Point-LIO

Sahil Chaudhary, Richa Mohta, Taylor Pool, Shreyansh Sharma

***Abstract*—This paper presents Point-LIO, a novel filtering based approach to LIDAR-Inertial Odometry. The algorithm treats inputs from Inertial Measurement Units (IMUs) and LIDAR sensors as proper measurement updates within an Extended Kalman Filter (EKF) framework. Our objective is to evaluate the performance of the Point-LIO system on data collected on ground vehicles navigating through long, narrow corridors. Given the necessity for real-time processing, we are implementing the algorithm in C++. Furthermore, we enhance the algorithm's accuracy further by incorporating an extension based on the Unscented Kalman Filter (UKF) - demonstrating the capability to estimate robot state during aggresive motion.**

***Index Terms*—State Estimation, SLAM, LIDAR, IMU, Scan Registration, UKF.**

I. Introduction

Robots are ubiquitous in the world today, ranging from autopilot software in aircraft to home vacuum cleaners. A key component of all robotic systems is state estimation. Simply speaking, for a robotic agent to be of any use, it must know its location and the shape of its environment. Similar to humans, robots rely on sensor inputs to receive information about the world around them. This input helps them understand their surroundings and make decisions, much like how humans use their senses to navigate and interact with their environment. Over the years, a wide array of sensors have been developed that cater to the diverse needs of robotics. These sensors are different in terms of their functionalities as well as capabilities, ranging from simple proximity sensors that detect the presence of nearby objects to sophisticated cameras and LIDAR systems that provide detailed spatial information. Two of the most prevalent ones are Light Detection And Ranging (LIDAR) and Inertial Measurement Units (IMU).

LIDAR is a highly accurate sensor that can be used for precise ranging. It emits laser pulses, and measures the duration of time it takes for them to reflect back off neighbouring objects. It uses this total travel time to calculate the distances to objects, allowing for the creation of precise 3D maps of the surrounding area. LIDAR's ability to scan quickly allows it to collect discrete points in space, creating a dense point cloud that accurately represents the surrounding topography and geometry, facilitating precise navigation and interaction for robots. Robots can explore and interact with their environment with a high degree of confidence and precision, leveraging this comprehensive spatial data.

In robotic applications, IMU serves as an indispensable component providing high-rate measurements of specific force (acceleration) and angular rate (rotation) using a combination of accelerometers and gyroscopes. It offers real-time feedback by continuously detecting changes in rotation and acceleration, making it crucial for tracking a robot's motion thereby enabling a refined navigation and control system. By integrating these measurements over time, the IMU calculates changes in velocity and orientation relative to the object's starting point. However, because IMUs are prone to errors that accumulate over time (known as drift), this dead reckoning becomes less reliable the longer the object moves and hence results in inaccurate orientation and location prediction.

The goal of this paper is to combine LIDAR and IMU sensors together into a single filtering-based estimator. Integrating these complementary sensing modalities harnesses the strengths of each to achieve high-rate and robust odometry estimation, offering practical advancements in robotics for navigation in varied and demanding environments. Additionally, this integration enhances the adaptability of robotic systems to dynamic surroundings, enabling them to effectively navigate through complex terrains and overcome obstacles in real-time.

We show that our method does not perform well, due to the high number of LIDAR points and sensitivity to timing. Our approach has found that Point-LIO is not suitable for operation on real robots, and we discourage others from attempting to utilize this approach.

II. Background

As stated in the introduction, we seek to show that IMU and LIDAR can be fused together in an intelligent manner. LIDAR is a sensor that samples at discrete points in time and returns the range and azimuth associated with the points in question. However, a common practice is to accumulate these points into a scan. Even though this aggregation aids in data processing, it introduces motion distortion, specifically when the LIDAR is in motion at high speeds. This distortion can compromise the accuracy of spatial measurements and pose challenges for navigation and mapping tasks. In addition to increasing lag, the low frame rate restricts the available bandwidth.

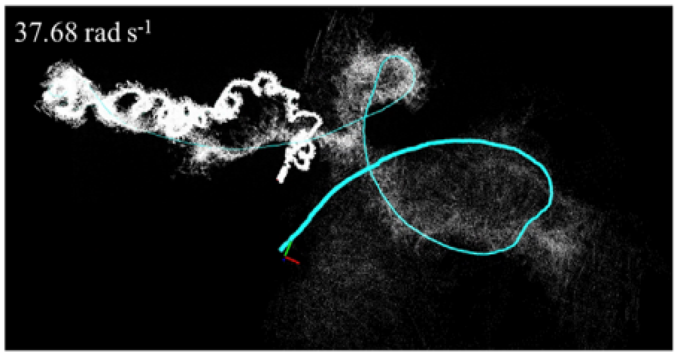


Fig. 1: Taken from Point LIO [5]

Traditional methods that seek to compensate for motion distortion include constant velocity based methods [10]. Other methods involve more sophistication, including LOAM [16], CT-ICP [4].

