Adaptive Covariance Estimation of LiDAR-based Positioning Errors for UAVs

Akshay Shetty, Member, IEEE, and Grace Xingxin Gao, Senior Member, IEEE

Abstract—Outdoor positioning for Unmanned Aerial Vehicles (UAVs) commonly relies on global navigation satellite system (GNSS) signals, which might be reflected or blocked in urban areas. Thus, additional on-board sensors such as Light Detection and Ranging (LiDAR) are desirable to aid positioning. To fuse measurements from different sensors, it is important, yet challenging, to accurately characterize the error covariance matrices of individual sensor measurements. In this paper, we propose a novel method for adaptively estimating the LiDAR-based positioning error covariance matrix based on the point cloud features surrounding the UAV. We model the position error as a multivariate Gaussian distribution and estimate its covariance matrix from individual surface and edge feature points. To validate our algorithm, we perform simulations and show that our covariance matrix model is more accurate than a distance-based covariance matrix model. Furthermore, we conduct an outdoor experiment and implement a Kalman Filter (KF) to fuse global positioning system (GPS) signals and LiDAR position measurements. We demonstrate a clear improvement in the UAV’s global position estimation using our adaptive covariance matrix for LiDAR-based measurements.

Keywords—Unmanned aerial vehicles (UAVs), light detection and ranging (LiDAR), covariance estimation, sensor fusion, kalman filter (KF)

I. INTRODUCTION

Emerging applications in UAVs such as 3D modeling, filming, surveying, search and rescue, and delivering packages, involve flying in urban environments. In these scenarios, autonomously navigating a UAV has certain advantages such as optimizing flight paths and sensing and avoiding collisions [1], [2]. To enable such autonomous control, we need a continuous and reliable source for the UAVs’ positioning. In most cases, GNSS is primarily relied on for outdoor positioning. However, in an urban environment, GNSS signals from the satellites are often blocked or reflected by surrounding structures, causing large errors in the position output [3].

In cases when GNSS is unreliable, additional on-board sensors such as LiDAR are commonly used to obtain the navigation solution [4], [5]. For simplicity, we refer to the LiDAR sensor as just LiDAR in this paper. An on-board LiDAR provides a real-time point cloud of the surroundings of the UAV. In dense urban environments, a LiDAR is able to detect a large number of features from surrounding structures. Positioning based on LiDAR point clouds has been demonstrated primarily by applying different simultaneous localization and mapping (SLAM) algorithms [6]. In many cases, algorithms implement variants of Iterative Closest Point (ICP) [7] to register new point clouds. ICP takes two point clouds as inputs and iteratively estimates the translation and rotation between them by minimizing an error metric such as point-to-point or point-to-plane distances [8], [9]. Other methods not dependent on ICP [10], [11], include point cloud matching based on polar coordinates of points [12] and tracking based on features in a point cloud [13], [14]. Implementation of some LiDAR-based SLAM algorithms [15]–[17] are available online. In our work, we implement ICP [18] for point cloud registration since it is simple, popular, and many researchers have a first-hand experience with it. Although many variants exist, the vanilla version is used (point-to-point distance, linear interpolation in the reference scan).

However, the accuracy of the aforementioned positioning algorithms depends on the distribution of features around the LiDAR. Figure 1 shows three different urban scenarios and the effect of point cloud features on LiDAR-based positioning estimates. Previous methods to analyze the position error covariance matrix have been demonstrated in simulations and practice [19]–[21]. These include training kernels based on likelihood optimization [22], [23] and obtaining the covariance matrix based on scanned lines [24]. The seminal work by Censi [25], [26] computes the covariance matrix based on the Fisher Information matrix and the Jacobian of the ICP measurement model. In [27] the authors explore the limitations of Censi’s covariance which does not explicitly account for the rematching step. Censi’s covariance also assumes the same sensor noise for all points in the point cloud. In practical cases, due to
different surface textures and reflectivity different points have different error distributions. Furthermore, the above methods were proposed for 2-dimensional (2D) LiDARs. With recent advances in sensor technology, 3D LiDARs are being increasingly used for autonomous applications. These 3D LiDARs capture a significantly wider range of surrounding information compared to 2D LiDARs.

Our work presents a novel approach where we intuitively estimate the covariance matrix for 3D LiDARs. We hypothesize that LiDAR-based positioning errors are affected by the surface and edge feature points in the point cloud. We model the position error due to each feature as a multivariate Gaussian distribution whose covariance matrix is characterized based on the feature location and orientation relative to the LiDAR. Finally, we combine all the individual distributions to obtain the LiDAR-based positioning error covariance matrix.

We focus our approach only on the ICP positioning output since we have an intuitive heuristic for the uncertainty in positioning errors, in contrast to the orientation errors.

