3D LiDAR-IMU Integration for State Estimation and Verification Using a GNSS/INS/LiDAR Simulation Chain

Daniela Sánchez-Morales, Mohamed Bochkati, Andreas Schütz, Shan Zhao, Thomas Pany
Institute of Space Technology and Space Applications (ISTA), Space Systems Research Center (FZ-Space)
Universität der Bundeswehr München

BIOGRAPHY

Sánchez-Morales Daniela studied Telematics Engineering at Instituto Tecnológico Autónomo de México (ITAM) in Mexico City and holds a Master's degree in satellite applications engineering from the Technical University Munich (TUM). Currently her research focuses on LiDAR, relative navigation algorithms particularly for terrestrial applications and sensor fusion.

Bochkati Mohamed Mohamed Bochkati is a research associate at the Institute of Space Technology and Space Applications (ISTA) at the Universität der Bundeswehr, München. His research focuses on Deeply Coupled GNSS/INS integration and novel calibration techniques for MEMS-IMUs. Before he joined the University of the Bundeswehr, he was a research associate at the Institute of Geodesy (ife) at the Leibniz University Hannover, where his research activities include quantum inertial navigation systems and GNSS receiver clock modeling. He holds a Bachelor's and Master's degree in Geodesy and Geoinformation from Technical University Munich.

Schütz Andreas is a Research Associate at the Universität der Bundeswehr München focusing on Sensor Fusion, mainly of GNSS Software Receiver Technology and inertial navigation sensors. He holds a Master's degree in Geodesy and Geoinformation from the Technical University of Munich (TUM).

Zhao Shan received a bachelor in engineering from University of Electronic Science and Technology of China in 2018 and is studying in Earth Oriented Space Science and Technology master program in Technical University of Munich (TUM), Germany. Since 2020 she has been a research assistant at the Institute of Space Technology and Space Applications.

Pany Thomas is with the Universität der Bundeswehr München at Space Systems Research Center (FZ-Space) where he leads the satellite navigation unit LRT 9.2 of the Institute of Space Technology and Space Applications (ISTA). He teaches navigation focusing on GNSS, sensors fusion and aerospace applications. Within LRT 9.2 a good dozen of full-time researchers investigate GNSS system and signal design, GNSS transceivers and high-integrity multi-sensor navigation (inertial, LiDAR) and is also developing a modular UAV-based GNSS test bed. ISTA also develops the MuSNAT GNSS software receiver and recently focuses on smartphone positioning and GNSS/5G integration. He has a PhD from the Graz University of Technology (sub auspiciis) and worked in the GNSS industry for seven years. He authored around 200 publications including one monography and received five best presentation awards from the US Institute of Navigation. Thomas Pany also organizes the Munich Satellite Navigation Summit.

ABSTRACT

Multi-sensor fusion is an inevitable part when designing an autonomous vehicle. The integration of measurements coming from different sensors has the advantage that the navigation system is more likely to provide a solution in a more varied range of scenarios. Due to the nature of the measurement principle of each sensor, the error dynamics are completely different and the strengths of a certain sensor compensate for the weaknesses of another. Nevertheless, in order to design and tune our navigation system according to the application we are targeting, it is needed to have a flexible framework where we can test different type of sensors, different sensor configurations, environments (urban, semi-urban, rural) and vehicle dynamics. Acquiring data from measurement campaigns is not only costly and time consuming, but usually one cannot have complete control of the environment and it is also challenging to acquire the ground truth data. For that reason, in this work, we present an extension to a previously introduced framework called SIMSS [1]. This framework is currently able to generate GNSS, IMU and LiDAR synthetic data and process it an RTK/INS/LiDAR loose coupled Kalman filter. In this work, we successfully show the quality of the generated synthetic data and the integration with our navigation modules within the MuSNAT. Hence demonstrating its use in testing and verification of our algorithms, which in turn gives us the opportunity to offer a more robust navigation system.