We take inspiration from Point-LIO, a framework that entirely circumvents the motion distortion problem because it processes each LIDAR point sequentially on its own. This sequential processing of points also enables an odometry rate in the range of kHz, whereas, in traditional scan-based methods, it is typically 10 Hz. This enables the LIDAR odometry rate to be even faster than the IMU odometry rate, which is around 100 Hz.

Building on this foundation, our methodology involves the use of point-wise processing of LIDAR points by checking the plane correspondence of the given LIDAR point with its five nearest neighbors, as part of the measurement model. If the point does not lie in the plane, we add it to the map by assuming that it is a new point. This essentially encompasses the addition of new information to the map. If it lies in the plane, measurement update proceeds without any changes to the map. In order to enable fast insertion of points to the map as well as fast querying of the five nearest neighbors for the plane correspondence, we represent the points of the point cloud map using a variation of the KD-Tree known as the incremental KD-Tree. This tailored approach contributes to the robustness and overall effectiveness of our methodology.

A KD-tree, at its base, is a binary tree where points are inserted at each time step. When the robot reaches a particular point or node, it searches through the KD-tree for any prior information about that point. If the point is not present, then it is added to the KD-tree at an appropriate location. To efficiently process each LIDAR point in turn, the need for rapid matching between a given point and the map of LIDAR points is crucial. To address this, the authors in [2] developed a KD-tree structure that can be incrementally updated over time in a very efficient manner. However, if excessive points accumulate, the tree can become imbalanced, making searches through the tree computationally more expensive. Therefore, balancing—i.e., rearranging the points.

While it is true that traditional KD-trees are incremental in their updates, this novel data structure also incorporates adaptive downsampling and a sophisticated rebalancing algorithm that ensures optimal search efficiency and reweighting. The incremental KD-tree specifically addresses the challenge of real-time updates with minimal computational overhead. It does this by allowing points to be added to the tree as they are encountered without the need for a complete rebuild of the structure. When the tree becomes too dense or imbalanced, the adaptive downsampling kicks in to thin out regions with a high concentration of points, thereby preventing the tree from becoming overly complex and maintaining the efficiency of search operations.

Moreover, the sophisticated rebalancing algorithm periodically evaluates the structure of the KD-tree to identify areas where the tree may have become imbalanced. It then performs targeted rearrangements of the nodes to ensure that the depth of the tree remains as uniform as possible, thereby optimizing the search path for any given point. This rebalancing is crucial for maintaining the KD-tree's efficiency, especially in dynamic environments where the distribution of points can change rapidly. This same tree was used by Point-LIO [5] and is one of the key drivers of real time performance.

Another significant issue that arises is that IMUs can become saturated in aggressive and fast motions, as the robot's angular velocity and linear acceleration might exceed the IMU's measuring range. This can yield the IMU to be useless during those time-steps. In order to overcome this, we use a stochastic process-augmented kinematic model [5], in which the IMU measurements, which are angular velocity and linear acceleration, are modeled as outputs of the system. This means that they are part of the state vector and are used in the update stage. Doing this enables us to estimate the angular velocity and linear acceleration as part of the state, and then correct the estimate using a filtering approach such as the Extended Kalman Filter (EKF) or the Unscented Kalman Filter (UKF). If during a certain maneuver or time-step the actual IMU reading becomes saturated, we just skip the measurements from those saturated channels, and use the current estimate of the angular velocity and linear acceleration from the propagate model during that time. This ensures that the system is more robust during aggressive and fast motion, as the system can continue to operate effectively and maintain functionality even when the IMU measurements are saturated. Thus, overall reliability and performance is enhanced.

The Extended Kalman Filter is a variant of the Kalman Filter for non-linear systems. The fundamental principle of the EKF is to linearize the measurement and system models with respect to the current state estimation. This linearization process makes it possible to use the Kalman Filter without any other change. It is assumed that the noise in the system and the measurements is parameterized by a Gaussian distribution. However, the linearity of the system and the quality of the first estimation of the state have a major impact on the filter's accuracy. Additionally, errors may occasionally result and accumulate from the linearization procedure, particularly if the system is highly non-linear.

