GPS-LIDAR SENSOR FUSION
AIRED BY 3D CITY MODELS FOR UAVS

BY
AKSHAY SHETTY

THESIS
Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:
Assistant Professor Grace Xingxin Gao
ABSTRACT

Recently, there has been an increase in outdoor applications for small-scale Unmanned Aerial Vehicles (UAVs), such as 3D modelling, filming, surveillance, and search and rescue. To perform these tasks safely and reliably, a continuous and accurate estimate of the UAVs positions is needed. Global Positioning System (GPS) receivers are commonly used for this purpose. However, navigating in urban areas using only GPS is challenging, since satellite signals might be reflected or blocked by buildings, resulting in multipath errors or non-line-of-sight (NLOS) situations. In such cases, additional on-board sensors are desirable to improve global positioning of the UAV. Light Detection and Ranging (LiDAR), one such sensor, provides a real-time point cloud of its surroundings. In a dense urban environment, LiDAR is able to detect a large number of features of surrounding structures, such as buildings, as opposed to in an open-sky environment. This characteristic of LiDAR complements GPS, which is accurate in open-sky environments, but may suffer large errors in urban areas.

To fuse GPS and LiDAR measurements, Kalman Filtering and its variations are commonly used. However, it is important, yet challenging, to accurately characterize the error covariance of the sensor measurements.

In this thesis, we propose a GPS-LiDAR fusion technique with a novel method for efficiently modelling the error covariance in position measurements based on LiDAR point clouds. For GPS measurements, we eliminate NLOS satellites and model the covariance based on the measurement signal-to-noise ratio (SNR) values.

We use the LiDAR point clouds in two ways: to estimate incremental motion by matching consecutive point clouds; and, to estimate global pose by matching with a 3D city model. We aim to characterize the error covariance matrices in these two aspects as a function of the distribution of features in the LiDAR point cloud.
To estimate the incremental motion between two consecutive LiDAR point clouds, we use the Iterative Closest Point (ICP) algorithm. We perform simulations in different environments to showcase the dependence of ICP on features in the point cloud. While navigating in urban areas, we expect the LiDAR to detect structured objects, such as buildings, which are primarily composed of surfaces and edges. Thus, we develop an efficient way for modelling the error covariance in the estimated incremental position based on each surface and edge feature point in the point cloud. A surface point helps to estimate motion of the LiDAR perpendicular to the surface, while an edge point helps to estimate motion of the LiDAR perpendicular to the edge. We treat each feature point independently and combine their individual error covariance to obtain a total error covariance ellipsoid for the estimated incremental position.

For our 3D city model, we use elevation data of the State of Illinois available online and combine it with building information extracted from OpenStreetMap, a crowd-sourced mapping platform. We again use the ICP algorithm to match the LiDAR point cloud with our 3D city model, which provides us with an estimate of the UAV’s global pose. Additionally, we also use the 3D city model to determine and eliminate NLOS GPS satellites. We use remaining pseudorange measurements from the on-board GPS receiver and a stationary reference receiver to create a vector of double-difference measurements. We create a covariance matrix for the GPS double-difference measurement vector based on SNR of the individual pseudorange measurements.

Finally, all the above measurements and error covariance matrices are provided as an input to an Unscented Kalman Filter (UKF). The states of the filter include the globally referenced pose of the UAV. Before implementation, we perform an observability analysis for our filter. To validate our algorithm, we conduct UAV experiments in GPS-challenged urban environments on the University of Illinois at Urbana-Champaign campus. We observe that our model for the covariance ellipsoid from on-board LiDAR point clouds accurately represents the position errors and improves the filter output. We demonstrate a clear improvement in the UAV’s global pose estimates using the proposed sensor fusion technique.
ACKNOWLEDGMENTS

Foremost, I would like to express my gratitude to my adviser, Assistant Professor Grace Xingxin Gao for the continuous support of my study and research, for her patience, motivation and enthusiasm. I would also like to thank the members of our research group: Craig Babiarz, Derek Chen, Enyu Luo, Hsi-Ping Chu, Matt Peretic, Sri Ramya Bhamidipati and Shubhendra Chauhan for helping me at various stages of this thesis.

In addition, I would like to thank the Safe Autonomous Flight Environment (SAFE50) and the Unmanned Aircraft System (UAS) Traffic Management (UTM) teams at NASA’s Ames Research Center for funding my research, and for hosting me in summer 2016 to conduct research at Ames.

Finally, I would like to thank my parents Sudha and Prabhakar Shetty, my sister Pooja Shetty, and Beatriz Maldonado for their ever-lasting support and words of encouragement.
TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION ........................................... 1
  1.1 GPS Signals in Urban Areas .................................. 1
  1.2 Additional Sensors - LiDAR ................................. 2
  1.3 Related Work on Sensor Fusion .............................. 3
  1.4 Contribution of This Thesis ................................. 4
  1.5 Thesis Outline .................................................. 6

CHAPTER 2 LIDAR-BASED ODOMETRY ................................. 7
  2.1 Iterative Closest Point (ICP) Algorithm ................. 7
  2.2 ICP Odometry: Simulations ................................. 8
  2.3 ICP Implementation in Urban Areas ....................... 9
  2.4 Summary .......................................................... 13

CHAPTER 3 MATCHING LIDAR AND 3D CITY MODEL ............... 14
  3.1 Generating 3D City Model ................................. 14
  3.2 On-board LiDAR - 3D City Model ......................... 16
  3.3 Implementation and Experimental Results ................ 18
  3.4 Summary .......................................................... 19

CHAPTER 4 MODELING LIDAR POSITION ERROR COVARIA-
  NCE ................................................................. 21
  4.1 Point Cloud Feature Extraction ........................... 21
  4.2 Position Error Covariance for Individual Feature Points 22
  4.3 Combining Error Ellipsoids ................................ 26
  4.4 Summary .......................................................... 28

CHAPTER 5 GPS MEASUREMENT MODEL .............................. 29
  5.1 Model of GPS Measurements ............................... 29
  5.2 NLOS Satellites Elimination .............................. 31
  5.3 GPS Measurement Vector and Covariance ................ 32
  5.4 Summary .......................................................... 34
Emerging applications in unmanned aerial vehicles (UAVs) have led to forecasts [1, 2] of widespread use in the near future. Some of these applications such as 3-dimensional (3D) modelling, filming, surveying, search and rescue, and delivering packages, involve flying in urban environments. In these scenarios, autonomously navigating a UAV has certain advantages [3] such as optimizing flight paths and sensing and avoiding collisions. However, to enable such autonomous control, we need a continuous and reliable source for the UAVs’ positioning.

1.1 GPS Signals in Urban Areas

In most cases, Global Positioning System (GPS) is primarily relied on for positioning in outdoor environments. The position output from a GPS receiver depends on the quality of signals it receives from GPS satellites. However, in
an urban environment, GPS signals from the satellites are often blocked or reflected by surrounding structures [4–6], causing large errors in the position output. Figure 1.1 shows the types of errors affecting GPS signals in urban areas.

Different techniques [7] have been demonstrated to reduce the effect of multipath and NLOS signals on positioning errors. These techniques include using adaptive filters [8], using multiple antennas [9], and using multifrequency measurements [10]. Furthermore, there has been a considerable amount of research into reducing positioning errors with the aid of 3D city models. One of the methods includes choosing candidate positions, predicting the measurements at these positions using the 3D city model, and finally comparing the predicted and received measurements to improve the predicted position [11–13]. In addition, other existing techniques include shadow matching [14] and geometric modelling [15] of the NLOS signals.

1.2 Additional Sensors - LiDAR

In cases when GPS is unreliable, additional on-board sensors such as Light Detection and Ranging (LiDAR) are commonly used to obtain the navigation solution. For simplicity, we refer to the LiDAR sensor as just LiDAR in this thesis. An on-board LiDAR provides a real-time point cloud of the surroundings of the UAV. Thus, in a dense urban environment, LiDAR is able to detect a large number of features from surrounding structures, such as buildings. Figure 1.2 shows a sample point cloud generated by LiDAR in an urban environment.