I. INTRODUCTION

In recent years, research on autonomous navigation in urban environments has increased and one of the most challenging tasks is to obtain a precise, reliable and robust positioning. GNSS technologies have played an essential role providing it, nevertheless they face particular problems that cannot be avoided, like: signal blockage, multipath (MP), interference, etc. Therefore, in order to guaranty a continuous service, one requires to add sensors that can support the GNSS when this is not available or when its accuracy falls below a certain level. Hence, multi-sensor systems are becoming vital to enhance the overall positioning and navigation.

The addition of sensors also means a higher complexity in the design of the system. It becomes more difficult the integration of a diverse range of technologies, as well as the synchronization and data handling. One also has to take into account the context in which the system is working, as the navigation system configuration may be optimized to operate in a certain environment or vehicle. More over, each sensor handles different hypotheses in their processing, including initialization, support of specific environmental features, light conditions, calibration procedures, etc. The ability of analysing and modifying all these parameters within a framework would not only help in profiling better the settings (sensors' and processing of raw data) to use for a particular test-case scenario but also to have a first glimpse of the results and modify accordingly the navigation filter. Furthermore, it allows repeatability of experiments, ground truth data is available and one is better prepared before performing and actual measurement campaign in the real world , making it a cost and time effective solution.

On a previous work [1], we presented the Satellite and Inertial Measurement Simulation System (SIMSS), where we successfully showed the generated synthetic data for both GNSS and IMU sensors. In this paper, we introduce an extension to the SIMSS-tool that is now connected to an open-source Autonomous Driving simulator called CARLA [2], from which we can get simulated LiDAR point clouds. The advantage that it brings compared with other approaches is that CARLA already supports a 3D LiDAR sensor that can be configured and from which we can obtain measurements with different properties. Moreover, one can have the possibility to test more realistic vehicle dynamics as the vehicle follows realistic traffic rules and traffic behavior. Contrary to the capabilities of CARLA, the SIMSS-tool is able to generate synthetic IMU observations, i.e. accelerations and rotation rates in the body-frame, where one can explicitly define more advanced error models for different classes of IMU sensors, as well as synthetic RTK-processable carrier-phase data at both intermediate frequency (IF) and RINEX level. Whereas CARLA only provides a very basic sensor model definition for that pair of sensors (gyro biases and white noise for all axes).