An alternative to the EKF that works especially well for non-linear systems that are parametrized with non-Gaussian noise distributions is the Unscented Kalman Filter (UKF). In the UKF, a collection of representative points (referred to as sigma points) are selected deterministically rather than linearizing the system and measurement models. Therefore sigma points are particularly advantageous in non-linear systems as they offer a more accurate representation of the mean and covariance of the state distribution. In order to estimate the state's mean and covariance in the following time step, the UKF propagates these sigma points through the non-linear system of propgate and measurement models. This algorithm bypasses linearizing the process and measurements models, thereby sidestepping the inaccuracies associated with linearization.

Point-LIO combines point-based LIDAR point processing with this novel state formulation and state propagation model to make a robust and reliable state estimator. This approach is particularly effective during aggressive and fast motion scenarios, where traditional scan-based LIDAR processing and IMU integration methods may falter.

III. Related Work

The very first modern approach to LiDAR-Inertial Odometry was by LOAM [16]. This approach leveraged the extraction of planar and edge features to conduct more sophisticated matching, thereby enhancing the accuracy and reliability of odometry estimation. Furthermore, LiDAR Odometry operates on the principle of scan registration, a concept deeply rooted in the iterative closest point method [1], which plays a pivotal role in aligning successive LiDAR scans and refining the estimation process.

While the distinction between scan-to-scan and scan-to-map procedures can indeed alleviate the computational workload for odometry, the registration of scans from one to the next often accelerates the accumulation of drift. Furthermore, this registration process demands significant overlap between consecutive scans, a condition that may not be satisfied by compact solid-state LiDARs with limited Field of View (FoV). [7]

[13] uses a scan-to-map registration by combining IMU measurements in an effective iterated Kalman filter approach. A fundamental issue that the scan-to map framework faces is maintaining the map structure such that it can not only add points from new scans but also simultaneously allow for efficient queries. To address this problem, an incremental k-d tree is proposed.

The Kalman Filter [6] is one of the best works presented. It establishes the optimal state estimation for linear systems with Gaussian noise. It continues to be an important tool for handling challenging estimation and localization problems and improving system performance in a variety of disciplines, such as navigation, control systems, and signal processing.

The fusion of IMU and LIDAR can be classified into two broad categories. The first is a smoothing method, inspired in part by work done on factor graphs [3]. Smoothing methods maintain a window of states accumulated over time. They seek to optimize for the most likely states using multiple sensor measurements. Approaches that utilize the smoothing method include LIO-SAM [9] and Super Odometry [17]. Another method that also utilizes a form of smoothing, albeit in a continuous-time formulation is Continuous Time - Iterative Closest Point (CT-ICP) [4].

Point-LIO [5], belonging to the second classification of sensor fusion, departs from this branch of smoothing-based methods by emphasizing the simplicity of utilizing an Extended Kalman Filter [12]. Additionally, Point-LIO sidesteps a fundamental problem with radially spinning LIDAR scanners, which is motion distortion caused by each point being sampled at different discrete time intervals. Point-LIO overcomes this issue by recognizing that each LIDAR point has its own individual timestamp. By utilizing the timestamp of each point independently and processing each point in turn, Point-LIO avoids the need for complex de-warping operations often introducing a higher risk of failure. This ensures precise alignment of measurements. The smoothness of the splines also inherently prevents aggressive motions. Another common approach is to utilize the IMU measurements for motion compensation [14]. In order to undistort the LIDAR points, these algorithms integrate the LiDAR pose utilizing the IMU data within a frame. However, this method is limited by the IMU frequency and suffers from the IMU noise and biases.

Motion distortion has been tackled by assuming constant velocity during the frame [15]. However, this method only applies when the scan duration is short and the motion is not aggressive. Another popular method for motion compensation is based on continuous-time trajectory optimization, such as those based on B-Splines [8]. Continuous-time trajectories compensate for the distortion of each point by enabling the evaluation of pose at any instant in time. However, this method is computationally expensive and takes a long time, and is hence usually done offline.

Unfortunately, the Extended Kalman Filter represents a sub-optimal approach to propagating Gaussian distributions through non-linear functions. This is because of its reliance on linear approximations, which may not adequately handle the dynamics of the system. In order to combat this issue and shore up the abilities of Point-LIO, we contribute an Unscented Kalman Filter [11]. Unlike the Extended Kalman Filter, the Unscented Kalman Filter which utilizes sigma point samples propagated through a non-linear function to obtain a more accurate approximation of the posterior distribution. This approach offers improves performance in highly complex and dynamic environments.