Figure 1.2: Point cloud generated by on-board LiDAR in a dense urban environment.
Positioning based on LiDAR point clouds has been demonstrated primarily by applying different simultaneous localization and mapping (SLAM) algorithms [16]. In many cases, algorithms implement variants of Iterative Closest Point (ICP) [17] to register new point clouds [18–21]. Other methods not dependent on ICP [22, 23], include point cloud matching based on polar coordinates of points [24] and tracking based on features in a point cloud [25]. Versions of source code for the implementation of some LiDAR-based SLAM algorithms [26–29] are available online.

Furthermore, there has been analysis on the covariance of position estimates from algorithms based on LiDAR point clouds [30–32]. These methods have been demonstrated in simulations and practice, and include training kernels based on likelihood optimization [33–35], obtaining the covariance based on the Fisher Information matrix [36,37], and obtaining the covariance based on 2D scanned lines [38].

### 1.3 Related Work on Sensor Fusion

The characteristic of LiDAR to detect a large number of features in urban areas complements GPS, which is accurate in open-sky environments, but may contain large errors in urban areas. Different techniques to integrate LiDAR and GPS have been demonstrated to improve the navigation solution.

The most straightforward way to integrate these sensors is to loosely-couple them, i.e. to directly use the position output from the sensors. Prior work [39–42] with LiDAR equipped UAVs demonstrated such loosely-coupled architectures. However, in certain cases, the measurements made by the individual sensors might contain errors, or might be insufficient to estimate the position. In such scenarios, a tightly-coupled sensor fusion architecture tends to work better than a loosely-coupled one. Tightly-coupled architectures with GPS and a 2D laser scanner [43–45] have been demonstrated to provide an accurate navigation solution in challenging environments. Another application of a tightly-coupled structure, to eliminate NLOS satellites by detecting skyline from omnidirectional camera images [46, 47], has also shown an improvement in urban positioning.
1.4 Contribution of This Thesis

The main contribution of this thesis is a GPS-LiDAR fusion technique with a novel method for efficiently modelling the error covariance in position measurements derived from LiDAR point clouds. Figure 1.3 shows the different components involved in the sensor fusion.

1.4.1 Analyzed ICP odometry limitations

We introduce ICP, the algorithm that we use for point cloud matching in this thesis, and explain how we estimate the motion between two consecutive point clouds. In order to analyze prior work and the limitations of ICP, we perform simulations and analyze the effect of surrounding features on the error covariance of position estimates. We validate ICPs’ limitations on real world datasets by implementing an ICP mapping algorithm [21,28] in urban environments.

1.4.2 Analyzed LiDAR - 3D city model matching

We introduce Illinois Geospatial Data Clearinghouse and OpenStreetMap, which we use to build our globally referenced 3D city model. We then implement ICP to match the on-board LiDAR point cloud with the 3D city model. Next, we analyze the effect that initial positioning errors have on the global position output, depending on the surrounding features in the LiDAR point cloud.

1.4.3 Characterized error covariance model for LiDAR-based position estimates

Once we analyze how position estimates from LiDAR-based point clouds depend on the surrounding features, we proceed to build the error covariance model. First, we extract surface and edge feature points from the point cloud. We then model the position error covariance based on these individual feature points. Finally, we combine all the individual covariance matrices to model the overall position error covariance ellipsoid. We validate our model in urban environments.
1.4.4 Created GPS measurement model with NLOS satellite elimination

We use the pseudorange measurements from a stationary reference receiver and an on-board GPS receiver to obtain a vector of double-difference measurements. Using the double-difference measurements eliminates clock bias and atmospheric error terms, hence reducing the number of unknown variables. We use the global position estimate from the LiDAR - 3D city matching to construct LOS vectors to all the detected satellites. We then use our previously built 3D city model to detect NLOS satellites, and consequently refine the double-difference measurement vector. We create a covariance matrix for the GPS double-difference measurement vector based on SNR of the individual pseudorange measurements.

1.4.5 Proposed filter structure for GPS-LiDAR integration

Finally, we propose an Unscented Kalman Filter (UKF) structure to integrate all GPS and LiDAR measurements. We perform an observability test for the filter, based on Lie derivatives. Finally, we implement the filter on an urban dataset to show an improvement in the navigation solution.
1.5 Thesis Outline

Chapter 2 introduces the point cloud matching algorithm that we use for odometry based on consecutive point clouds. It discusses ICP algorithm, and performs simulations to analyze ICPs’ limitations. It then implements an ICP mapping method on urban datasets to verify the limitations.

Chapter 3 introduces the LiDAR - 3D city model matching algorithm. It first details the steps taken to build the 3D city model. It then analyzes the performance of using ICP for matching in urban environments.

Chapter 4 focuses on building the error covariance model for position estimates obtained from the LiDAR, as mentioned in Chapters 2 and 3. It models the covariance ellipsoid as a function of the distribution of surface and edge features in the point cloud.

Chapter 5 focuses on creating the GPS measurement vector and it’s covariance. It introduces the model for the received GPS pseudorange measurements, and the steps taken to create a vector of double-difference measurements.

Chapter 6 presents the Unscented Kalman Filter structure to integrate the measurements from GPS and LiDAR, described in the previous chapters. It then presents an observability analysis of the filter structure. Next, it elaborates details of the experimental setup and evaluates the filter performance on an urban dataset.

Chapter 7 concludes this thesis.
CHAPTER 2
LIDAR-BASED ODOMETRY

In this chapter, we analyze odometry estimates based on the on-board LiDAR. First, we introduce the Iterative Closest Point (ICP) algorithm used on the point clouds. Next, we perform simulations to examine some limitations of ICP in certain under-constrained situations. Finally, we evaluate the performance of ICP over some urban datasets.

2.1 Iterative Closest Point (ICP) Algorithm

ICP is commonly used for registering three-dimensional point clouds. It takes a reference point cloud $q$, an input point cloud $p$, and estimates the rotation matrix $R$ and the translation vector $T$ between the two point clouds. We represent the point clouds $p$ and $q$ as matrices with size $N \times 3$, where $N$ is the number of points in the point cloud, and the $i^{th}$ row of the matrix is the 3D coordinates of $i^{th}$ point $p_i$ and $q_i$, respectively.

There has been extensive literature introducing different variants of the algorithm [17]. These generally consist of the following steps:

- **Matching:** This step involves matching each point in the input point cloud $p_i$, to a point in the reference point cloud $q_i$. The most common method is to find the nearest neighbors of each point in the input point cloud. Different methods could be used to find the nearest neighbor, such as simple brute-force, Delaunay triangulation [48], or kDtrees [49]. The overall search time for the methods depends on the spatial distribution of the points. For our application, a kDtree performs best since the two point clouds are relatively close to each other [50].

- **Defining Error Metric:** After matching each point, the next step is to define the error metric for the point pairs. Among different metrics, such as point-to-point or point-to-plane, we choose point-to-point,
Figure 2.1: The input to ICP is a reference point cloud $q$ and an input point cloud $p$ as shown in (a). The algorithm calculates the rotation matrix $R$ and the translation vector $T$ such that the error metric $E$ is minimized. (b) shows the reference point cloud $q$ and the transformed input point cloud $R \cdot p + T$.

which is generally more robust to difficult geometry [17]. The total error between the two point clouds is defined as follows:

$$E = \sum_{i=1}^{N} \| R \cdot p_i + T - q_i \|,$$

where $N$ is the number of points in the input point cloud $p$; and $R$ and $T$ are the estimated rotation matrix and translation vector respectively.

- **Minimization:** The last step of the algorithm is the minimization of the error metric w.r.t. the rotation matrix $R$ and the translation vector $T$ between the two point clouds.

We use ICP to estimate the movement of the LiDAR between consecutive point clouds. The previous LiDAR point cloud is used as the reference $q$ and the current LiDAR point cloud is used as the input $p$. Figure 2.1 shows our implementation [51] of ICP to estimate the LiDAR odometry.

### 2.2 ICP Odometry: Simulations

The goal of the simulations is to analyze the limitations for ICP. We perform the simulations in 2D plane, but the analysis can be extended to 3D. We define the pose of the LiDAR at time $t$ as:

$$x_t = [x_t, y_t, \theta_t],$$

8
where $x_t$ and $y_t$ represent the position coordinates of the LiDAR; and $\theta_t$ represents the yaw.

For the simulations, we follow a similar method as described in [36,37]:

- Choose an initial pose of the LiDAR at time $t - 1$ as $x_{t-1}$. Generate a point cloud $p$ for the chosen environment. In order to simulate real point clouds, to each point $p_i$, add Gaussian noise to each direction.