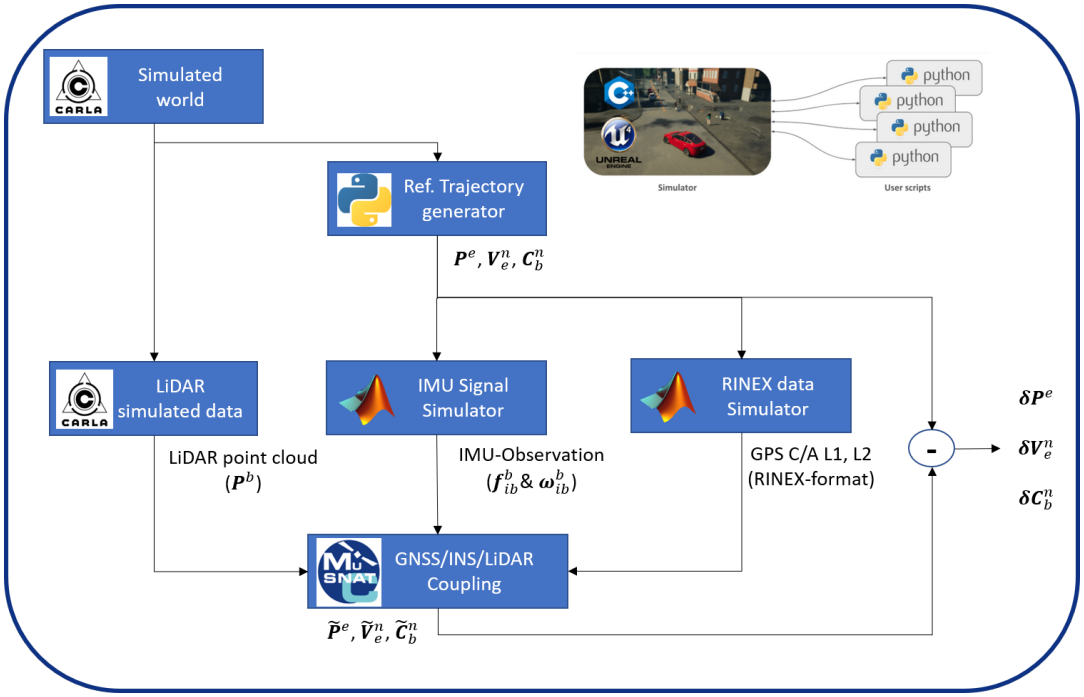

Figure 1: Extended SIMSS: Extended Satellite and Inertial Measurement Simulation System.

Figure 1 shows the architecture of our proposed approach. First, we have the simulated world coming from CARLA, where the environment model is loaded and actors like the vehicle and the sensors are setup. From this module we obtain the LiDAR point clouds and the ground truth trajectory. Therefore, the first section will be focused in describing in detail how this trajectory is generated and how the simulation is configured, as well as the work we have done to generate and ingest our own developed

maps. The above mentioned reference trajectory serves as an input for the IMU signal simulator and the RINEX data simulator. So in Section III, the methodology to compute the corresponding accelerations and rotation rates are presented. Next, in Section IV, we explain the sub-modules of the LiDAR processing chain and what the update used in later stages of processing consists of. For the present work, the RINEX data simulator does not play an important role as we are checking the capabilities of the tool to enhance the the positioning solution coming from supportive sensors. Nevertheless, it is needed for initialization within the filter and synchronization of data. It should be clarified that in the future we will work in scenarios where this tool is more exploded (i.e. GNSS outages along the trajectory). Section III briefly describes the conditions in which we ran our loosely coupled GNSS/INS/LiDAR integration [3] and finally, in Section VI a summary of ours results and conclusions is presented.

II. SYNTHETIC LIDAR DATA AND TRAJECTORY GENERATION

LiDAR sensors are used in a wide range of applications due to the high accuracy of its measurements and the fact that it is an active sensor, which is advantageous as it not depends on external light conditions. In recent years, development surrounding this technology has increased, specially for autonomous driving. This progress covers several topics: transition to smaller devices (solid state LiDARs, MEMS LiDARs), higher resolution and better distribution of the generated measurements, lossless compression of its data, detection and classification of objects, generation of 2D data structures from the original point clouds by using the intensity and depth information, etc. Simulators are not behind either. One could mainly separate them in the following: a) Geometry based [4] [5] [6], b) Physics based [7] [8], c) Generated from real data through AI [9], d) adapted from commercial video-games [10] [11].

Depending on the application, the complexity to get reliable data, worth to use for development and testing of algorithms, may vary. Apart from making the simulator flexible enough to use different sensor configurations, one depends highly on the scenes that are available and the detail in which they are constructed, i.e. the granularity of the meshes that represent each of the objects in the scene. The reason for this, is that most of the available simulators obtain the range that LiDAR provides by computing the intersection between a particular triangular mesh and the origin of the sensor. Therefore, the quality of the synthetic data and the realism of it will be directly related to the 3D model that is being used and its assets (trees, vegetation, cars, etc.).

Another important point to consider is that our main focus is terrestrial applications. Therefore, the data needed to develop the 3D models is more diverse compared, for example, with airborne laser scanning, where only an upper view of the scene is needed and which can be achieved by the integration of 2.5D elevation maps. As we are not interested into scanning objects while being static but when the vehicle is moving, we also need a tool where we can simulate the movement of the car.

As already mentioned, the use case is very important. Some simulators may even offer configurable settings for the sensor like beam divergence, which do not assume the laser beam as an infinite line but as a cone that leaves an elliptical foot print in the surface it intersects; some also provide the full waveform of the reflected laser pulse, very useful for forestry and vegetation classification. A more standard set of parameters would include the rate at which the sensor works, maximum range it reaches, number of laser beams, vertical/horizontal angular resolution and accuracy.