IV. Point-LIO

Point-LIO consists of an Extended Kalman Filter that relies on a state propagation step.

The equations for the IMU are given as follows:

$$f_a = {}^b a - {}^b R_w {}^w g + b_a + \eta_a \tag{1}$$
$$\hat{\omega} = {}^b \omega + b_\omega + \eta_\omega \tag{2}$$

Where f_a is the specific forces, ${}^b a$ is the acceleration in the frame of the IMU (also the body frame). Also, ${}^b R_w$ is the rotation from the world frame to the body frame, ${}^w g$ is the gravity vector in the world frame, b_a is the bias of the IMU, and η_a is additional white noise.

Note that in our case,

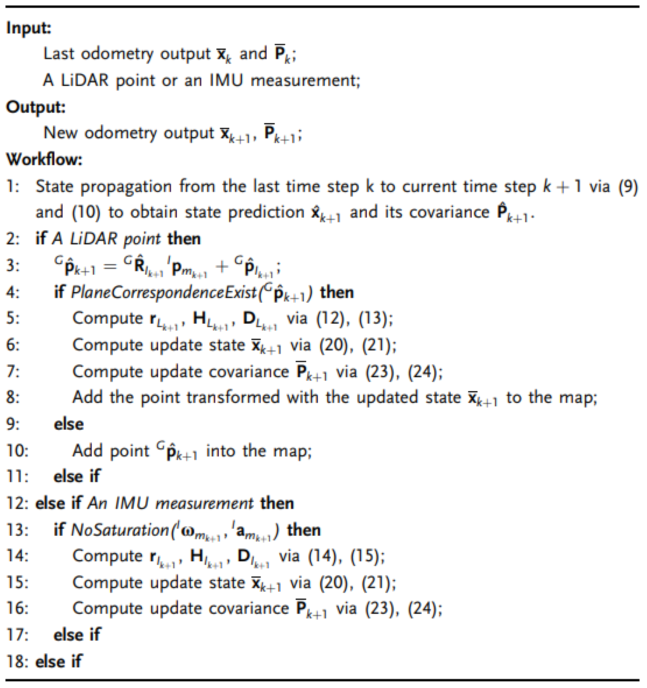
$${}^w g = \begin{bmatrix} 0 & 0 & -9.81 \end{bmatrix}^{\mathrm{T}} \tag{3}$$
$$\dot{b_a} \sim \mathcal{N}(0, \Sigma_{b_a}) \tag{4}$$
$$\eta_a \sim \mathcal{N}(0, \Sigma_{\eta_a}) \tag{5}$$

Now, the bias is chosen to be modeled as Brownian motion, which is a random walk. This means that the rate of change of the bias is modeled according to a normal distribution.

When we process an IMU measurement, we examine the residual:

$$h_{\mathrm{IMU}} = \begin{bmatrix} \hat{\omega} - {}^b\omega - b_\omega \\ f_a - {}^b a - {}^b a \end{bmatrix} \tag{6}$$

By using this residual in the Kalman update equations, we can correct the motion prediction of our robot.



```
Input:
    Last odometry output $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{P}}_k$;
    A LiDAR point or an IMU measurement;
Output:
    New odometry output $\bar{\mathbf{x}}_{k+1}$, $\bar{\mathbf{P}}_{k+1}$;
Workflow:
1:  State propagation from the last time step k to current time step $k+1$ via (9)
    and (10) to obtain state prediction $\hat{\mathbf{x}}_{k+1}$ and its covariance $\hat{\mathbf{P}}_{k+1}$.
2:  if A LiDAR point then
3:      ${}^G\hat{\mathbf{p}}_{k+1} = {}^G\hat{\mathbf{R}}_{I_{k+1}} {}^I\mathbf{p}_{m_{k+1}} + {}^G\hat{\mathbf{p}}_{I_{k+1}}$;
4:      if PlaneCorrespondenceExist(${}^G\hat{\mathbf{p}}_{k+1}$) then
5:          Compute $\mathbf{r}_{L_{k+1}}$, $\mathbf{H}_{L_{k+1}}$, $\mathbf{D}_{L_{k+1}}$ via (12), (13);
6:          Compute update state $\bar{\mathbf{x}}_{k+1}$ via (20), (21);
7:          Compute update covariance $\bar{\mathbf{P}}_{k+1}$ via (23), (24);
8:          Add the point transformed with the updated state $\bar{\mathbf{x}}_{k+1}$ to the map;
9:      else
10:         Add point ${}^G\hat{\mathbf{p}}_{k+1}$ into the map;
11:     else if
12: else if An IMU measurement then
13:     if NoSaturation(${}^I\boldsymbol{\omega}_{m_{k+1}}$, ${}^I\mathbf{a}_{m_{k+1}}$) then
14:         Compute $\mathbf{r}_{I_{k+1}}$, $\mathbf{H}_{I_{k+1}}$, $\mathbf{D}_{I_{k+1}}$ via (14), (15);
15:         Compute update state $\bar{\mathbf{x}}_{k+1}$ via (20), (21);
16:         Compute update covariance $\bar{\mathbf{P}}_{k+1}$ via (23), (24);
17:     else if
18: else if
```