In this paper we build on our previous work in [28]. Specifically, we present simulations to validate and verify our covariance matrix model; we provide a more discussion about the point cloud feature extraction and covariance matrix modelling; we present additional experimental results by comparing our method to the covariance estimation work in [25]. The algorithm in [25] suggests a definite magnitude under an assumption that there is no wrong convergence of ICP. To ensure a fair comparison, we aim to minimize the chances of wrong convergence for a point cloud pair by performing multiple ICP instances, each with different initial guesses. This increases the chances of at least one of the initial guesses falling within the attraction area of the true solution. The ICP output is set to the instance output with the minimum root-mean-square matching error.

The rest of the paper is organized as follows. Section II discusses our feature extraction algorithm, and the steps taken to obtain our LiDAR-based positioning error covariance matrix. Section III introduces our simulation environment to generate LiDAR point clouds. It evaluates the ICP algorithm in different scenes and compares the performance of our adaptive covariance matrix versus a baseline distance-based covariance matrix. Section IV discusses experimental results obtained for an outdoor dataset. It describes a KF to fuse GPS and LiDAR position measurements, and shows the advantage of using our adaptive covariance matrix. Finally, Section V concludes the paper.

II. LI DAR-BASED POSITION ERROR COVARIANCE

For LiDAR-based positioning, we use an ICP implementation [18] to estimate the translation between two point clouds. We include the ‘WorstRejection’ option in [18] which rejects a given percentage of the worst point pairs every iteration. Finally, the ICP output can be represented as:

$$\hat{\Delta}x = \Delta x + \nu,$$

where $$\hat{\Delta}x$$ and $$\Delta x$$ are the ICP estimated translation and the true translation between two point clouds; $$\nu$$ is the estimation error which is assumed to be drawn from a zero mean multivariate Gaussian distribution: $$\nu \sim \mathcal{N}(0, R)$$.

In this section, we adaptively construct the covariance matrix $$\hat{R}_{\text{adaptive}}$$ to estimate $$R$$ as a function of features in the LiDAR point cloud. We focus primarily on surface and edge features, since in urban environments we typically observe structured objects. Additionally, most commonly occurring objects can be represented as a combination of these features.

A. Feature Extraction

For extracting surface and edge features from the point cloud, we follow the method described in [13], [14]. We use similar parameters as [13], [14] since they show the features to be a good representation of the point cloud and obtain state-of-the-art odometry. The feature extraction algorithm can be summarized as:

- For each point in the point cloud, compute the curvature $$c_i$$ as follows:

$$c_k = \frac{1}{\lVert P_k \rVert} \sum_{l \in B, l \neq k} \lVert p_k - p_l \rVert,$$

where:

- $$p_k$$: Coordinates of $$k^{th}$$ point relative to the LiDAR
- $$B$$: Neighborhood of 20 points near $$k$$.

- Discard points on occluded objects and stray points. These are detected based on the range and distance between consecutive points.
- Divide the point cloud into 6 equal angular sections of 60° each. For each section, sort the curvature values. Points with curvature values below 0.01 are assumed to be on a plane and are classified as surface points. Points with curvature values above 0.3 are assumed to be at an edge and are classified as edge points. A maximum of 15 surface points and 10 edge points are chosen from each of the 6 sections.

- If a point is classified as either a surface or an edge point, 10 neighboring points on each side are discarded.
B. Constructing Individual Covariance Matrices

1) Surface Feature Points: For each surface feature point \( i \), our goal is to model the covariance matrix \( \mathbf{R}_i^s \) associated with \( \nu_{si} \). Here \( \nu_{si} \) is the observation of \( \nu \) from (1) due to the \( i^{th} \) surface feature. We first compute the surface normal \( \hat{s}_i = [\hat{s}_{i1}, \hat{s}_{i2}, \hat{s}_{i3}] \) by using available MATLAB functions [29]. To obtain the surface normal, we use 50 neighboring points in order to sufficiently capture the local surface plane. To create a 3D error ellipsoid for the surface feature point, we then create an orthonormal basis with the corresponding normal. We select a vector \( \vec{n}_i^s \) that is perpendicular to \( s_i \) as follows:

\[
\vec{n}_i^s = [0 \ -\hat{s}_{i3} \ \hat{s}_{i2}] 
\]

(3)

Then we select \( \vec{m}_i^s \) as a cross product between \( \vec{n}_i^s \) and \( \hat{s}_i \) to create an orthonormal basis:

\[
\vec{m}_i^s = \vec{n}_i^s \times \hat{s}_i 
\]

(4)

We normalize \( \vec{n}_i^s \) and \( \vec{m}_i^s \) to obtain the following basis: \( \{\hat{s}_i, \vec{n}_i^s, \vec{m}_i^s\} \).