Having on mind the above-mentioned points, we decided to work with the CARLA simulator. It is grounded on Unreal Engine to run the simulation and uses the `OpenDRIVE` standard to define roads and urban settings. The advantages that it offers, specifically to our purposes, can be summarized as follows:

- The simulator supports a 3D LiDAR with the following configurable parameters: number of channels, maximum range, the rotation frequency of the lasers, Filed of View (FOV), atmosphere attenuation rate, drop-off zero intensity limit and noise standard deviation.
- The simulator also counts with LiDAR instance and semantic ground-truth data, which can be used for data driven classification or segmentation techniques.
- Ingestion of maps through 3rd party tools, allowing the possibility to import our own developed 3D models, which in turn facilitates the testing of different customized scenarios.
- One has access to its sensor suit and the control of static and dynamic actors (pedestrians, vehicles, traffic lights, etc.).

For the development of our LiDAR processing, working with such tool help us to analyze our performance in rich and poor feature environments, test the handling of occlusion and partial view of objects, experiment with different point cloud densities, differentiate between navigation-suited objects from those that cannot be taken as reference (e.g. moving cars) and, more importantly, examine our results under different car dynamics and trajectories.

1. Maps

A costumed map, created in `Roadrunner` [12], contains all assets that will be imported into the simulated world as well as the road definitions. Importing self-generated maps is optional, one can also use the pre-built maps offered by the simulator. For

initial tests we have used some of the already available environments. Nevertheless, we already have available a self-built map of a section of our campus. Table 1 summarizes the specifications of the GIS data used to construct it.

Data type	Format	Resolution
Aerial image	Tiff	+/- 0.2 m
Elevation map	ASCII	Height: +/- 0.2 m
Point clouds	LAS	Height: 0.1 m Positional: 0.5 m
Vector data	Shape file	n/a

Table 1: GIS data obtained from LDBV (Landesamt für Digitalisierung, Breitband und Vermessung)

Figure 2 shows a more detailed description of the process. After collecting all needed data to use as reference to develop the map, one may have to apply a geographic datum transformation in order to assure that the data is properly aligned. Here, one can start constructing all artifacts wanted in the final scene like buildings, roads, vegetation, markings, etc. The final output of this module is a FBX file containing the mesh of the scene and an OpenDRIVE file with the road network information, which is later imported into the simulator. An example of the ingested map of the UniBw München campus into the simulator is shown in Figure 3.

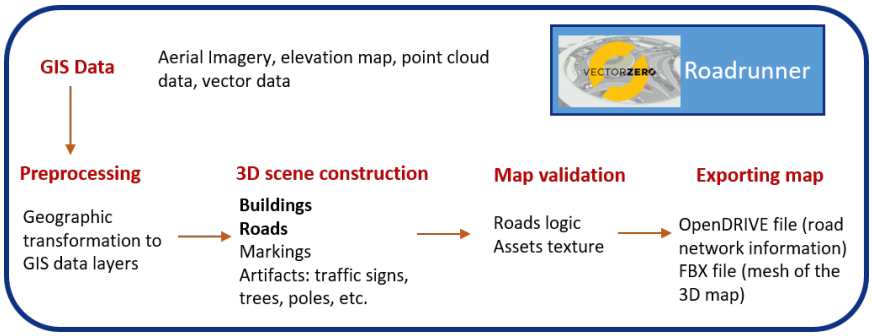

Figure 2: Workflow for generation and ingestion of maps in the CARLA simulator.