Fig. 2: Algorithm from Point LIO
[5]

A. Voxel Grid

The original Point-LIO paper utilized a unique data structure to store LIDAR points in map. The name of that structure was an incremental KD-tree. This structure was so complicated that it warranted its own paper. The implementation was multi-threaded and relied directly on difficult to understand POSIX thread calls. The entry points for the threading were very unclear, and as a whole, attempts to use or replicate the structure were unsuccessful. Upon further investigation, we discovered that the Point-LIO authors had created a more recent branch doing away with the incremental KD-tree entirely. Rather than build upon their previous work, they opted for a simpler solution based on voxel grids. The voxel grid is a standard approach in point cloud mapping that has resulted in general success. It relies on the idea of sorting points into voxels, or cubes representing regions of space using a hashing function. These voxels contain a maximum number of points and allow for fast nearest neighbor lookup. They also have the advantage of being embarrassingly incremental. In contrast to a KD-tree, which relies on rebalancing, dynamic deletion, and other clever strategies to ensure real-time performance, adding and deleting points from the voxel grid is as simple as pushing back and popping off the front of the deque associated with the voxel. Additionally, the hashing function associated with finding the voxel of choice is fast and deterministic, easy to debug. We implemented our own voxel grid from scratch in C++, templated to accept any type that could return a 3-vector representing its coordinates in space. Our voxel grid found nearest neighbors extremely quickly, and constituted a large part of our efforts.

In practice, we chose a voxel size of .25 meters, and a maximum of 20 points per voxel. However, these parameters are easily changeable and can be adapted to the given environmental demands. Furthermore, we utilized the Cantor hash function to map each voxel coordinate to an unsigned integer. The Cantor hash function represents a bijective map from two non-negative integers to a non-negative integer. The purpose of the hash function is the ensure that the correct voxel is selected, which we can then use to find closest neighbors. The method to find the closest neighbors to a point is to run through all points in the corresponding voxel and then sort those points based on distance. Because of the small number of points contained in the voxel cell, this is the optimal approach.

Fig. 3: Velodyne VLP16

B. Velodyne LIDAR Decoding

Traditional methods for LIDAR processing involve understanding the structure of the point cloud–where is each point located? Because of the incremental nature of our approach, we needed to answer an additional question: at what time was each point collected? To answer this question, we delved into the technical reference manual for the Velodyne VLP-16, which was the LIDAR of choice for our project. The Velodyne returns a single time stamp associated with the start of each data packet, but within each packet, hundreds of points are collected. Because of this, we implemented our own driver processing Velodyne binary data into raw point clouds with timestamps associated with each point. Implementing this section required precise attention to each part of the timing and it had to run in real time. The manual that we followed can be found at https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf.

C. LIDAR Measurement

When we process a LIDAR point, we first check in the map for any possible matches using the voxel grid discussed earlier. Following this check, if there are less than five points within a 2 meter radius of the point itself, or if the five points do not form a good plane fit, then the new point is added to the map.

In the case that a good plane fit consisting of more than five points is found, we need to compute the residual, which

Fig. 4: LIVOX Avia Sensor

$$
\begin{aligned}
\mathbf{x}_{k+1} \boxminus \bar{\mathbf{x}}_{k+1} &= \left(\hat{\mathbf{x}}_{k+1} \boxplus \delta\mathbf{x}_{k+1}\right) \boxminus \bar{\mathbf{x}}_{k+1} \\
&\approx \underbrace{\left(\hat{\mathbf{x}}_{k+1} \boxplus \delta\mathbf{x}^{o}_{k+1}\right) \boxminus \bar{\mathbf{x}}_{k+1}}_{} + \mathbf{J}_{k+1}\left(\delta\mathbf{x}_{k+1} - \delta\mathbf{x}^{o}_{k+1}\right)
\end{aligned}
$$