\[
\hat{n}_i^s = \frac{\vec{n}_i^s}{\|\vec{n}_i^s\|} \quad \hat{m}_i^s = \frac{\vec{m}_i^s}{\|\vec{m}_i^s\|} 
\]

(5)

After creating the orthonormal basis, we proceed to create the error covariance matrix. We use the basis as our eigenvectors:

\[
\mathbf{V}_i^s = [\hat{s}_i \ \hat{n}_i^s \ \hat{m}_i^s] 
\]

(6)

For the error covariance matrix, we hypothesize that each surface feature point contributes in reducing position error in the direction of the corresponding surface normal. Additionally, we assume that surface points closer to the LiDAR are more reliable than those further away, because the density of points and the LiDAR accuracy deteriorates with distance. We do this by weighing each error covariance matrix by the distance to the corresponding surface feature. Hence, we use the following eigenvalues corresponding to the eigenvectors in (6):

\[
\mathbf{L}_i^s = d_i^s \begin{bmatrix}
a^s & b^s & c^s \\
b^s & d^s & e^s \\
c^s & e^s & f^s 
\end{bmatrix}, 
\]

(7)

where:

- \( a^s, b^s \) : Constants for all surface points such that: \( a^s \ll b^s \)
- \( d_i^s \) : Distance of the \( i^{th} \) surface point from the LiDAR.

We empirically set the constants \( a^s = 0.2m \) and \( b^s = 25m \). Finally, using the eigenvectors \( \mathbf{V}_i^s \) and the eigenvalues \( \mathbf{L}_i^s \), we construct the position error covariance matrix for the surface point as follows:

\[
\mathbf{R}_i^s = \mathbf{V}_i^s \cdot \mathbf{L}_i^s \cdot \mathbf{V}_i^s^{-1} 
\]

(8)

Figure 3(a) shows a sample error ellipsoid for a surface point generated by (8).

2) Edge Feature Points: Similar to surface points, for each edge point we model the covariance matrix \( \mathbf{R}_j^e \) associated with \( \nu_{ej} \). Here \( \nu_{ej} \) is the observation of \( \nu \) from (1), due to the \( j^{th} \) edge feature. We first find the orientation of the edge by finding the closest edge points in both the scans above and below \( j \). If the distance between \( j \) and the closest edge points is below a threshold, we use the points to estimate the edge orientation. Thus we obtain the edge vector \( \hat{e}_j \). Next, we follow similar steps as we did for each surface point. We first create an orthonormal basis with \( \hat{e}_j \):

\[
\hat{n}_j^e = [0 \ -\hat{e}_{j3} \ \hat{e}_{j2}] 
\]

(9)

\[
\hat{m}_j^e = \hat{n}_j^e \times \hat{e}_j 
\]

(10)

After normalizing \( \hat{n}_j^e \) and \( \hat{m}_j^e \), as done in (5), we obtain the required basis \( \{\hat{e}_j, \hat{n}_j^e, \hat{m}_j^e\} \) which we use as the eigenvectors:

\[
\mathbf{V}_j^e = [\hat{e}_j \ \hat{n}_j^e \ \hat{m}_j^e] 
\]

(11)

For the error covariance matrix, we model the error covariance ellipsoid with the hypothesis that each edge feature point helps in reducing position error in the directions perpendicular to the edge vector. A vertical edge, for example, would help in reducing horizontal position error. Additionally, we again assume that edge points closer to the LiDAR are more reliable than those further away, because the density of points and the LiDAR accuracy deteriorates with distance. We do this by weighing each error covariance matrix by the distance to
the corresponding edge feature. Hence, we use the following eigenvalues corresponding to the eigenvectors in (11):

$$L_j^e = d_j^e \begin{bmatrix} a^e & \cdots & b^e \\ \vdots & \ddots & \vdots \\ \vdots & \cdots & b^e \end{bmatrix},$$  

(12)

where:

- $a^e, b^e$: Constants for all edge points such that: $a^e \gg b^e$
- $d_j^e$: Distance of the $j^{th}$ edge point from the LiDAR.

Again, we empirically set the constants $a^e = 25m$ and $b^e = 0.5m$. Finally, using the eigenvectors $V_j^e$ and the eigenvalues $L_j^e$, we construct the position error covariance matrix for the edge point as follows:

$$R_j^e = V_j^e \cdot L_j^e \cdot V_j^e^{-1}$$  

(13)

Figure 3(b) shows a sample error ellipsoid for an edge point generated by (13).