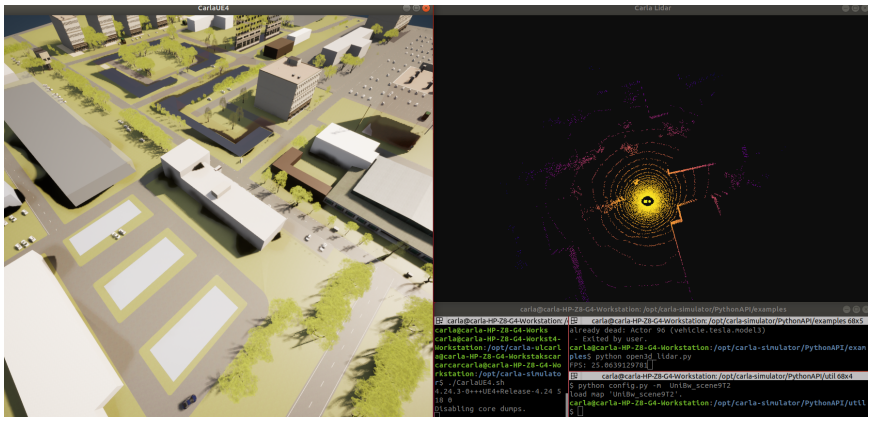

Figure 3: Ingested map of our university (UniBw München) in CARLA.

2. Trajectory generation

A simulation in CARLA is mainly composed of: a) the CARLA Simulator that computes the physics and renders the scene and all actor properties, b) client scripts written using a Python API, to spawn actors (vehicles, pedestrian, sensors and traffic signs), c) provision of the right configuration of the actors and their properties (attachment of sensors to the vehicle, configuration of the sensors, etc) and, d) retrieval of the sensor data, processing them (optional) and computation of the parameters needed by the controller for the next simulation step. Computed values are then sent back to CARLA Simulator, thus forming an client-server interaction loop.

For our development, we created two simple python modules. The first is called `Route Designer`. The goal of this module is to generate a route as a way-point file that can be saved for later use and as a log. Here, the user is able to give some basic parameters that prepares the world and type of ride, for example: the map to be loaded into the simulation (pre-built or self-generated), the average speed to keep between way-points and the starting point in which the vehicle will be spawned. Through the simulation the user is asked the path to choose at intersections. For the moment, we are not handling more complicated driving scenarios. The car drives forward on a specific lane and only at junctions the user can change the trajectory.

The second module is `Waypoint navigation`. It receives as input the waypoint file generated previously, from which information like average speed, map used and list of way-points is retrieved and used to setup in the current environment. The module is fixed to record the data of the following sensors: GNSS, IMU and LiDAR. In this part of the workflow is when one can use different configurations of the LiDAR sensor given the same trajectory i.e. same features along the path the vehicle is driving but different sensor properties. The main output of this module is the vehicle state, which is obtained at the rate at which the simulator is configured as well as the sensors' data.

For the extended SIMSS tool, the simulated world becomes the main link between all simulation tools, as it provides the reference trajectory needed as input in the IMIU Signal Simulator and the Rinex simulator. In that way we assure that the dynamics of the car are represented correctly in the three data sets. Once the data is generated, it can be used as a normal sensor recording to be processed in the different navigation modules that the MuSNAT provides.

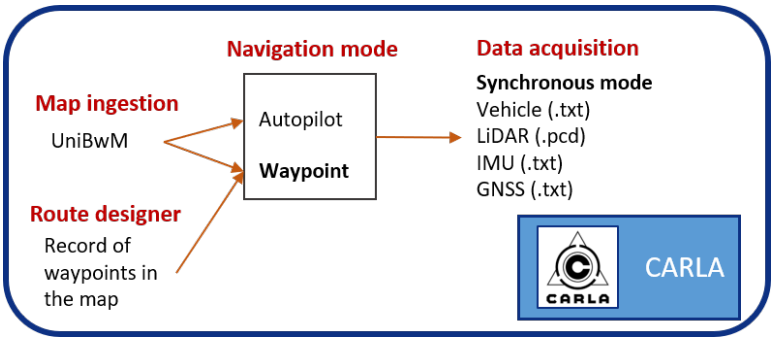

Figure 4: Workflow for trajectory generation.