- Choose a random new pose at time $t$ as $x_t$ in the vicinity of $x_{t-1}$. Generate point cloud $q$ with respect to the new pose. To each point $q_i$, add Gaussian noise to each direction. Perform ICP to match the point clouds $p$ and $q$, and obtain an estimate of the final pose, $\hat{x}_t$. Calculate the error in the estimated pose, $e = x_t - \hat{x}_t$. Repeat this step multiple times, with a different new pose, $x_t$ each time.

- Using the vector of error values obtained above, create the error covariance matrix $\text{cov}(e)$ [52].

We choose the simulation parameters similar to [36,37]. The LiDAR has 52 rays distributed in $360^\circ$, and the Gaussian noise in the point clouds has zero mean and 0.03 m standard deviation.

We choose three different environment cases for the simulation as shown in Figure 2.2. ICP fails to estimate the motion in under-constrained environments, such as while the LiDAR is moving through an infinite hallway. It is important to understand here that ICP correctly registers the two point clouds; however, the motion estimation is incorrect due to the structure of the point clouds.

Prior work of estimating the covariance using ICP is primarily based on the distribution of points in the point cloud [34–37]. However, the impact of certain environmental features for ICP is generally ignored. For example, a Fisher Information matrix [36,37] would deem movement along the hallway in both (b) and (c) to be unobservable. However, the presence of wall edges in (c) make the movement along the hallway, indeed, observable.

### 2.3 ICP Implementation in Urban Areas

We apply ICP to urban environments to evaluate the performance of the algorithm on real and large-scale datasets. Specifically, we implement the
Figure 2.2: Simulation of ICP in different environments. We move the LiDAR from \( x_{t-1} \) to \( x_t \), and generate noise corrupted point clouds \( p \) and \( q \) respectively. We perform ICP to find the transformation between \( p \) and \( q \), and thus estimate \( \hat{x}_t \) and obtain the error \( e \). This is repeated multiple times for different values of \( x_t \) to finally estimate the error covariance \( \text{cov}(e) \). The environment plays an important role in shaping the error covariance matrix. The error covariance matrix in (c) is smaller along the hallway than in (b) due to the presence of wall edges.
Figure 2.3: ICP mapping output generated in a region with adequate feature distribution on University of Illinois at Urbana-Champaign campus.

Figure 2.4: ICP mapping output generated in a region with poor feature distribution on University of Illinois at Urbana-Champaign campus.
Figure 2.5: ICP odometry estimate in an urban area. The LiDAR traverses around a building as shown in (a). However as seen in (b), ICP is unable to correctly estimate the path of the LiDAR through the alleyway. This is due to a poor distribution of features within the alleyway.

ICP mapping algorithm [21, 28]. For data collection, we use a Velodyne VLP-16 Puck Lite LiDAR [53].

Figures 2.3 and 2.4 show the mapping results at two different locations with adequate and poor environmental feature distributions. The region mapped in Figure 2.3, has an adequate distribution of features throughout the trajectory. Hence, the resulting 3D map consists of clearly distinguishable objects such as buildings and trees.

In contrast, the 3D map generated in Figure 2.4 contains points randomly scattered towards the end of the trajectory. The initial portion of the map consists an adequate distribution of features, hence generating a clear map.

Figure 2.6: The point cloud generated by the LiDAR in the alleyway. The point cloud is similar to the infinite hallway environment simulated in Figure 2.2 (b).
However, towards the middle of the trajectory as we pass through an alleyway between two buildings, the mapping algorithm fails and adds scattered points to the map.

To take a closer look at the error caused in Figure 2.4, we collected another dataset near the same alleyway, but with a reduced LiDAR range.

As shown in Figure 2.5, ICP is again unable to estimate the motion of the LiDAR through the alleyway. This is because the consecutive point clouds generated by the LiDAR throughout the alleyway look similar; hence, ICP estimates that the LiDAR has not moved significantly.

Figure 2.6 shows a point cloud generated by the LiDAR as it passes through the alleyway. The point cloud primarily contains surfaces on both sides, and is similar to the infinite hallway environment shown in Figure 2.2.

2.4 Summary

In this chapter, we introduced ICP, the algorithm that we use for estimating motion between two consecutive LiDAR point clouds. We performed simulations to analyse the limitations of ICP, and observed the impact of the environment on the estimated position error. In addition, we implemented a state-of-the-art ICP mapping algorithm on urban environments, and observed how the accuracy of ICP estimates and the quality of the generated 3D map depend on the distribution of features in the LiDAR point clouds.
CHAPTER 3

MATCHING LiDAR AND 3D CITY MODEL

In this chapter, we describe our method to estimate the global pose of the LiDAR with the aid of a 3D city model. First, we mention the sources we use to build the 3D city model. Next, we explain how we match the LiDAR point cloud to the 3D city model. Finally, we implement and analyze the results for different urban scenarios.

3.1 Generating 3D City Model

We generate our 3D city model using data from two sources: Illinois Geospatial Data Clearinghouse [54] and OpenStreetMap (OSM) [55, 56]. The Illinois Geospatial Data provides a point cloud, which primarily contains adequate details for the ground surface and the building rooftops, while OpenStreetMap provides building footprints and height information.

3.1.1 Illinois Geospatial Data Clearinghouse

The Illinois Geospatial Data Clearinghouse is supported by the Prairie Research Institute of University of Illinois at Urbana-Champaign. Figure 3.1 shows the available data for the State of Illinois. The point cloud data for Champaign County was last updated in April, 2008. It was collected by a fixed wing aircraft flying at an altitude of 1700 meters, equipped with a LiDAR system including a differential GPS unit and an inertial measurement system to provide superior global accuracy.

Since the data was collected from a relatively high altitude, the point cloud represents more of a top-view of the area. As a result, the sides of building structures contain only a few points. Figure 3.2 shows a section of the point cloud for Champaign County.
Figure 3.1: Geospatial data available at Illinois Geospatial Data Clearinghouse [54]. We work specifically with the dataset for Champaign County, highlighted in red.

Figure 3.2: Section of the point cloud for Champaign County dataset. The airborne-LiDAR detects mainly the rooftops and provides only a few points on the sides of the buildings.
3.1.2 OpenStreetMap

In order to complete the 3D city model, we need additional information for the sides of buildings. We use OpenStreetMap (OSM) to obtain this information. OSM is a freely available, crowd-sourced map of the world, which allows users to obtain information such as building footprints and heights. Figure 3.3 shows the OSM interface with building footprint for a small area in Champaign County. Once we extract the relevant building information, we proceed to incorporate the building sides. Figure 3.4 shows the point cloud for the same section of Champaign County dataset as shown in Figure 3.2, after incorporating the building sides.

3.2 On-board LiDAR - 3D City Model

Once we have the 3D city model ready, we focus on estimating the global pose of the LiDAR. In order to obtain a global pose estimate, we match the on-board LiDAR point cloud with the 3D city model using ICP described in

![OpenStreetMap interface](56) with detailed building footprint information that can be exported.
We implement the following steps in order to estimate the global pose:

- Use the position output from on-board GPS receiver as an initial guess. However, if the GPS receiver is unable to estimate the position due to the unavailability of GPS satellites, use the position estimate from the previous iteration as an initial guess. For orientation, use the estimate from the previous iteration. Thus, we obtain an initial pose guess $\hat{x}_0^L$.

- Project the on-board LiDAR point cloud $p_L$ to the same space as the 3D city model $q_{city}$ [57] using $\hat{x}_L^0$.

- Implement ICP, to obtain the rotation $R_L$ and translation $T_L$ between the two point clouds. Use this output to obtain an estimate for the global pose $\hat{x}_L$.

Figure 3.5 shows the results of implementation of the above method.
Figure 3.5: Global pose estimation with the aid of a 3D city model. (a) shows the initial position guess $\hat{x}_L^0$ (red) and the on-board LiDAR point cloud $p_L$ projected on the same space as the 3D city model $q_{city}$. (b) shows the updated global position $\hat{x}_L$ (green) after the ICP step. We observe an improvement in the global position, as the LiDAR point cloud matches with the 3D city model.