C. Combining Covariance Matrices

Once we have modeled the covariance matrices due to individual features, we proceed to estimate the distribution of $\nu$ given observations due to surrounding surface and edge features: $p(\nu|\nu_S, \nu_E)$. Using Bayes rule, the above distribution can be written as:

$$p(\nu|\nu_S, \nu_E) = \frac{p(\nu)}{p(\nu_S, \nu_E)} \prod_{i=1}^{n} p(\nu_{s_i}) \prod_{j=1}^{m} p(\nu_{e_j})$$  

(14)

As long as the prior probability distribution function $p(\nu)$ is constant over its area of interest, it will cancel out with $p(\nu_S, \nu_E)$ and $p(\nu_{s_i})$ and $p(\nu_{e_j})$ to affect $p(\nu|\nu_S, \nu_E)$ only as a scaling factor [30]. Thus:

$$p(\nu|\nu_S, \nu_E) = K \prod_{i=1}^{n} p(\nu|\nu_{s_i}) \prod_{j=1}^{m} p(\nu|\nu_{e_j}),$$  

(15)

where $K$ is the scaling factor. Since we model $p(\nu|\nu_{s_i})$ and $p(\nu|\nu_{e_j})$ as Gaussian, the conditional probability distribution function can be written as:

$$p(\nu|\nu_S, \nu_E) = K_1 \prod_{i=1}^{n} \exp \left\{ -\frac{1}{2} (\nu - \nu_{s_i})^T (R_{s_i}^e)^{-1} (\nu - \nu_{s_i}) \right\} \prod_{j=1}^{m} \exp \left\{ -\frac{1}{2} (\nu - \nu_{e_j})^T (R_{e_j}^e)^{-1} (\nu - \nu_{e_j}) \right\}$$  

(16)

$$p(\nu|\nu_S, \nu_E) = K_1 \exp \left\{ -\frac{1}{2} \sum_{i=1}^{n} (\nu - \nu_{s_i})^T (R_{s_i}^e)^{-1} (\nu - \nu_{s_i}) -\frac{1}{2} \sum_{j=1}^{m} (\nu - \nu_{e_j})^T (R_{e_j}^e)^{-1} (\nu - \nu_{e_j}) \right\}$$  

(17)

The optimal estimate for any symmetric cost function is the same for Gaussian distributions and results in the minimum of the argument of the exponential in (17). The covariance for the minimum [30] is given by:

$$\hat{R}_{adaptive} = \left( \sum_{i=1}^{n} (R_{s_i}^e)^{-1} + \sum_{j=1}^{m} (R_{e_j}^e)^{-1} \right)^{-1}$$  

(18)

III. Simulations

In order to analyze the LiDAR-based positioning errors we created a new simulation environment based on the Unity game engine [31]. We chose a simulated environment to validate our algorithm because of the following reasons: precise ground truth information is available, experiments can be performed multiple times to generate large amounts of data, and a variety of scenes can be easily set up. The goal of the simulations is to generate a reference position error covariance matrix $\hat{R}$ at multiple locations and to analyze the performance of our adaptive covariance matrix versus a distance-based covariance matrix model.
A. Simulation Setup

In this section, we simulate the Velodyne VLP-16 [32] LiDAR, which we use as our LiDAR sensor for experiments in section IV-B. Based on the specifications of the LiDAR, we set the range of the LiDAR to 100m and add a zero mean Gaussian noise $\mathcal{N}(0, 0.03m)$ to the range measurements. We perform simulations in three different scenes as shown in Figure 4. The first scene is an enclosed scene within four walls in the formation of a square. The square side is set to 300m in order to include certain challenging point clouds such as having a wall just on one side, or having no features at all. The second scene is a small town that consists of a variety of objects such as houses, vehicles, lamp posts, fences, vegetation, etc. We choose this scene in order to simulate low-altitude UAV flights in dense urban regions. The third scene is a large urban city that primarily consists of structures such as bridges and tall buildings. This scene simulates low and medium altitude flights in urban cities.

B. Constructing Covariance Matrices

First, we select a random position $x^0$ in the simulation scene and generate the corresponding LiDAR point cloud $\mathcal{P}^0$. We explain below in section III-D the process of choosing these positions. Next, we construct three types of covariance matrices at each position $x^0$: a reference covariance matrix $\mathbf{R}$, our adaptive covariance matrix $\mathbf{R}_{\text{adaptive}}$, and distance-based covariance matrices $\mathbf{R}_{\text{dist}}$.