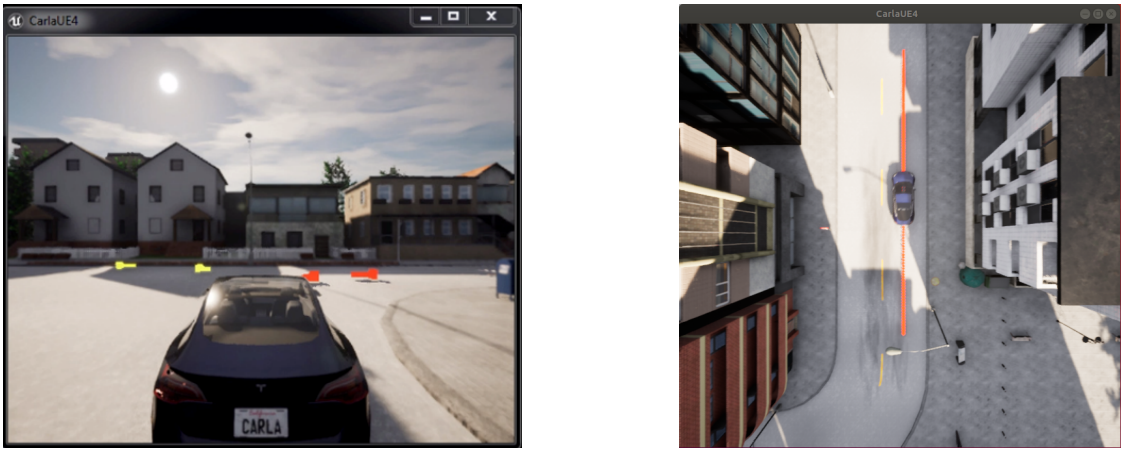

Figure 5: Trajectory generation, [left] route designer and a cluster classified as *Structural Element* and [right] Waypoint navigation

3. CARLA time synchronization and sensors' data

The simulated world, maintained by the server, has its own clock and time. The time-step can be fixed or variable depending on the user's preferences. By default, the asynchronous mode is used by the simulator. Nevertheless, as we are retrieving information from more than one source, synchronization becomes relevant. Therefore, we work in synchronous mode with fixed steps. That means that the complete control of the simulation is taken by the client and the server always waits for the next instruction. This ensures that the simulation time and physics will be synchronous and the acquisition of data coming from the sensors is reliable. For the test done in the current work, the simulation was set to have fixed steps of 0.05 seconds.

The simulated LiDAR is vertical and mounted on top of the roof of the vehicle with a translation w.r.t. the body frame of the car of (X = 0.0 m, Y = 0.0 m, Z = 2.0 m). It has the same characteristics as a Velodyne VLP-16 (See Table 2), but without adding any noise to the range measurements.

The GNSS and IMU are placed at the origin of the coordinate system of the vehicle, so no offset is introduced. It is worth to mention that the IMU measurements were just taken as a reference, for a basic comparison with the ones we are generating. The GNSS, on the other hand, are taken as additional information to the vehicle's state.

From the vehicle itself the position in the map coordinate system and the velocity are extracted.

Parameter	Value
No. Channels	16
Maximum range	100 m
Rotation frequency	10 Hz
Lower FOV	-15.0 °
Upper FOV	15.0 °
Atmosphere attenuation rate	0.004
Dropoff general rate	0.01
Dropoff zero intensity limit	0.01
Noise standard deviation	0.0

Table 2: LiDAR configuration

III. TRAJECTORY-BASED IMU SIMULATION

The aim of this section is to demonstrate the simulation of the IMU observations, i.e. acceleration $\boldsymbol{f}_{ib}^{b}$ and rotation rates $\boldsymbol{w}_{ib}^{b}$ along a predefined trajectory expressed in a geographic coordinate system (longitude, latitude and height), e.g. WGS84. This trajectory is generated using the waypoint navigation of CARLA where also the LiDAR-point clouds simulation take place. Here we assume that the IMU-axes are mounted on the virtual car according to the ENU realisation where the y-axis is pointing to the forward direction, the x-axis is perpendicular to the driving direction and the z-axis completes the right hand system. From our experience, the optimal sampling rates to design a trajectory in CARLA i.e. 20 Hz [6], Therefore, the corresponding IMU observation will have the same sampling speed, which is not sufficient to be fused with LiDAR or GNSS, since the minimum required/recommanded sampling rates should be at least 100 Hz. For this reason, it is worthy to simulate this sensor with different sampling frequencies which allow to cover all dynamic scenarios.