### 3.3 Implementation and Experimental Results

While navigating in urban areas, the GPS receiver position output used for the initial position guess $\hat{x}_L^0$ might contain large errors in certain directions. This might cause ICP to converge to a local minimum, depending on features in the point cloud $p_L$ generated by the on-board LiDAR. Thus, the goal of this section is to evaluate how our LiDAR - 3D city model matching algorithm

Figure 3.6: LiDAR-3D city model matching in urban area with adequate feature distribution. We begin with a grid of initial position guesses (red) around the true position (black). The position estimates after matching (blue) converge to the true position.
Figure 3.7: LiDAR-3D city model matching in urban area with poor feature distribution. We begin with a grid of initial position guesses (red) around the true position (black). The position estimates after matching (blue) are parallel to the building surface.

performs in urban environments with erroneous initial position guesses.

We begin by selecting a grid of initial position guesses, near the true position. We define a grid of points extending to 20 metres in each direction: North, South, East and West. We choose two different environments to evaluate the matching algorithm. Figure 3.6 shows an urban scenario with an adequate distribution of features. Thus, ICP is able to correctly match the two point clouds and provide an accurate position estimate after matching. In contrast, Figure 3.7 shows an urban scenario with a relatively poor distribution of features: just the one building surface close to the LiDAR. Thus, the on-board LiDAR point cloud looks similar for different initial position guesses parallel to the wall. Hence, ICP is unable to estimate the position accurately, in the direction parallel to the building surface.

3.4 Summary

In this chapter, we described our algorithm for estimating the LiDAR’s global pose with the aid of a 3D city model. We first introduced our sources to generate the 3D city model. Next, we described our LiDAR - 3D city model matching algorithm and evaluated its performance in certain urban environments. We observed how the distribution of features affected the final position estimates. Figure 3.8 summarizes the different sections of this chapter.
Figure 3.8: Summary of the on-board LiDAR - 3D city model matching algorithm presented in this chapter.
CHAPTER 4

MODELING LIDAR POSITION ERROR COVARIANCE

In this chapter, we build the error covariance matrix for position estimates from the on-board LiDAR point clouds, as described in Chapters 2 and 3. We model this error covariance matrix based on the distribution of features in the point cloud. First, we extract surface and edge feature points from the point cloud. Next, we describe the model for position error covariance due to each individual feature point. Finally, we combine these individual error covariance matrices to obtain a total position error covariance ellipsoid.

4.1 Point Cloud Feature Extraction

In urban environments, we typically observe structured objects such as buildings. Hence, we focus primarily on surface and edge features in the point cloud. We extract these feature points based on the curvature at each point, as described in [25]. The algorithm can be summarized in the following steps:

- For each point in the point cloud, the curvature \( c_i \) is defined as follows [25]:

\[
c_i = \frac{1}{\|X_i\|} \cdot \left\| \sum_{j \in S, j \neq i} (X_i - X_j) \right\|, \quad (4.1)
\]

where:

\( X_i \) : Coordinates of \( i^{th} \) point relative to the LiDAR

\( S \) : Small neighborhood of points near \( i \).

- Discard stray points and points on occluded objects. These are detected based on the range and distance between consecutive points.
Figure 4.1: Surface feature points (red) and edge feature points (red) on point clouds (green) generated by on-board LiDAR.

- Divide the point cloud into 6 equal sections. For each section, sort the curvature values. Points with the lowest curvature values are classified as surface points, and points with the highest curvature values are classified as edge points.

- If a point is classified as either surface or an edge point, its neighboring points are discarded. This is done in order to prevent multiple detections of the same features.

Figure 4.1 shows the above feature extraction algorithm implemented on two different point clouds generated by the on-board LiDAR.

4.2 Position Error Covariance for Individual Feature Points

Once we extract the surface and edge feature points from the point cloud, we proceed to model the position error covariance based on each individual feature point.

4.2.1 Surface Feature Point

For each surface feature point \( j \), we implement the following steps to model the position error covariance:

- First, we compute the normal \( \hat{n} \) at the surface feature point \( j \). We use the \texttt{pcnormals} function from the Computer Vision System Toolbox in
MATLAB. We use 9 of the neighboring points to fit a plane needed to estimate the normal $\hat{u}^j$.

- In order to create a 3-dimensional error ellipsoid for the surface feature point, we first create an orthonormal basis with the corresponding normal $\hat{u}^j$. We select a vector $\vec{n}_u^j$ that is perpendicular to $\hat{u}^j$ as follows:

$$\vec{n}_u^j = \begin{bmatrix} 0 & -\hat{u}_3^j & \hat{u}_2^j \end{bmatrix}$$ (4.2)

Then we select $\vec{m}_u^j$ as a cross product between $\vec{n}_u^j$ and $\hat{u}^j$ to create an orthonormal basis:

$$\vec{m}_u^j = \vec{n}_u^j \times \hat{u}^j$$ (4.3)

We normalize $\vec{n}_u^j$ and $\vec{m}_u^j$ to obtain the following basis: $\{\hat{u}^j, \vec{n}_u^j, \vec{m}_u^j\}$.

$$\hat{n}_u^j = \frac{\vec{n}_u^j}{\|\vec{n}_u^j\|} \quad \hat{m}_u^j = \frac{\vec{m}_u^j}{\|\vec{m}_u^j\|}$$ (4.4)

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

$$V_u^j = \begin{bmatrix} \hat{u}^j & \hat{n}_u^j & \hat{m}_u^j \end{bmatrix}$$ (4.5)

We model the error covariance ellipsoid with the hypothesis 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 of the density of points. Hence, we use the following eigenvalues corresponding to the eigenvectors in (4.5):

$$L_u^j = d_u^j \begin{bmatrix} a_u & \cdot & \cdot \\ \cdot & b_u & \cdot \\ \cdot & \cdot & b_u \end{bmatrix}$$ (4.6)

where:

$a_u, b_u$ : Constants for all surface points such that: $a_u \ll b_u$

d$u^j$ : Distance of the $j^{th}$ surface point from the LiDAR
The values for constants $a_u$ and $b_u$ are tuned during implementation.

- Finally, using the eigenvectors $V^j_u$ and the eigenvalues $L^j_u$, we construct the position error covariance matrix for the $j^{th}$ surface point as follows:

$$R^j_u = V^j_u \cdot L^j_u \cdot V^j_u^{-1}$$  \hspace{1cm} (4.7)

Figure 4.2 shows a sample error ellipsoid generated by (4.7).

### 4.2.2 Edge Feature Point

For each edge feature point $j$, we implement the following steps to model the position error covariance:

- First, we find the orientation of the edge. We find 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 $\hat{e}^j$.

- Once we obtain the edge vectors, we follow similar steps as we did for each surface point. We first create an orthonormal basis with $\hat{e}^j$:

$$\hat{\mathbf{n}}^j_e = \begin{bmatrix} 0 & -\hat{e}_3^j & \hat{e}_2^j \end{bmatrix}$$  \hspace{1cm} (4.8)

$$\hat{\mathbf{m}}^j_e = \hat{n}^j_e \times \hat{e}^j$$  \hspace{1cm} (4.9)

After normalizing $\hat{n}^j_e$ and $\hat{m}^j_e$ as done in (4.4), we obtain the required basis $\{\hat{e}^j, \hat{n}^j_e, \hat{m}^j_e\}$. 

24
Next, we proceed to construct the position error covariance matrix. Similar to the case of a surface feature point, we use the basis as the eigenvectors:

\[
V_e^j = \begin{bmatrix}
\hat{e}^j & \hat{n}_e^j & \hat{m}_e^j
\end{bmatrix}
\] (4.10)

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 assume that edge points closer to the LiDAR are more reliable than those further away, because of the density of points. Hence, we use the following eigenvalues corresponding to the eigenvectors in (4.10):

\[
L_e^j = d_e^j \begin{bmatrix}
a_e & \cdot & \cdot \\
\cdot & b_e & \cdot \\
\cdot & \cdot & b_e
\end{bmatrix},
\] (4.11)

where:

\[
a_e, b_e : \text{Constants for all edge points such that: } a_e \gg b_e
\]

\[
d_e^j : \text{Distance of the } j^{th} \text{ edge point from the LiDAR}
\]

The values for constants \( a_e \) and \( b_e \) are tuned during implementation.