To construct the reference covariance matrix $\mathbf{R}$, we begin by generating 200 normally distributed sample positions around $x^0$ with 1m standard deviation in each direction: $x^i \sim \mathcal{N}(x^0, (1m)^2)$. Figure 5 shows an example of sample positions obtained from such a distribution. We generate the LiDAR point clouds $\mathcal{P}^i$ at each of these sample positions $x^i$. Next, for each sample, we use the ICP algorithm on $\mathcal{P}^0$ and $\mathcal{P}^i$ to estimate the translation $\Delta x^i$ and consequently the sample position $\hat{x}^i$. Then, we stack all the individual ICP errors $\nu^i = (\hat{x}^i - x^i)$ to create the error matrix $\nu$. Figure 6 shows the distribution of the errors for the same example as in Figure 5. Finally, we construct the reference covariance matrix as follows:

$$\mathbf{R} = \begin{bmatrix} \text{cov}(\nu_x, \nu_x) & \text{cov}(\nu_x, \nu_y) & \text{cov}(\nu_x, \nu_z) \\ \text{cov}(\nu_y, \nu_x) & \text{cov}(\nu_y, \nu_y) & \text{cov}(\nu_y, \nu_z) \\ \text{cov}(\nu_z, \nu_x) & \text{cov}(\nu_z, \nu_y) & \text{cov}(\nu_z, \nu_z) \end{bmatrix}, \quad (19)$$

where:

$$\text{cov}(\eta, \zeta) = \frac{1}{N-1} \sum_{i=1}^{N} (\eta_i - \mu_\eta)(\zeta_i - \mu_\zeta)$$

is the covariance for two random variable vectors $\eta$ and $\zeta$ with means $\mu_\eta$ and $\mu_\zeta$ respectively. We assume that the reference covariance matrix adequately captures the covariance of the LiDAR-based positioning errors.

To construct our adaptive covariance matrix, we use the initial point cloud $\mathcal{P}^0$ and follow the procedure described in section II. We obtain $\mathbf{R}_{\text{adaptive}}$ from (18). We obtain our distance-based covariance matrices as follows:

$$\hat{\mathbf{R}}_{\text{dist}} = \sigma_{\text{dist}} \cdot \left( \sum_{i=1}^{n} (d^i_s)^{-1} + \sum_{j=1}^{m} (d^j_e)^{-1} \right)^{-1} \cdot \mathbb{I}, \quad (20)$$

where $\sigma_{\text{dist}}$ is a tuned weight, $d^i_s$ and $d^j_e$ are the distances of the $i^{th}$ surface point and $j^{th}$ edge point from the LiDAR, and $\mathbb{I}$ is a $3 \times 3$ identity matrix. We vary $\sigma_{\text{dist}}$ from 0.1 to 2 as explained below in section III-D.

C. Comparison Metrics

We use two metrics to validate the accuracy of our adaptive covariance matrix. For our first metric, we use the metric proposed in [33] which is commonly used in the computer vision community. For simplicity, we refer to it as the Förstner metric in the remainder of the paper. The metric can be computed between two covariance matrices $\mathbf{A}$ and $\mathbf{B}$ as follows:

$$d^F(\mathbf{A}, \mathbf{B}) = \sqrt{\sum_{i=1}^{n} \ln^2 \lambda_i(\mathbf{A}, \mathbf{B})}, \quad (21)$$

where $\lambda_i(\mathbf{A}, \mathbf{B}), i = 1 \cdots n$ are the generalized eigenvalues obtained from $|\lambda \mathbf{A} - \mathbf{B}|$. Here $d^F(\mathbf{A}, \mathbf{B}) \in (0, \infty)$ and a lower value indicates a higher similarity between the covariance matrices. For our second metric, we use the correlation matrix distance proposed in [34]. We include this metric to evaluate how accurately our adaptive covariance matrix captures the shape of the reference covariance matrix. For simplicity, we refer to it as the Herdin metric in the remainder of the paper. The metric can be computed between two covariance matrices $\mathbf{A}$ and $\mathbf{B}$ as follows:

$$d^H(\mathbf{A}, \mathbf{B}) = 1 - \frac{\text{trace}(\mathbf{A}, \mathbf{B})}{\lVert \mathbf{A} \rVert_f \lVert \mathbf{B} \rVert_f}, \quad (22)$$

where $\lVert \cdot \rVert_f$ represents the Frobenius norm of a matrix. Here $d^H(\mathbf{A}, \mathbf{B}) \in (0, 1)$ and a lower value indicates a higher similarity between the shape of the covariance matrices.
D. Simulation Procedure and Results

Algorithm 1 explains our simulation procedure to construct the covariance matrices (from section III-B) and to compute the Förstner and the Herdin metrics (from section III-C). For each scene we pick 100 uniformly random positions within the scene bounds where the UAV does not collide with objects. At each position, the reference covariance matrix is computed from the ICP estimation errors for 200 samples, as described in section III-B. Next, we compute the Förstner and Herdin metrics, comparing the reference covariance matrix with our adaptive covariance matrix and a range of distance-based covariance matrices. Finally, we compute the mean metrics for each scene by averaging over the metrics computed at all the 100 positions.

