \in$LoopClosureNodes) **do**
    $matched, M =$
    $FeatureMatching(NewNode, Node)$
    if $matched = true$ **then**
      $LoopClosureEdge =$
      $CreateEdge(M, NewNode, Node)$
      $g = AddEdge2Graph(g, LoopClosureEdge)$
    end if
end if
Return g

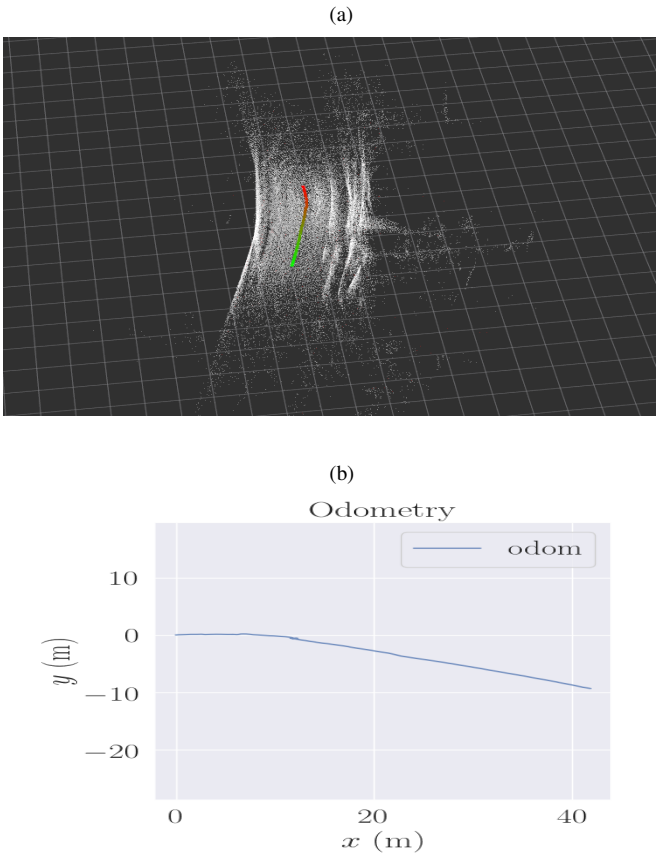


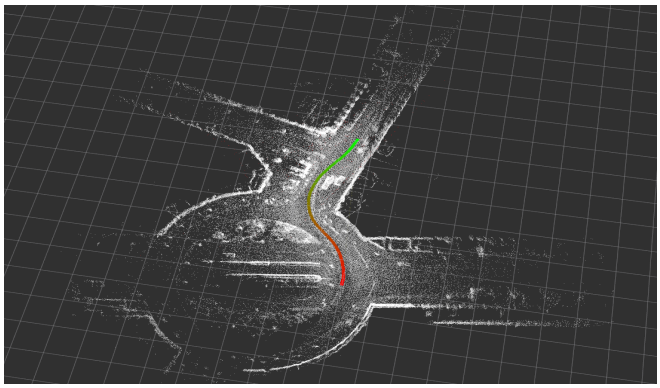
Fig. 4: (a) Sample KITTI map and (b) trajectory of KITTI raw dataset using our CICP algorithm. (Sequence: kITTI drive 0001 (2011/08/06).

the non-Gaussian noise completely. Fig. 4 and 5 shows a sample trajectory on the KITTI dataset using our CICP algorithm.

One of the interesting KITTI datasets is sequence number 06, and its estimated trajectory results are shown in Fig. 6. Here, a presence of a truck affects the trajectory significantly if ICP was used. The result of the ICP based trajectory estimation is shown in Fig. 6 (a). The selected circular region shows how the trajectory gets affected due to the presence of a truck. In comparison to the results from ICP, our CICP algorithm performs significantly better in this sequence as it can be seen from Figure 6 (b).

We have evaluated on other KITTI sequences and compared our approach to the traditional ICP algorithm [6] [5]. In all the sequences we have added non-Gaussian noise in the Lidar data. Fig. 8 and Fig. 9 shows sample trajectories of both the ICP and CICP algorithms for KITTI Sequences 07 and 10, respectively. In addition we have also evaluated the results on different KITTI sequences where the ARA team added shot noise at random locations in the Lidar data. Table I shows the RMSE comparison of ICP as well as our CICP

(a)



(b)