The main advantage of this tool, is the ability to simulate a variety of sensor errors such as bias instability (BI), random walk (RW), scale factor (SF) errors, sensor errors due to thermal drift, non-orthogonalities, misalignment, and their combinations according to manufacturer spec. sheets of real IMUs. Additionally, since we are accounting for all physical parameters for inertial navigation, such as a global gravity model, earth rotation, transport rates [13] etc., this tool can be used for local small trajectories, i.e. couple of meters, as well as long-distances, i.e. thousands of kilometers. The mathematical model that explains the generation of error-free accelerometer and gyroscope signals based on a georeferenced trajectory will be described in details in next subsections.

1. Accelerometer Signal Simulation

According to [14] the accelerometer readings in the body-frame of the IMU can be described by the following equation

$$\boldsymbol{f}_{ib}^{b} = \boldsymbol{C}_{n}^{b}\left[\ \dot{\boldsymbol{v}}^{n} + (2\boldsymbol{\omega}_{ie}^{n} + \boldsymbol{\omega}_{en}^{n}) \times \boldsymbol{v}^{n} - \mathbf{g}^{n}\ \right] + \delta\boldsymbol{f}_{ib}^{b} \tag{1}$$

where $\boldsymbol{f}_{ib}^{b}$ is the specific force vector along the three axes the IMU and $\boldsymbol{C}_{n}^{b}$ is the rotation matrix between the *n-frame* (navigation frame) and the *b-frame*. $\boldsymbol{v}^{n}$ is the vehicle velocity in the *n-frame* (v^{E}, v^{N} and v^{U}). Since it not straightforward to simulate all three velocity component in the same time, some assumption should made to reduce the number of the design parameters, which can be achieved by applying the so-called *non-holonomic* constraints [15]. This physical constrains is valid only for terrestrial vehicles and assumes that the velocity in the plane perpendicular to the forward direction is almost zero ($v_{x}^{b} \approx 0$ and $v_{z}^{b} \approx 0$), so that only the forward speed v_{y}^{b} along the y-axis is measurable. For simplicity reason we rename v_{y}^{b} as v. Therefore, the velocity vector is equal to

$$\boldsymbol{v}^n = \begin{bmatrix} v^E \\ v^N \\ v^U \end{bmatrix} = \boldsymbol{C}_b^n \boldsymbol{v}^b = \boldsymbol{C}_b^n \begin{bmatrix} 0 \\ v \\ 0 \end{bmatrix} = \begin{bmatrix} v\sin\psi\cos\theta \\ v\cos\psi\cos\theta \\ v\sin\theta \end{bmatrix} \tag{2}$$

From the right part of the above equation it can bee seen that only the yaw ψ and the pitch angle θ are introduced in this simulation and thus the rotation angle around y-axis of the IMU, i.e. the roll angle ϕ cannot be generated.

The vehicle acceleration $\dot{\boldsymbol{v}}^n$ can be obtained by differentiating the velocity $\boldsymbol{v}^n$ w.r.t. time as follows

$$\dot{\boldsymbol{v}}^n = \frac{d\boldsymbol{v}^n}{dt} = \frac{d}{dt}\begin{bmatrix} v\sin\psi\cos\theta \\ v\cos\psi\cos\theta \\ v\sin\theta \end{bmatrix} = \begin{bmatrix} \dot{v}\sin\psi\cos\theta + \dot{\psi}v\cos\psi\cos\theta - \dot{\theta}v\sin\psi\sin\psi \\ \dot{v}\cos\psi\cos\theta - \dot{\psi}v\sin\psi\cos\theta - \dot{\theta}v\cos\psi\sin\psi \\ \dot{v}\sin\theta + \dot{\theta}v\cos\theta \end{bmatrix} \tag{3}$$