Figure 7 shows the mean Förstner metrics obtained for each scene (steps 18-21 in Algorithm 1). We compare the metrics for our adaptive covariance matrix versus a distance-based covariance matrix model. We observe that our adaptive covariance matrix performs better than the distance-based covariance matrices for all three scenes. The improvement is large in structured environments (scenes A and C) since the primary features are surfaces and edges. In unstructured environments (scene B), the improvement is smaller, but our model still performs better than the distance-based covariance matrix model.

The Herdin metric is a measure of how accurately the shape of the covariance matrices match, and hence (22) involves normalization w.r.t. the matrix norms. Thus, the Herdin metric comparing a reference covariance matrix with a range of distance-based covariance matrices remains constant w.r.t. $\sigma_{\text{dist}}$. Below is a comparison of the mean Herdin metrics for all scenes:

<table>
<thead>
<tr>
<th>Scene</th>
<th>$d^H_{\text{adaptive}}$</th>
<th>$d^H_{\text{dist}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0479</td>
<td>0.3143</td>
</tr>
<tr>
<td>B</td>
<td>0.2429</td>
<td>0.2868</td>
</tr>
<tr>
<td>C</td>
<td>0.1152</td>
<td>0.3546</td>
</tr>
</tbody>
</table>

TABLE I: Comparison between mean Herdin metric for our adaptive covariance matrix ($d^H_{\text{adaptive}}$) and for distance-based covariance matrices ($d^H_{\text{dist}}$).

We observe that our adaptive covariance matrix performs better than a distance-based covariance matrix model w.r.t. the mean Herdin metric for all three scenes. Similar to the Förstner metric, the improvement is larger in the structured scenes A and C.
Algorithm 1 Simulation Procedure

1: for each simulation scene do
2:     for j = 1 to 100 do
3:         \( \hat{x}_j^0 \leftarrow \) new random position in scene
4:         \( P_j^0 \leftarrow \) LiDAR point cloud at \( \hat{x}_j^0 \)
5:         for i = 1 to 200 do
6:             \( \hat{x}_j^i \leftarrow \) new position \( \sim \mathcal{N}(\hat{x}_j^0, (1m)I) \)
7:             \( P_j^i \leftarrow \) LiDAR point cloud at \( \hat{x}_j^i \)
8:             \( \Delta x_j^i \leftarrow ICP(P_j^i, P_j^0) \) to estimate translation
9:             \( \hat{x}_j^i \leftarrow \hat{x}_j^0 + \Delta x_j^i \)
10:             \( \nu_j^i \leftarrow \) store error \( (\hat{x}_j^i - \hat{x}_j^0) \)
11:     \( R_j \leftarrow \) reference covariance matrix \( cov(\nu_j) \)
12:     \( R_{j,\text{dist}} \leftarrow \) distance-based covariance matrix
13:     \( d_F^{j,\text{dist}} \leftarrow d_F(R_j, \tilde{R}_{j,\text{dist}}) \)
14:     \( d_H^{j,\text{dist}} \leftarrow d_H(R_j, \tilde{R}_{j,\text{dist}}) \)
15:     \( \tilde{R}_{j,\text{adaptive}} \leftarrow \) adaptive covariance matrix
16:     \( d_F^{j,\text{adaptive}} \leftarrow d_F(R_j, \tilde{R}_{j,\text{adaptive}}) \)
17:     \( d_H^{j,\text{adaptive}} \leftarrow d_H(R_j, \tilde{R}_{j,\text{adaptive}}) \)
18:     \( \tilde{d}_F^{\text{dist}} \leftarrow \text{mean}(d_F^{j,\text{dist}}) \)
19:     \( \tilde{d}_H^{\text{dist}} \leftarrow \text{mean}(d_H^{j,\text{dist}}) \)
20:     \( \tilde{d}_F^{\text{adaptive}} \leftarrow \text{mean}(d_F^{j,\text{adaptive}}) \)
21:     \( \tilde{d}_H^{\text{adaptive}} \leftarrow \text{mean}(d_H^{j,\text{adaptive}}) \)

IV. GPS-LiDAR INTEGRATION

In this section, we test the performance of our adaptive covariance model for real outdoor data. For real world experiments, it is challenging to generate reference covariance matrices. Thus, we apply a KF to fuse GPS and LiDAR, and compare the positioning errors to validate our adaptive covariance model.