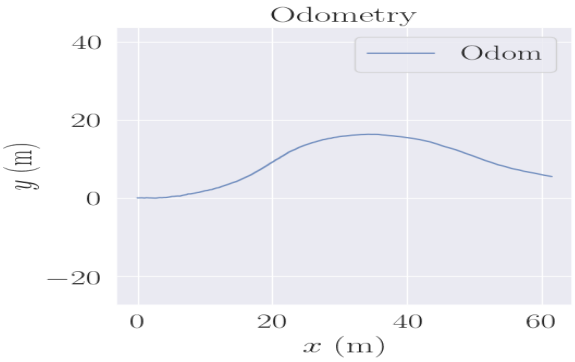
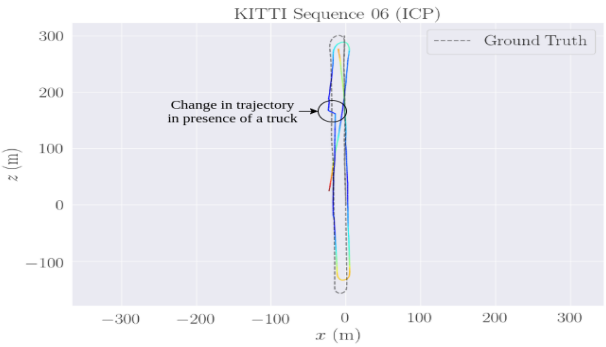


Fig. 5: (a)Sample KITTI map and (b) trajectory of KITTI raw dataset using our CICP algorithm. (Sequence: KITTI drive 0005 (2011/08/06).

(a)



(b)

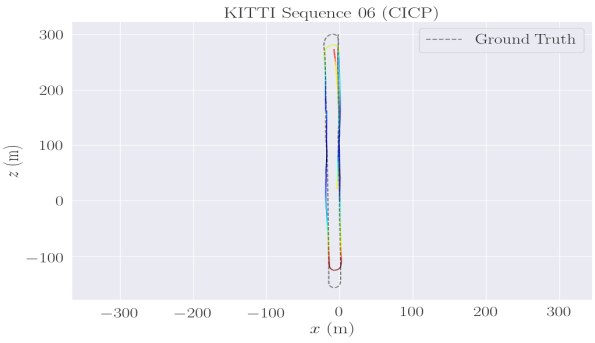


Fig. 6: KITTI Sequence 06 (with injected noise). (a) ICP with pose graph optmization (RMSE: 15.7829) (b) CICP with pose graph optimization (RMSE: 10.7086).

algorithm for various KITTI sequences.

TABLE I: RMSE comparison of CICP and ICP in the presence of additional noise.

KITTI Sequences	ICP-SLAM	CICP-SLAM
SEQ00	50.3672	45.3321
SEQ01	21.0067	14.5678
SEQ02	62.8892	51.0022
SEQ04	9.3461	3.4325
SEQ05	37.9821	28.2211
SEQ06	15.7829	10.7086
SEQ07	16.213905	12.091864
SEQ09	62.167	55.876
SEQ10	35.102207	29.453647

From Table I it is evident that CICP algorithm performs significantly better than the traditional ICP algorithm in presence of non-Gaussian noise.

V. Conclusions

In this work, we have proposed a novel Correntropy Iterative Closest Point (CICP) SLAM algorithm along with pose

Fig. 7: Presence of a truck, which is close and affects the odometry.

graph optimization for loop detection and closure. We have shown how CICP performs better than the traditional ICP in response to non-Gaussian noise. To evaluate our approach we have introduced shot noises to the raw Lidar data and evaluated the trajectory of both ICP and CICP algorithms and shown the advantages of applying correntropy in noise filtering. Our future work will be focusing on using both the Lidar and the Stereo data for the ICIP algorithm. We plan to use the idea of Correntropy to find the similarity between the Lidar point clouds and the Stereo Point clouds and estimate the pose of the robot along with building a high-definition