Fig. 5: Equation showing the Jacobian source for relative covariance

$$
\mathbf{J}_{k+1} = \left.\frac{\partial\left(\left(\hat{\mathbf{x}}_{k+1} \boxplus \delta\mathbf{x}\right) \boxminus \bar{\mathbf{x}}_{k+1}\right)}{\partial \delta\mathbf{x}}\right|_{\delta\mathbf{x} = \delta\mathbf{x}^{o}_{k+1}} = \begin{bmatrix} \mathbf{A}(\theta_{k+1})^{-T} & \mathbf{0}_{3\times 21} \\ \mathbf{0}_{3\times 21} & \mathbf{I}_{21\times 21} \end{bmatrix}
$$

$$
\mathbf{A}(\mathbf{u})^{-1} = \mathbf{I} - \frac{\lfloor \mathbf{u} \rfloor}{2} + \left(1 - \frac{\|\mathbf{u}\|}{2}\cot\left(\frac{\|\mathbf{u}\|}{2}\right)\right)\frac{\lfloor \mathbf{u} \rfloor}{\|\mathbf{u}\|}, \theta_{k+1} = {}^{G}\overline{\mathbf{R}}_{I_{k+1}} \boxminus {}^{G}\hat{\mathbf{R}}_{I_{k+1}}
$$

Fig. 6: Equation showing the Jacobian formulation

is the inner product of the normal vector, n of the plane to the LIDAR point expressed relative to a point on the plane itself.

$$h_{\mathrm{LIDAR}} = n^{\mathrm{T}} p \tag{7}$$

D. Kalman Update Equations

The Kalman Update equations are given by calculating the Jacobians with respect the state of the measurement residual. It's important to get these Jacobians right for optimal performance. We noticed that the Jacobians are very sparse–with just a few nonzero entries in each matrix. Because of this, we opted to make the Jacobians use the Eigen::Sparse formulation, and this provided speedups for our work.

The means by which we performed the update also took into account the fact that absolute covariance for odometry systems is not very practical. Rather, a relative covariance is more appropriate, where the covariance represents the uncertainty with respect to the previous true state. Because of this, the Kalman update equations were modified slightly to accommodate this. The equation to do so is given by Figures 5, 6, and 7, taken from Point-LIO.

$$\overline{\mathbf{P}}_{k+1} = \mathbf{J}_{k+1}\mathbf{P}_{k+1}\mathbf{J}^{T}_{k+1}$$

Fig. 7: Equation showing the covariance propagation from the absolute to relative case

The need to understand the meaning behind each LIDAR point is one of the best strengths of Point-LIO. However, this approach was not without its shortcomings.

E. Shortcomings

We found in total three shortcomings of the Point-LIO algorithm that made it challenging to implement in real-time.

1) Shortcomings: We found that given the high number of points per scan, it was generally impossible to achieve the real-time performance we sought for. This seemed to defeat the very reason for doing Point-LIO. However, we still feel it is a valuable result from a theoretical perspective. Regardless, the lack of parallelism for processing individual LIDAR points greatly slows down any LIDAR-Inertial algorithm.

2) Extended Kalman Filter: A second shortcoming was the added linearization at each time step from the Extended Kalman Filter. This linearization means that we lose information in a very fast way between sampling. The result is a less-than-optimal state estimate, especially during times of aggressive motion, which was the point of the algorithm in the first place.

3) IMU and LIDAR Time Synchronization: A third shortcoming was the very involved need to synchronize the IMU and LIDAR point measurements across time. Unfortunately, due to the fact that LIDAR measurements arrive in a packet of 30,000 points every tenth of a second, their distribution across any given time window is non-uniform. The result is that we can no longer assume that all LIDAR measurements have been received prior to a given IMU measurement. And vice versa. We decided to overcome this issue by allocating individual buffers for both IMU and LIDAR points. When a IMU measurement was received, we processed all LIDAR points before the IMU measurement. Similarly, when a LIDAR point was received, we processed all IMU measurements prior to the time stamp associated with the LIDAR point. Unfortunately, this still resulted in negative time differences between the current state of the filter, and the time stamp associated with the current measurement being processed. Ultimately, the time synchronization required to achieve this optimal state estimate is nontrivial and was a significant source of problems later on in our approach. However, we were able to show nominal performance given an IMU by itself.