Finally, using the eigenvectors \( V_e^j \) and the eigenvalues \( L_e^j \), we construct

Figure 4.3: Position error covariance ellipsoid \( R_e^j \) contributed by the \( j^{th} \) surface point. \( \hat{e}^j \) represents the corresponding edge vector; and \( \hat{n}_e^j \) and \( \hat{m}_e^j \) complete orthonormal basis.
the position error covariance matrix for the $j^{th}$ edge point as follows:

$$R_j^e = V_j^e \cdot L_j^e \cdot V_j^e^{-1}$$  \hspace{1cm} (4.12)$$

Figure 4.3 shows a sample error ellipsoid generated by (4.12).

## 4.3 Combining Error Ellipsoids

In this section, we focus on obtaining the overall position error covariance ellipsoid using the models we described in the previous section. We first introduce some mathematical background required for combining multiple error ellipsoids. Then, we proceed to apply the result for our case by combining the error ellipsoids for each surface and edge feature point. We make the following assumptions for our system:

- The position of the LiDAR can be represented by a Gaussian random vector.

- The error ellipsoid from each surface normal and edge is from a statistically independent observation.

### 4.3.1 Mathematical background:

For a Gaussian random vector $X$, the equation of an ellipsoid [58] [59] for an observation $O_i$ can be represented as:

$$(\hat{X} - X)^T \cdot R_i^{-1} \cdot (\hat{X} - X) = C,$$  \hspace{1cm} (4.13)$$

where, $\hat{X}$ is the true value and $R$ is the covariance of the random vector. $C$ represents the size of the ellipsoid that encloses the observation. For a set of $n$ observations $(O_1, O_2, ..., O_n)$, the probability density function of $X$ can be expressed as:

$$p(X|O_1, O_2, ..., O_n) = \prod_{i=1}^{n} p(X|O_i)$$

$$= \frac{1}{\sqrt{(2\pi)^n |R|}} \prod_{i=1}^{n} \exp \left( -\frac{1}{2} (X - O_i)^T \cdot R_i^{-1} \cdot (X - O_i) \right)$$
\[
= \frac{1}{\sqrt{(2\pi)^n |R|}} \exp \left( -\frac{1}{2} \sum_{i=1}^{n} (X - O_i)^T R_i^{-1} (X - O_i) \right),
\]

(4.14)

where, \( |R| \) represents the determinant of the combined covariance matrix \( R \). Thus the optimal estimate for \( X \) is obtained by minimizing the argument of the exponential in the above equation:

\[
\hat{X} = \left( \sum_{i=1}^{n} R_i^{-1} \right)^{-1} \sum_{i=1}^{n} R_i^{-1} O_i.
\]

(4.15)

And the combined covariance matrix is:

\[
R = \left( \sum_{i=1}^{n} R_i^{-1} \right)^{-1}.
\]

(4.16)

### 4.3.2 LiDAR Combined Covariance

Using the result from (4.16), we obtain the overall error covariance for position output from LiDAR odometry as follows:

\[
R_L = \left( \sum_{j=1}^{n_u} R_u^{-1} + \sum_{j=1}^{n_e} R_e^{-1} \right)^{-1},
\]

(4.17)

Figure 4.4: Overall position error ellipsoids \( R_L \), for two point clouds generated by the on-board LiDAR in an urban environment.
where, $\mathbf{R}_j^u$ and $\mathbf{R}_j^e$ are the covariance matrices from individual surface (4.7) and edge (4.12) feature points respectively. $n_u$ and $n_e$ are the number of surface and the number of edge feature points in the point cloud.

Figure 4.4 shows the resulting position error covariance ellipsoids $\mathbf{R}_L$ for two point clouds in urban environments. In the horizontal plane, the covariance ellipsoid is larger in the direction parallel to the building sides. In the vertical direction, the size of the covariance ellipsoid remains constrained due to points detected on the ground.

### 4.4 Summary

In this chapter, we presented our algorithm to estimate the error covariance ellipsoid for position estimates from on-board LiDAR point clouds. We extracted surface and edge feature points from the point cloud and modelled their individual position error covariance ellipsoids. Finally, we combined the effect of all the feature points to obtain the overall position error covariance ellipsoid. Figure 4.5 summarizes the steps explained in this chapter.
CHAPTER 5

GPS MEASUREMENT MODEL

In this chapter, we build our GPS measurement vector and its covariance. First, we describe the model for the received GPS pseudorange measurements and then proceed to obtain the double difference measurements. Next, we look at how we use the 3D city model to aid in detecting non-line-of-sight (NLOS) satellites. Finally, we construct the measurement vector and the corresponding error covariance matrix.

5.1 Model of GPS Measurements

In this thesis, we use the GPS pseudorange measurements to aid global positioning. The pseudorange measurements between a GPS receiver $u$ and the $k^{th}$ satellite can be modelled as [60]:

$$\rho_u^k = r_u^k + c(\delta t_u - \delta t^k) + I_u^k + T_u^k + \epsilon_u^k,$$

(5.1)

where:

- $r_u^k$: True range between receiver $u$ and $k^{th}$ satellite
- $c$: Speed of light
- $\delta t_u$: Clock bias for receiver $u$
- $\delta t^k$: Clock bias for $k^{th}$ satellite
- $I_u^k$: Ionospheric error
- $T_u^k$: Tropospheric error
- $\epsilon_u^k$: Noise in pseudorange measurement.

In order to eliminate certain error terms, we use double-difference pseudorange measurements. Double-difference measurements are calculated by differencing the pseudorange measurements between two receivers and be-
Figure 5.1: Geometry of single-difference measurements. Here \( \mathbf{x}_u \) and \( \mathbf{x}_r \) are the positions of the user and reference receivers in the Earth Centered Earth Fixed (ECEF) frame [60]. The baseline between these receivers \( \mathbf{x}_{ur} \), is assumed to be significantly smaller than the ranges to the satellites \( r_u^k \) and \( r_r^k \).

Between two satellites. First, we begin with the single difference measurements between two receivers \( u \) and \( r \) which can be expressed as:

\[
\rho_{ur}^k = \rho_u^k - \rho_r^k \\
= (r_u^k - r_r^k) + (I_{\rho_u}^k - I_{\rho_r}^k) + (T_{\rho_u}^k - T_{\rho_r}^k) + c(\delta t_u - \delta t_r) + (\epsilon_{\rho_u}^k - \epsilon_{\rho_r}^k) \\
= r_{ur}^k + I_{\rho_{ur}}^k + T_{\rho_{ur}}^k + c\delta t_{ur} + \epsilon_{\rho_{ur}}^k \tag{5.2}
\]

For short baselines between the two receivers, the ionospheric error \( I_{\rho_{ur}}^k \) and tropospheric error \( T_{\rho_{ur}}^k \) can be assumed to be negligible. Thus, (5.2) can be approximated as:

\[
\rho_{ur}^k \approx r_{ur}^k + c\delta t_{ur} + \epsilon_{\rho_{ur}}^k \tag{5.3}
\]

Furthermore, the baseline between the two receivers can be assumed to be significantly smaller than the distance between the receivers and the satellites. Thus, the single-difference range term \( r_{ur}^k \) in (5.3) can be approximated as:

\[
r_{ur}^k = r_u^k - r_r^k \approx -\mathbf{1}_v^k \cdot \mathbf{x}_{ur}, \tag{5.4}
\]
where, $\mathbf{x}_{ur}$ is the baseline between the two receivers; and $\mathbf{1}_r^k$ is a unit vector from the receiver $r$ to the satellite $k$ as shown in Figure 5.1.

Next, in order to eliminate the clock bias terms, we difference the pseudorange measurements between two satellites. Using (5.3) the double difference pseudorange measurements between two GPS receivers $u$ and $r$, and between two satellites $k$ and $l$ can be represented as:

\[
\rho_{ur}^{k-l} = \rho_{ur}^k - \rho_{ur}^l \\
\approx (r_{ur}^k + c\delta t_{ur} + \epsilon_{\rho_{ur}}^k) - (r_{ur}^l + c\delta t_{ur} + \epsilon_{\rho_{ur}}^l) \\
= (r_{ur}^k - r_{ur}^l) + \epsilon_{\rho_{ur}}^{k-l} \tag{5.5}
\]

Using (5.4), the above expression for the double difference pseudorange measurement can be expressed as follows:

\[
\rho_{ur}^{k-l} \approx -(\mathbf{1}_r^k - \mathbf{1}_r^l) \cdot \mathbf{x}_{ur} + \epsilon_{\rho_{ur}}^{k-l} \tag{5.6}
\]

Thus, we have the double difference pseudorange measurements as a function of just the satellite geometry and position of the GPS receivers.