A. Kalman Filter Details

Our state vector for the KF consists of the UAV’s global position and velocity: \( \mathbf{X} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T \). The prediction and the update steps of the filter can be represented as the following equations [35]:

Prediction Step:
\[
\mathbf{X}_{k|k-1} = \mathbf{F} \mathbf{X}_{k-1|k-1} + \mathbf{w}_k \\
\mathbf{P}_{k|k-1} = \mathbf{F} \mathbf{P}_{k-1|k-1} \mathbf{F}^T + \mathbf{Q}_k
\]

Update Step:
\[
\begin{align*}
\mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}^T (\mathbf{H} \mathbf{P}_{k|k-1} \mathbf{H}^T + \mathbf{R}_k)^{-1} \\
\mathbf{X}_{k|k} &= \mathbf{X}_{k|k-1} + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H} \mathbf{X}_{k|k-1}) \\
\mathbf{P}_{k|k} &= (I - \mathbf{K}_k \mathbf{H}) \mathbf{P}_{k|k-1},
\end{align*}
\]

where \( \mathbf{X} \) is the state vector; \( \mathbf{z} \) is the measurement; \( \mathbf{P} \) is the covariance of the state vector; \( \mathbf{F} \) is the state transition model; \( \mathbf{H} \) is the measurement model; \( \mathbf{w} \) is the process noise; \( \mathbf{Q} \) is the covariance of the process noise; \( \nu \) is the observation noise; \( \mathbf{R} \) is the covariance of the measurement noise; \( \mathbf{K} \) is the Kalman gain; \( k \) and \( k - 1 \) denote the current and previous time steps. For our state transition, we use the following model:

\[
\mathbf{F} = \begin{bmatrix}
1 & 0 & 0 & \Delta t & 0 & 0 \\
0 & 1 & 0 & 0 & \Delta t & 0 \\
0 & 0 & 1 & 0 & 0 & \Delta t \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix},
\]

where \( \Delta t \) is the duration for which we predict the states. We fix the process noise covariance as [36]:

\[
Q = 10 \begin{bmatrix}
\frac{\Delta t^3}{3} & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 \\
0 & \frac{\Delta t^3}{3} & 0 & 0 & \frac{\Delta t^2}{2} & 0 \\
\frac{\Delta t^2}{2} & 0 & 0 & \Delta t & 0 & 0 \\
0 & \frac{\Delta t^2}{2} & 0 & 0 & \Delta t & 0
\end{bmatrix}
\]

The GPS and LiDAR position measurements input to the filter are:

\[
\mathbf{z}_k^{GPS} = \tilde{\mathbf{x}}_k^{GPS} \\
\mathbf{z}_k^{LiDAR} = \Delta \mathbf{x}_k = \hat{\mathbf{x}}_k - \hat{\mathbf{x}}_{k-1|k-1},
\]

where \( \tilde{\mathbf{x}}_k^{GPS} \) is the GPS output of the UAV’s global position. \( \Delta \mathbf{x} \) is estimate of the translation between two consecutive point clouds obtained using ICP as shown in (1). The final LiDAR measurement \( \mathbf{z}_k^{LiDAR} \) is a sum of the ICP estimate in the global frame and the previous filter position estimate. For the GPS output we use a fixed covariance matrix \( \mathbf{R}^{GPS} = (1m)I \). For the LiDAR measurements, we compare the results between using our adaptive covariance model obtained from (18) versus using a distance-based covariance matrix. In Figure 7, we observe that a distance-based covariance matrix with weight \( \sigma_{\text{dist}} \approx 1.0 \) performs best according to our metrics. Thus, for our comparison we use a distance-based covariance matrix with \( \sigma_{\text{dist}} \) set to 1.0 in (20). The ICP translation estimate and our adaptive covariance matrix are measured in the LiDAR body frame. To transform these quantities to the global frame, we use an on-board attitude and heading reference system (AHRS) combined with an inertial measurement unit (IMU).

For the measurement model, we use the same constant matrix for both GPS and LiDAR measurements, since they both relate to the position states:

\[
\mathbf{H} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\]

B. Experimental Results

We use the iBQR UAV shown in Figure 8, designed and built by our research group for data collection. The UAV has
an arm length of 0.6m, and a payload capacity of 2kg. We use a Velodyne VLP-16 Puck Lite LiDAR [32], a u-blox LEA-6T GPS receiver connected to a Maxtena antenna, and an Xsens Mti-30 IMU. We use an AscTec MasterMind as the on-board computer, to log the data from all these sensors. We limit the range of the LiDAR to 15m, in order to evaluate our adaptive covariance model in certain challenging situations.

To initialize the filter, we assume that the UAV begins operation in an open-sky environment with accurate and reliable GPS signals to obtain the global origin of the system. We initialize the position states as zeros and the state covariance matrix $P_0$ as an identity matrix.

We implement the KF on an urban dataset collected on our campus of University of Illinois at Urbana-Champaign. For our trajectory, we begin at the South-West corner of the HydroSystems Building, head North and keep moving along the building till we reach our starting position again. The most common way of gathering ground truth for outdoor trajectories is the use of precise GNSS. Unfortunately, this data is only available in open outdoor environments and not for urban environments. Thus, we approximate our ground truth trajectory based on visual cues such as: walking on the sidewalk, walking in the middle of the alley and walking nearby parked cars.