V. Results

Our code is available at https://github.com/taylorpool/point_lio.

The results for IMU dead reckoning and IMU data modeled as output of the system are shown in fig. 8 and fig. 9 respectively.

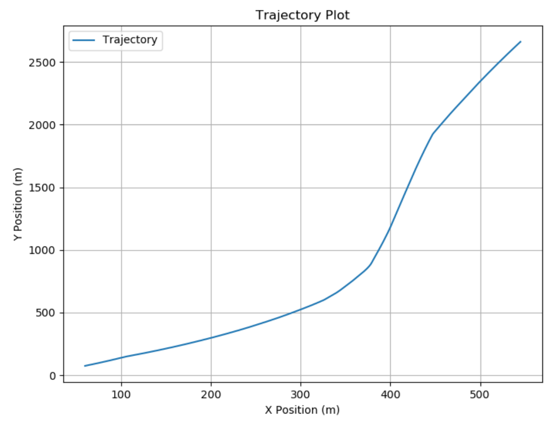


Fig. 8: Trajectory and Landmark Visualization

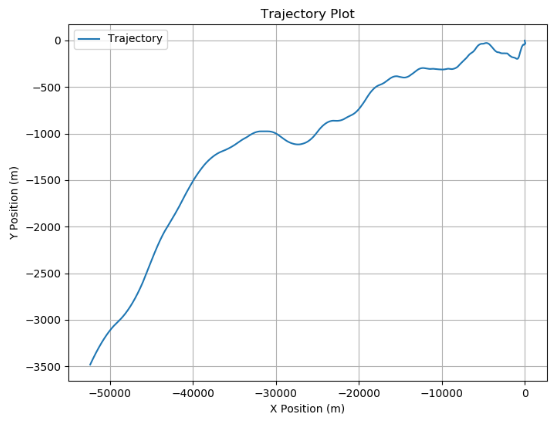


Fig. 9: Trajectory and Landmark Visualization

VI. Conclusion

A. Learning

We learned about the methodology utilized in a Kalman Filter, specifically how zero-mean noise plays a vital role in the residual and update functions.

We also learned that motion distortion can be entirely short-circuited if one is willing to pay the price of nonparallel structures and algorithms.

Furthermore, we learned that the added complexity associated with an incremental KD-tree is not necessary and does not constitute enough of a performance gain to warrant its use.

Implementing these algorithms in C++ was a valuable experience that taught us much about static and dynamic matrices, as well as dense and sparse matrices.

Finally, we also learned about processing raw Velodyne packet data into individual points with time stamps. The drivers currently out in the field do not have this capability, which was very surprising to us. We postulate that many people are not worried about motion distortion and prefer to ignore it entirely.

B. Final Verdict

We have presented our findings implementing Point-LIO: an algorithm for LIDAR-Inertial Odometry that seeks to replace scan registration with point-by-point registration. We implemented our approach in C++ with a significant amount of effort dedicated towards mapping, processing LIDAR points, and also forming the Extended and Unscented Kalman Filters on-manifold. We found that Point-LIO is unsuitable for real-time computation given the inherently sequential nature of the processing requirements. Furthermore, we found that the time synchronization between each individual LIDAR point and IMU measurement to be difficult to implement properly and wholly non-intuitive. The result was that we simply do not recommend this algorithm for future implementations of LIDAR-Inertial Odometry. However, we do recommend further investigations into LIDAR-Inertial Odometry as a whole, because this is a field that has gained much relevance through the advent of self-driving car technology.

Acknowledgment

The authors would like to thank the creators of Point-LIO, and the instructors/teaching assistants of 16-833 for their insights during the class.

References

[1] Paul J Besl and Neil D McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. Spie. 1992, pp. 586–606.

[2] Yixi Cai, Wei Xu, and Fu Zhang. *ikd-Tree: An Incremental K-D Tree for Robotic Applications*. 2021. arXiv: 2102.10808 [cs.RO].

[3] Frank Dellaert, Michael Kaess, et al. "Factor graphs for robot perception". In: *Foundations and Trends® in Robotics* 6.1-2 (2017), pp. 1–139.

[4] Pierre Dellenbach et al. "Ct-icp: Real-time elastic lidar odometry with loop closure". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 5580–5586.

[5] Dongjiao He et al. "Point-LIO: Robust High-Bandwidth Light Detection and Ranging Inertial Odometry". In: *Advanced Intelligent Systems* 5.7 (2023), p. 2200459.