## 5.2 NLOS Satellites Elimination

Before proceeding to create the vector of GPS double difference pseudorange measurements, we check if any of the satellites detected by the receiver are non-line-of-sight signals. We use the 3D city model described in section 3.1 to detect the NLOS satellites. We implement the following steps:

- Use the position output generated by the LiDAR-3D city model matching, as described in Chapter 3, to locate the receiver on the 3D city model.
- Draw LOS vectors from the receiver to every satellite detected by the receiver.
- Detect and eliminate satellites whose corresponding LOS vectors intersect the 3D city model.

Fig. 5.2 shows the implementation of the algorithm in an urban scenario.
5.3 GPS Measurement Vector and Covariance

After eliminating the NLOS satellites, we proceed to create the GPS measurement vector and its covariance with the remaining satellites. Furthermore, we only select satellites that are visible to both the user and the reference receivers, in order to be able to difference the measurements. First, we create a vector of single-difference pseudorange measurements between the receivers:

\[
\rho_{ur}^{\text{SD}} = \begin{bmatrix}
\rho_{u1}^r \\
\rho_{u2}^r \\
\vdots \\
\rho_{uK}^r \\
\rho_{r1}^r \\
\rho_{r2}^r \\
\vdots \\
\rho_{rK}^r
\end{bmatrix} = A_{\text{SD}} \cdot \begin{bmatrix}
\rho_{1u} \\
\rho_{2u} \\
\vdots \\
\rho_{Ku} \\
\rho_{1r} \\
\rho_{2r} \\
\vdots \\
\rho_{Kr}
\end{bmatrix}, \quad (5.7)
\]
where:

\[
A_{SD} = \begin{bmatrix}
1 & -1 & & & \\
1 & -1 & & & \\
& \ddots & \ddots & & \vdots \\
& & 1 & -1 & \\
\end{bmatrix}.
\]

We arrange these measurements such that the \(K^{th}\) satellite has the highest elevation with respect to the user receiver. Next, we create a vector of the double-difference pseudorange measurements between the two receivers and between the satellites:

\[
\rho_{\text{DD}ur} = \begin{bmatrix}
\rho_{1 \cdot K}^{\text{ur}} \\
\rho_{2 \cdot K}^{\text{ur}} \\
\vdots \\
\rho_{(K-1) \cdot K}^{\text{ur}} \\
\end{bmatrix} = A_{\text{DD}} \cdot \rho_{\text{SD}ur},
\]

(5.8)

where:

\[
A_{\text{DD}} = \begin{bmatrix}
1 & -1 & & & \\
1 & -1 & & & \\
& \ddots & \ddots & & \vdots \\
& & 1 & -1 & \\
\end{bmatrix}.
\]

Once we have the vector of GPS measurements, we proceed to model its covariance. We begin with the covariance of the individual pseudorange measurements made by the receivers. We assume that the pseudorange measurements are independent, and that the variance for each measurement is a function of the corresponding signal-to-noise ratio \((C/N_0)^k_u\) in dB, where \(k\) refers to the \(k^{th}\) satellites and \(u\) refers to the user receiver [61]. We use the following covariance matrices for the user and the reference receivers respectively:

\[
R_{\rho_{\text{ua}}} = \begin{bmatrix}
10^{-\frac{(C/N_0)^1_u}{10}} & & & \\
& \ddots & & \\
& & \ddots & \\
& & & 10^{-\frac{(C/N_0)^K_u}{10}} \\
\end{bmatrix}.
\]

(5.9)
\[
R_{\rho r} = \begin{bmatrix}
10^{-\frac{(C/N_0)^2}{10}} & 10^{-\frac{(C/N_0)^2}{10}} & \cdots & 10^{-\frac{(C/N_0)^K}{10}} \\
\end{bmatrix}.
\] (5.10)

As seen in (5.7) and (5.8), the single-difference and double-difference pseudorange measurements are linear functions of the individual pseudorange measurements. Thus, we propagate the covariance from the individual measurements to the double-difference measurements as follows:

\[
R_{\rho SDur} = A_{SD} \cdot \begin{bmatrix}
R_{\rho u} & R_{\rho r}
\end{bmatrix} \cdot A_{SD}^T
\] (5.11)

\[
R_{\rho DDur} = A_{DD} \cdot R_{\rho SDur} \cdot A_{DD}^T
\] (5.12)

### 5.4 Summary

In this chapter, we described the steps we take to obtain the double-difference pseudorange measurement vector and its covariance. We explained the GPS pseudorange measurement model and how to obtain double-difference measurements from it. Then, we described how we use the 3D city model in order to eliminate NLOS satellites. Finally, we used the remaining satellites to create the double-difference measurement vector and its covariance matrix. Figure 5.3 summarizes the steps implemented in this chapter.
Figure 5.3: Summary of GPS measurement model and NLOS satellite elimination described in this chapter.
CHAPTER 6

GPS-LIDAR INTEGRATION

In this chapter, we describe our algorithm for integrating the GPS and LiDAR measurements obtained in the previous chapters. First, we explain the structure of our UKF, including the state vector, the prediction model and the relation between the measurements and the state vector. Next, we perform a non-linear observability analysis of the filter, under the availability of different measurements. Then, we list the equations for the UKF. Finally, we implement the filter in urban environments.

6.1 Kalman Filter Structure

6.1.1 State vector and prediction model

For our application, the primary goal of implementing a Kalman filter is to estimate the pose of the UAV. In addition to using a LiDAR and a GPS receiver mentioned in the previous chapters, we also use an inertial measurement unit (IMU) on-board the UAV. Thus, our state vector consists of the following states:

\[
x^T = [p_g^u, v_g^u, q_g^u, b_\omega^T, b_a^T, q_i^T]
\]

(6.1)

where:

- \(p_g^u\): Position of the UAV in the local GPS frame
- \(v_g^u\): Velocity of the UAV in the local GPS frame
- \(q_g^u\): Orientation of the UAV in the local GPS frame
- \(b_\omega, b_a\): Bias in the IMU gyroscopes and accelerometers
- \(q_i^T\): Orientation offset between the local GPS frame and the IMU frame

Figure 6.1 shows the different frames of reference used in the filter.
Figure 6.1: Frames of reference. The local GPS frame (blue) refers to the local North-East-Down (NED) frame centered at position where UAV begins operation. The IMU frame (red) is slightly offset the local GPS frame due to biases in the magnetometers. The UAV frame (green) is fixed to the body of the UAV.

We use a constant velocity model for the prediction step of the filter. Additionally, we include the angular velocity $\omega_m$, and acceleration $a_m$ measurements from the IMU in the prediction step [62], instead of the relatively expensive correction step. Thus, the prediction step can be written as:

$$\dot{x} = f_0(x) + f_1(x)u_1 + f_2(x)u_2,$$  \hspace{1cm} (6.3)

with $\dot{x}$, $f_0$, $f_1$, $f_2$, $u_1$ and $u_2$ as marked in (6.2).
6.1.2 Measurement model and covariances

For the correction step, we use pose information from the LiDAR, orientation information from the IMU and position information from the GPS receiver. From the LiDAR we use pose information from two sources: one from the LiDAR odometry as described in Chapter 2, and the other from the LiDAR - 3D city model matching as described in Chapter 3. The following equations relate the LiDAR odometry measurements of position $h_1$ and orientation $h_2$, to the state variables:

$$h_1 = R_{q^u_g}(p^u_g - p^u_g)$$  \hspace{1cm} (6.4)

$$h_2 = R_{q^u_g}R^T_{q^u_g},$$  \hspace{1cm} (6.5)

where, $p^u_g$ and $q^u_g$ refer to the previous filter estimate of position and orientation of the UAV in the local GPS frame. The LiDAR - 3D city model matching measurements of position $h_3$ and orientation $h_4$, directly relate to the state variables:

$$h_3 = p^u_g$$  \hspace{1cm} (6.6)

$$h_4 = q^u_g.$$  \hspace{1cm} (6.7)

The orientation measurement from the IMU is with respect to frame that is offset with respect to the local GPS frame. Thus, it relates to the state variables as follows:

$$h_5 = q^i_g \otimes q^u_g.$$  \hspace{1cm} (6.8)

For measurements from the GPS receiver, we use the double-difference measurements $\rho^{DD}_{ur}$, we obtained in section 5.3. These measurements can be related to the state variables as follows:

$$h_6 = G_1 \cdot ((p^u_g)_{ECEF} - p_{ref})$$

$$h_7 = G_2 \cdot ((p^u_g)_{ECEF} - p_{ref})$$

\hspace{1cm} \vdots

$$h_{(K-1)+5} = G_{K-1} \cdot ((p^u_g)_{ECEF} - p_{ref})$$  \hspace{1cm} (6.9)

Here, $K$ is the number of satellites, $(p^u_g)_{ECEF}$ is the position of the UAV in the ECEF frame, and $p_{ref}$ is the position of the reference receiver in the ECEF frame. $G$ is a function of the unit vectors from the reference receiver.
to the satellites being used in the double-difference calculation, as shown in (5.6).