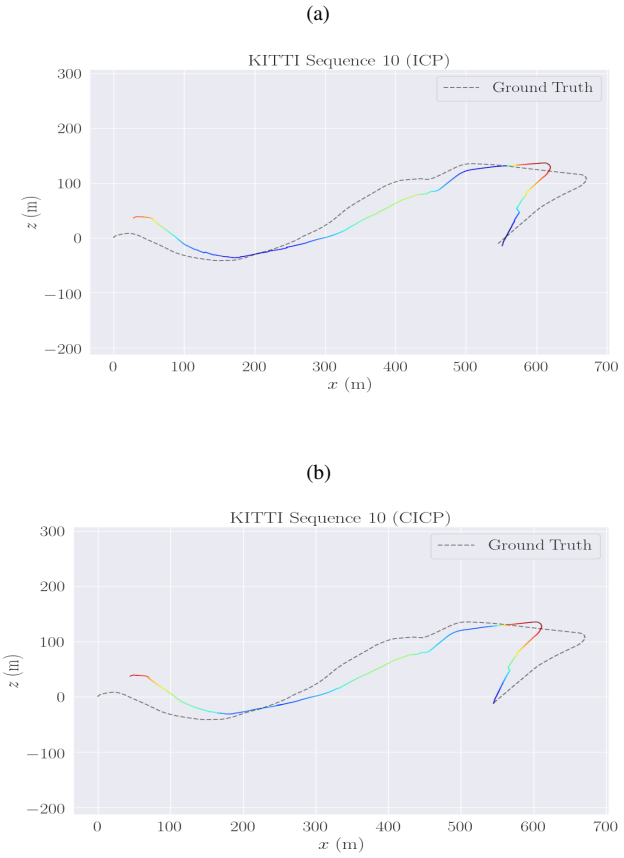


Fig. 8: KITTI Sequence 10 (with injected noise). (a) ICP with pose graph optmization (RMSE: 35.102207) (b) CICP with pose graph optimization (RMSE: 29.453647).

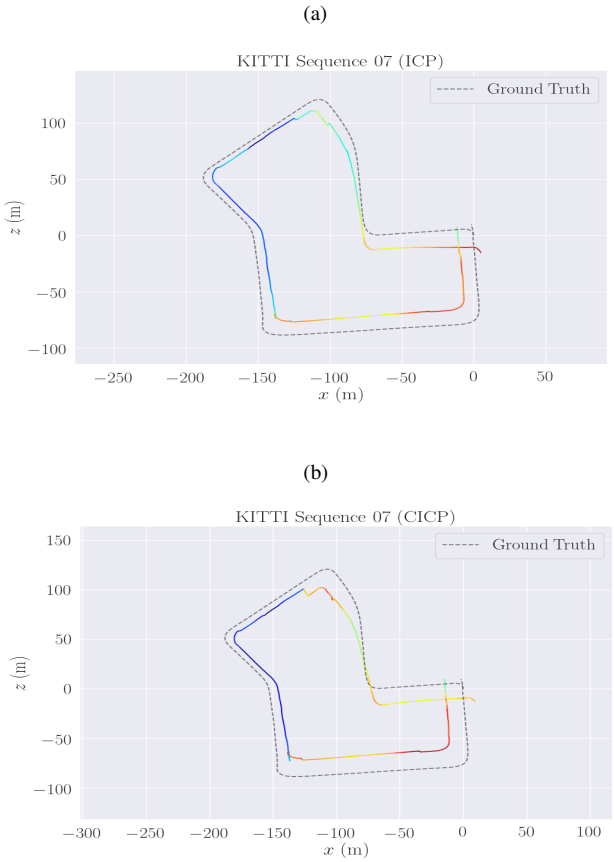


Fig. 9: KITTI Sequence 07 (with injected noise). (a) ICP with pose graph optmization (RMSE: 16.213905) (b) CICP with pose graph optimization (RMSE: 12.091864).

map of the environment.

REFERENCES

[1] 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.

[2] 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.

[3] B. Chen, X. Liu, H. Zhao, and J. C. Principe. Maximum correntropy kalman filter. *Automatica*, 76:70 – 77, 2017.

[4] S. Fakoorian, A. Mohammadi, V. Azimi, and D. 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.

[5] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

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

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

[8] R. Izanloo, S. A. Fakoorian, H. S. Yazdi, and D. Simon. Kalman filtering based on the maximum correntropy criterion in the presence of non-gaussian noise. In *CISS*, pages 500–505. IEEE, 2016.

[9] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. pages 3607 – 3613, 06 2011.

[10] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Trans. on Robotics*, 31(5):1147–1163, 2015.

[11] R. Mur-Artal and J. 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.

[12] PCL. Voxel grid filtering. `http://pointclouds.org/documentation/tutorials/voxel_grid.php`.

[13] A. Sehgal, A. Singandhupe, H. M. La, A. Tavakkoli, and S. J. Louis. Lidar-monocular visual odometry with genetic algorithm for parameter optimization. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, D. Ushizima, S. Chai, S. Sueda, X. Lin, A. Lu, D. Thalmann, C. Wang, and P. Xu, editors, *Advances in Visual Computing*, pages 358–370, Cham, 2019. Springer International Publishing.

[14] A. Singandhupe. Cicp-slam. `https://github.com/aralab-unr/CICP-SLAM`, 2020.

[15] 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.

[16] 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.

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

[18] K. Tateno, F. Tombari, I. Laina, and N. Navab. CNN-SLAM: real-time dense monocular SLAM with learned depth prediction. *CoRR*, abs/1704.03489, 2017.

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