Figure 9 shows a visualization of the position error covariance ellipsoid from our model for two point clouds along the trajectory. The point cloud in Figure 9(a) has a good distribution of features and our model estimates a low covariance in all directions. For the point cloud in 9(b) our model assigns a higher covariance parallel (North) to the wall and lower covariance perpendicular (East) to it. Here the KF will trust the LiDAR odometry output in the East direction since the corresponding covariance is about an order of magnitude lower than the GPS covariance.

Next, we check the GPS position and LiDAR odometry measurements versus the ground truth. For LiDAR odometry, we simply integrate the ICP output between consecutive point clouds. As shown in Figure 10, the LiDAR odometry trajectory accumulates drift over time and separates from ground truth. The lack of features during certain parts of the trajectory result in incorrect ICP estimation. The GPS measurements also contain significant errors due to the presence of nearby urban structures. The section of the trajectory to the North of the building shows certain GPS estimates inside the adjacent
building, whereas the section to the East shows the GPS estimates to be significantly away from the sidewalk.

Next, we implement the KF described in section IV-A. Figure 11 shows the output of the filter in three different scenarios: using a distance-based covariance matrix; using the covariance matrix from [37] referred to as Censi covariance; using our adaptive covariance matrix. By visual inspection we can observe that using our covariance model helps the filter estimate the UAV position more accurately compared to other models. The improvement is most prominent on the East and South sides of the building. Our adaptive covariance matrix correctly passes information to the filter regarding the ICP estimation errors, thus reducing the position errors perpendicular to the building wall. Table II shows the mean squared error (MSE) between the true and the estimated trajectories for the five different scenarios shown in Figures 10 and 11.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiDAR odometry only</td>
<td>9.9m</td>
</tr>
<tr>
<td>GPS position only</td>
<td>2.0m</td>
</tr>
<tr>
<td>GPS-LiDAR distance-based covariance</td>
<td>1.4m</td>
</tr>
<tr>
<td>GPS-LiDAR Censi covariance [37]</td>
<td>1.3m</td>
</tr>
<tr>
<td>GPS-LiDAR adaptive covariance</td>
<td>1.0m</td>
</tr>
</tbody>
</table>

TABLE II: Mean squared error of different trajectories. Our adaptive covariance matrix performs better than the distance-based covariance and Censi covariance [37].

V. CONCLUSION

This paper proposed a novel method for adaptively computing the LiDAR-based positioning error covariance matrix based on surrounding features. We explored performance of ICP in cases of varying availability of features to set up the motivation for our covariance matrix model. We described our algorithm to extract surface and edge feature points, and presented our model of adaptively estimating the position error covariance matrix based on these individual features. We presented the mathematical background behind combining these observations to obtain a final estimate for the LiDAR-based positioning error covariance matrix.

Next, we introduced our simulation environment where we generated multiple reference covariance matrices for three different scenes. We introduced the metrics we used for comparing covariance matrices, and evaluated the performance of our adaptive covariance matrix model versus a range of distance-based covariance matrices. Our model performed better than a distance-based model in capturing the reference covariance matrices. We observed large improvements in scenes with structured environments since our model is based on surface and edge features. For dense regions, we observe that our model performs slightly better than the distance-based model.

Finally, we tested our model for real outdoor data. We developed a basic KF to estimate the UAV’s global position by fusing position measurements from GPS and LiDAR. We compared the output from the filter using three different covariance matrices: our adaptive covariance model, Censi covariance model [37] and a distance-based covariance model. The trajectory output while using our model obtained a MSE of about 1.0m compared to about 1.3m obtained using Censi covariance and 1.4m using a distance-based covariance model.

VI. ACKNOWLEDGEMENT

The authors would like to sincerely thank Kalman Krishnakumar and his group at NASA Ames for supporting this work under the grant NNX17AC13G.

REFERENCES


Akshay Shetty received the B.Tech. degree in aerospace engineering from Indian Institute of Technology, Bombay, India in 2014. He received the M.S. degree in aerospace engineering from University of Illinois at Urbana-Champaign in 2017. He has worked as an Intern at NASA Ames Research Center, California during summer 2016 and summer 2017. He is currently pursuing the PhD degree at University of Illinois at Urbana-Champaign. His research interests include robotics and control.

Grace Xingxin Gao received the B.S. degree in mechanical engineering and the M.S. degree in electrical engineering from Tsinghua University, Beijing, China in 2001 and 2003. She received the PhD degree in electrical engineering from Stanford University in 2008. From 2008 to 2012, she was a research associate at Stanford University. Since 2012, she has been with University of Illinois at Urbana-Champaign, where she is presently an assistant professor in the Aerospace Engineering Department. Her research interests are systems, signals, control, and robotics.