The measurements $h_1$ and $h_2$ are in the LiDAR frame. Thus, for the covariance of $h_1$, we use $R_L$ that we derived in Chapter 4. The measurements $h_3$ and $h_4$ are from LiDAR - 3D city model matching, which are in the local GPS frame. Thus, for covariance of $h_3$, we rotate $R_L$ to the local GPS frame as follows [64]:

$$R_{GPS}^L = R_T(q_{ug}) \cdot R_L \cdot R(q_{ug})$$

(6.10)

For the orientation measurements $h_2$, $h_4$ and $h_5$, we use a fixed diagonal matrix. For the GPS double-difference measurements in (6.9), we use the covariance matrix $R_{\rho_{DDUR}}$, obtained in (5.12).

6.2 Non-linear Observability Analysis of Filter

Once we have the filter structure defined, prior to implementation, it is important to ensure that all the state variables are observable. To check the observability of our system, we refer to prior work that use Lie derivatives [62, 63, 65]. We check the observability in the presence of different sets of measurements.

6.2.1 Lie derivatives

The zeroth order Lie derivative of a function $h$, is calculated as:

$$\mathcal{L}^0 h(x) = h(x)$$

(6.11)

Further derivatives of $h$, with respect to a function $f$, is calculated recursively as:

$$\mathcal{L}^1 f h(x) = \frac{\partial h(x)}{\partial x} f(x) = \nabla h(x) f(x)$$

(6.12)

Thus, using (6.11) on the measurements described in the previous section, we obtain the following zeroth-order Lie derivatives for our filter:

$$\nabla \mathcal{L}^0 h_1 = \begin{bmatrix} \mathbf{R}(q_{ug}) & 0 & 0 & 0 & 0 \end{bmatrix}$$

(6.13)

$$\nabla \mathcal{L}^0 h_2 = \begin{bmatrix} 0 & 0 & \mathbf{a}^{[2,3]}_{3 \times 4} & 0 & 0 \end{bmatrix}$$

(6.14)
\[ \nabla L^0 h_3 = \begin{bmatrix} I_{3 \times 3} & 0 & 0 & 0 & 0 \end{bmatrix} \] (6.15)

\[ \nabla L^0 h_4 = \begin{bmatrix} 0 & 0 & I_{4 \times 4} & 0 & 0 \end{bmatrix} \] (6.16)

\[ \nabla L^0 h_5 = \begin{bmatrix} 0 & 0 & a_{4 \times 4}^{[5,3]} & 0 & 0 & a_{4 \times 4}^{[5,6]} \end{bmatrix} \] (6.17)

\[ \nabla L^0 h_6 = \begin{bmatrix} a_{1 \times 3}^{[6,1]} & 0 & 0 & 0 & 0 \end{bmatrix} \] (6.18)

\[ \nabla L^0 h_7 = \begin{bmatrix} a_{1 \times 3}^{[7,1]} & 0 & 0 & 0 & 0 \end{bmatrix} \] (6.19)

\[ \nabla L^0 h_8 = \begin{bmatrix} a_{1 \times 3}^{[8,1]} & 0 & 0 & 0 & 0 \end{bmatrix} \] (6.20)

Note that, the matrices \( a \) are not required for our analysis, hence we do not expand them fully. Furthermore, we limit the number of GPS measurements to 3 (which is equivalent to 4 GPS satellites since we are double-differencing the pseudorane measurements), as additional measurements are redundant for our observability analysis. Next, using (6.12), we obtain the first-order Lie derivative of \( h_1 \), with respect to \( f_0 \):

\[ \nabla L^1 f_0 h_1 = \begin{bmatrix} 0 & R(q_u) & 0 & 0 & 0 \end{bmatrix} \] (6.21)

\( h_3, h_6, h_7 \) and \( h_8 \), have a zeroth-order Lie derivative similar to \( h_1 \), such that it only affects the position states. Hence, the first-order Lie derivatives for these measurements are similar to (6.21). The first-order Lie derivative of \( h_2 \), with respect to \( f_0 \) is:

\[ \nabla L^1 f_0 h_2 = \begin{bmatrix} 0 & 0 & a_{3 \times 4}^{[10,3]} & a_{3 \times 3}^{[10,4]} & 0 & 0 \end{bmatrix} \] (6.22)

The first-order Lie derivatives of \( h_4 \) and \( h_5 \), are similar to (6.22). Finally, the second-order Lie derivative of \( h_1 \), with respect to \( f_0 \) is:

\[ \nabla L^2 f_0 f_0 h_1 = \begin{bmatrix} 0 & 0 & a_{3 \times 4}^{[11,3]} & 0 & a_{3 \times 3}^{[11,5]} & 0 \end{bmatrix} \] (6.23)

### 6.2.2 Observability matrix rank analysis

We proceed to create the observability matrix, whose rows are the gradients of the Lie derivatives we obtained. In order to prove that our system is observable, it is sufficient to show that the observability matrix has full column rank [62, 63]. Thus, in the presence of all the measurement sources,
our observability matrix looks as follows:

\[
\mathcal{O} = \begin{bmatrix}
\nabla L^0 h_1 \\
\nabla L^0 h_2 \\
\nabla L^0 h_3 \\
\nabla L^0 h_4 \\
\nabla L^0 h_5 \\
\nabla L^0 h_6 \\
\nabla L^0 h_7 \\
\nabla L^0 h_8 \\
\nabla L^1 f_0 h_1 \\
\nabla L^1 f_0 h_2 \\
\nabla L^2 f_0 f_0 h_1 
\end{bmatrix} = \begin{bmatrix}
R(\tau_u) & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & a_{[2,3]}^{3\times4} & 0 & 0 & 0 \\
I_{3\times3} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & I_{4\times4} & 0 & 0 & 0 \\
0 & 0 & a_{[5,3]}^{4\times4} & 0 & 0 & a_{[5,6]}^{4\times4} \\
a_{[6,1]}^{1\times3} & 0 & 0 & 0 & 0 & 0 \\
a_{[7,1]}^{1\times3} & 0 & 0 & 0 & 0 & 0 \\
a_{[8,1]}^{1\times3} & 0 & 0 & 0 & 0 & 0 \\
0 & R(\tau_u) & 0 & 0 & 0 & 0 \\
0 & 0 & a_{[10,3]}^{3\times4} & a_{[10,4]}^{3\times3} & 0 & 0 \\
0 & 0 & a_{[11,3]}^{3\times4} & 0 & a_{[11,5]}^{3\times3} & 0 
\end{bmatrix}
\]

(6.24)

In the above matrix, we can see that \(\nabla L^0 h_1, \nabla L^0 h_3, \nabla L^0 h_6, \nabla L^0 h_7, \nabla L^0 h_8\) together, account for the observability of the position states \(p_u\). The observability of the velocity \(v_u\), is accounted for by \(\nabla L^1 f_0 h_1\). Additionally, \(\nabla L^1 f_0 h_3, \nabla L^1 f_0 h_6, \nabla L^1 f_0 h_7, \nabla L^1 f_0 h_8\) together, would also make \(v_u\) observable, but have not been included in (6.24) to keep the matrix concise. \(\nabla L^0 h_2\) or \(\nabla L^0 h_4\), account for the observability of the orientation \(q_u\). The biases \(b_\omega\) and \(b_a\) are observable due to \(\nabla L^1 f_0 h_2\) and \(\nabla L^2 f_0 f_0 h_1\). Finally, the orientation offset \(q_i\) is observable due to \(\nabla L^0 h_5\).