[6] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[7] J. Lin and F. Zhang. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Virtual: IEEE, May 2020, pp. 3126–3131.

[8] J. Quenzel and S. Behnke. "Title of the Paper". In: *2021 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. Prague, Czech Republic: IEEE, Sept. 2021, pp. 5499–5506.

[9] Tixiao Shan et al. *LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping*. 2020. arXiv: 2007.00258 [cs.RO].

[10] Ignacio Vizzo et al. "Kiss-icp: In defense of point-to-point icp–simple, accurate, and robust registration if done the right way". In: *IEEE Robotics and Automation Letters* 8.2 (2023), pp. 1029–1036.

[11] Eric A Wan and Rudolph Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 adaptive systems for signal processing, communications, and control symposium (Cat. No. 00EX373)*. Ieee. 2000, pp. 153–158.

[12] Greg Welch, Gary Bishop, et al. "An introduction to the Kalman filter". In: (1995).

[13] W. Xu and F. Zhang. In: *IEEE Robotics and Automation Letters* 6 (2021), p. 3317.

[14] Wei Xu et al. "Fast-lio2: Fast direct lidar-inertial odometry". In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2053–2073.

[15] J. Zhang and S. Singh. "Title of the Paper". In: *Robotics: Science and Systems*. Vol. 2. Berkeley, USA, July 2014, pp. 1–9.

[16] Ji Zhang and Sanjiv Singh. "LOAM: Lidar odometry and mapping in real-time." In: *Robotics: Science and systems*. Vol. 2. 9. Berkeley, CA. 2014, pp. 1–9.

[17] Shibo Zhao et al. *Super Odometry: IMU-centric LiDAR-Visual-Inertial Estimator for Challenging Environments*. 2021. arXiv: 2104.14938 `[cs.RO]`.

Taylor Pool Taylor Pool is a Master's Student in Robotics at Carnegie Mellon University. His research lies in LIDAR-Inertial Odometry formulations and how to push the state of the art in the field.

Richa Mohta Richa Mohta is a Master's Student in Mechanical Engineering at Carnegie Mellon University. Her research is in the field of motion planning in unstructured environments.

Sahil Chaudhary Sahil Chaudhary is a Master's Student in Mechanical Engineering at Carnegie Mellon University. His research lies in controls and planning.

Shreyansh Sharma Shreyansh Sharma is a Master's Student in Mechanical Engineering at Carnegie Mellon University. His research is in the field of Computer Vision and Deep Learning Methods .

Appendix
The Incremental KD-Tree

1) **Construction**
The incremental KD-Tree (iKD-Tree) starts with a set of k-dimensional points, much like a regular KD-Tree. To create a balanced tree, these are split recursively into two subgroups, usually by a median split along one of the dimensions. Sorting can be highly difficult when attempting to identify the median at each break. It takes $O(n \log n)$ time to sort all the points along a single dimension and get the median, where n is the number of points. At every level of the tree, one splits and sorts the data recursively to create a balanced KD-Tree:
    - Every n points are sorted at the first level.
    - The two subsets (about $\frac{n}{2}$ points each) are sorted at the second level.
    - This keeps going until there is just one point in every subgroup.

    This is the very reason why the iKD-Tree is used. The KD-Tree is not suited for dynamic points and hence rebuilding the KD-tree every time for the complexity of $O(n \log n)$ is not recommended.

2) **Incremental Updates**
The unique feature of an iKD-Tree is its ability to efficiently handle incremental updates:
    Insertions: When a new point needs to be inserted, the point is added by traversing the tree from the root to a leaf, following the binary search property based on the point's coordinates relative to the splitting planes. After insertion, some balancing may be needed to maintain efficient search times.
    Deletions: To delete a point, it is located and removed by traversing the tree. If the removal of a point unbalances the tree, rotations or more complex restructuring might be required.

3) **Searching**
    Nearest Neighbor Search: The tree is traversed beginning at the root in order to determine the closest neighbour of a particular location. The approach examines the half of the tree that contains the target point recursively after determining the distance to the point recorded at each node. If the closest point discovered thus far is closer to the dividing plane than it is to the point itself, it might also search the other half.
    Range SearchRange searches work similarly to closest neighbour searches in that they traverse the tree and gather all points that are within a given range (distance) of a target point.

4) **Advantages of iKD-Tree**
    Dynamic Updating:Unlike standard KD Trees, iKD-Trees can handle dynamic data sets where points are frequently added or removed.

Balanced Structure: It maintains a balanced tree structure more effectively, which is crucial for maintaining good query performance.