For practical applications, there might be cases where the 3D city model is not available, thus making measurements \(h_3\) and \(h_4\) unavailable. Furthermore, while navigating through dense urban environments, the GPS measurements \(h_6, h_7, h_8\) might be unavailable. In these cases, theoretically the filter states would still be observable with the measurements \(h_1, h_2\) and \(h_5\). However, \(h_3, h_4, h_6, h_7\) and \(h_8\) act as corrections to prevent the state variables from drifting. We will examine this while implementing the filter.

### 6.3 UKF Equations

We implement an UKF [66] for our GPS-LiDAR integration, in order to accurately capture the non-linearity of the process and measurement models. In this section, we summarize the equations we use for the UKF:
• Initialize the augmented state vector and augmented state covariance matrix:

\[
\hat{x}_0 = E[x_0] \\
P_0 = E[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)^T] \\
\hat{x}_0^a = E[x^a] = \begin{bmatrix} \hat{x}_0^T & 0 & 0 \end{bmatrix}^T \\
\begin{bmatrix} P_0 & 0 & 0 \\ 0 & Q & 0 \\ 0 & 0 & R \end{bmatrix}
\]

Where:

\[
x^a = \begin{bmatrix} x^T \\ v^T \\ n^T \end{bmatrix}
\]

• Calculate sigma points:

\[
\chi^a_{k-1} = \left[ \hat{x}^a_{k-1} \hat{x}^a_{k-1} \pm \sqrt{(L + \lambda)P_{k-1}^a} \right],
\]

where \( \lambda \) is a scaling factor related to the spread of sigma points.

• Time update step:

\[
\chi^x_{k|k-1} = F[\chi^x_{k-1}, \chi^x_{k-1}] \\
\hat{x}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \chi_i^{x|k-1} \\
P_k^- = \sum_{i=0}^{2L} W_i^{(c)} [\chi_i^{x|k-1} - \hat{x}_k^-][\chi_i^{x|k-1} - \hat{x}_k^-]^T \\
\Upsilon_{k|k-1} = H[\chi^x_{k|k-1}, \chi^n_{k|k-1}] \\
\hat{\Upsilon}_k^- = \sum_{i=0}^{2L} W_i^{(m)} \Upsilon_{i,k|k-1}
\]

• Measurement update step:

\[
P_{\bar{y}_k\bar{y}_k} = \sum_{i=0}^{2L} W_i^{(c)}[\Upsilon_{i,k|k-1} - \hat{\Upsilon}_k^-][\Upsilon_{i,k|k-1} - \hat{\Upsilon}_k^-]^T \\
P_{x_k\bar{y}_k} = \sum_{i=0}^{2L} W_i^{(c)}[\chi_{i,k|k-1} - \hat{x}_k^-][\Upsilon_{i,k|k-1} - \hat{\Upsilon}_k^-]^T \\
K = P_{x_k\bar{y}_k} P_{\bar{y}_k\bar{y}_k}^{-1}
\]
\[
\tilde{x}_k = \tilde{x}_k^- + K(y_k - \hat{y}_k^-) \\
P_k = P_k^- - K P \hat{y}_k \hat{y}_k^T
\]

6.4 Implementation and Experimental Results

6.4.1 Experimental Setup

We use the iBQR UAV [67] designed and built by our research group for data collection. The UAV has an arm length of 0.6 m, and a payload capacity of 2 kgs. iBQR UAV provides us sufficient room to mount our sensors and on-board computer. We use a Velodyne VLP-16 Puck Lite LiDAR [53], a ublox LEA-6T GPS receiver [68] connected to a Maxtena antenna [69], and an Xsens Mti-30 IMU [70]. We use an AscTec MasterMind [71] as the on-board computer, to log the data from all these sensors in a rosbag file [29]. We limit the range of the LiDAR to 15 meters, in order to evaluate the LiDAR-based algorithms in certain under-constrained situations.

For our reference GPS receiver, we use a Trimble NetR9 [72] receiver connected to an antenna placed at the roof of Talbot Laboratory, University of Illinois at Urbana-Champaign. The reference receiver is within a kilometer of our data collection sites, which allows us to proceed with the assumptions we made in section 5.1 regarding a short baseline between the two GPS receivers.

6.4.2 Initializing Filter

To initialize the filter, we assume that the UAV begins operation in an open-sky environment with accurate and reliable GPS signals. We keep the UAV stationary and average the GPS receiver position output for the first few seconds to create the local GPS frame. It is important to have a reliable GPS frame, since it is used throughout the filter implementation.

We initialize all the state variables as zero, except for the orientation offset \( q_{i}^{i} \). We keep the UAV facing approximately North and average the first few seconds of IMU measurements to initialize \( q_{i}^{i} \). Finally, we initialize the state covariance matrix as an identity matrix.
Figure 6.2: Experimental setup for data collection. Our custom-made iBQR UAV mounted with a LiDAR, a GPS receiver and antennas, an IMU, and an on-board computer.

6.4.3 Filter Results

We implement the UKF on an urban dataset that we 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.

Before implementing the filter, we see how the different measurement sources perform for our dataset. As seen in Figure 6.3, the GPS measurements and the GPS position output contain large errors, due to the presence of nearby urban structures. Here we stack all the double difference measurements from (5.6) and compute the unweighted least square estimate of the baseline between the UAV and the reference receiver.

For the LiDAR measurements we check the output from simple ICP registration method as explained in section 2.1, the ICP implementation used in section 2.3, and our LiDAR - 3D city model matching algorithm described in Chapter 3. As seen from Figure 6.4, the first two methods accumulate drift over the course of the trajectory. The LiDAR - 3D city model matching algorithm does not drift over time, since the 3D city model is globally referenced. However, the position outputs still contain errors in situations where the LiDAR does not detect enough number of points or the matching algorithm converges to a local minimum.

Figure 6.5 shows the output of the filter for the same trajectory. The filter
Figure 6.3: Position estimates from GPS measurements. The position output from the GPS receiver (blue) and the unweighted least-squares position estimate (red) implemented on (5.6), contain large errors.

Figure 6.4: Position estimates from LiDAR point clouds. The positions estimated from incremental LiDAR odometry (green) and the ICP mapping implementation [21, 28] (blue) accumulate drift over time. The output from the LiDAR-3D city model matching (yellow) does not drift over time, but contains errors where the ICP algorithm might converge to a local minimum.
output estimates the actual path much more accurately than the individual measurement sources by themselves.

6.5 Summary

In this chapter, we described our method for integrating GPS and LiDAR measurements to estimate the UAV navigation solution. We detailed our Kalman filter structure followed by an observability analysis of the filter. Next, we showed the UAV and other experimental setup that we used for collecting data. Finally, we implemented our filter for an urban dataset and observed the improvement in positioning using our filter.
In summary, we proposed a GPS-LiDAR integration approach for estimating the navigation solution of UAVs in urban environments.

We used the on-board LiDAR point clouds in two ways: to estimate the odometry by matching consecutive point clouds, and to estimate the global pose by matching with an external 3D city model. We used the ICP algorithm for matching two point clouds. To analyse prior work and limitations of the ICP algorithm, we performed simulations in different environments. Furthermore, we implemented an ICP-based mapping method on urban datasets to verify the limitations of the ICP algorithm. For our 3D city model, we stated the sources and described the steps to create the model. We experimentally demonstrated the performance of our LiDAR - 3D city model matching algorithm.

Based on our analysis of the ICP algorithm, we built a model for the error covariance in the position estimates obtained from LiDAR point clouds. We modelled the error ellipsoid as a function of surface and edge feature points detected in the point cloud. We experimentally verified the modelled error covariance ellipsoid for urban point clouds.

For the GPS measurements, we used the individual pseudorange measurements from two receivers to create a vector double-difference measurements. We used the signal-to-noise ratio for individual pseudorange measurements, and propagated the uncertainty to obtain the covariance for the double-difference measurements.

Finally, we defined a Kalman filter structure to integrate the measurements from the LiDAR, the GPS receiver and an IMU. We performed an observability test for our filter structure, based on Lie derivatives. We implemented an Unscented Kalman filter, and experimentally demonstrated the improved positioning accuracy of our filter.
REFERENCES