$\boldsymbol{\omega}_{ie}^n$ denotes the Earth rotation rate projected on the *n*-frame, which can be expressed as follows

$$\boldsymbol{\omega}_{ie}^n = \boldsymbol{C}_e^n \boldsymbol{\omega}_{ie}^e = \begin{bmatrix} 0 & \omega_e\cos\varphi & \omega_e\sin\varphi \end{bmatrix}^T \tag{4}$$

$\boldsymbol{\omega}_{en}^n$ is the transport rate, which refers to the change of orientation of the *n*-frame with respect to the Earth.

$$\boldsymbol{\omega}_{en}^n = \begin{bmatrix} -\dot{\varphi} & \dot{\lambda}\cos\varphi & \dot{\lambda}\sin\varphi \end{bmatrix}^T = \begin{bmatrix} -\dfrac{v^N}{R_M+h} & \dfrac{v^E}{R_N+h} & \dfrac{v^E\tan\varphi}{R_N+h} \end{bmatrix}^T \tag{5}$$

Where R_N and R_M represent the two radii of curvature, i.e. the normal radius and the meridian radius [16]
$\mathbf{g}^n$ is the Earth's local gravity vector [16], which is written as

$$\mathbf{g}^n = \begin{bmatrix} 0 & 0 & -\mathrm{g} \end{bmatrix}^T \tag{6}$$

$\delta \boldsymbol{f}_{ib}^b$ represents the accelerometer's errors, which depend on the error model assumptions.

2. Gyroscope Signal Simulation

Since the rotation between coordinate frames can be expressed as a sum of rotations, the output of the gyroscope triad can be written as

$$\boldsymbol{\omega}_{ib}^b = \boldsymbol{C}_n^b(\boldsymbol{\omega}_{ie}^n + \boldsymbol{\omega}_{en}^n) + \boldsymbol{\omega}_{nb}^b + \delta\boldsymbol{\omega}_{ib}^b \tag{7}$$

where

$\boldsymbol{\omega}_{nb}^b$ is the mathematical gimbal's rate for a strap-down inertial system, which has components related to the vehicle's attitude and attitude change rate [13].

$$\begin{aligned} \boldsymbol{\omega}_{nb}^b &= \begin{bmatrix} 0 \\ \dot{\phi} \\ 0 \end{bmatrix} + \boldsymbol{C}_2^T(\phi)\begin{bmatrix} \dot{\theta} \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{C}_2^T(\phi)\boldsymbol{C}_1^T(\theta)\begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \\ &= \begin{bmatrix} \cos\phi & 0 & \cos\theta\sin\phi \\ 0 & 1 & \sin\theta \\ \sin\phi & 0 & \cos\theta\cos\phi \end{bmatrix}\begin{bmatrix} \dot{\theta} \\ \dot{\phi} \\ \dot{\psi} \end{bmatrix} \end{aligned} \tag{8}$$

$\delta\boldsymbol{\omega}_{ib}^b$ represents the gyroscope errors.

3. CARLA Trajectory Design

To generate simulated observation of both accelerometer and gyroscope, as indicated in Eqs. [1] and [7], many parameters such as velocity and attitude information are needed. As mentioned before, we are using the waypoint option from CARLA to output a go-referenced trajectory (λ, φ and h) which has a maximum sampling rates of 20 Hz. In the next step, a piece-wise cubic polynomial curve [17] can be fitted to the selected coordinates with different sampling frequencies, e.g. 100 Hz

$$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{9}$$

where a_i, b_i,c_i and d_i represent the coefficient to be estimated.

If $s_i(x)$ is evaluated at $x = x_i$, and for instance, if φ is the evaluated parameters, so $\dot{\varphi}$ would correspond to the coefficient b_i and likewise, the second derivative $\ddot{\varphi}$ would be equal to $2c_i$.

The main advantages of this method is, that it is possible to construct cubic functions $s_i(x)$ on each interval $[x_i, x_{i+1}]$ so that the resulting piece-wise curve $y = s_i(x) = s(x)$ and its first and second derivatives are all continues. Hence, based on the aforementioned characteristics it is possible to deduce the first as well as the second time derivative of the latitude, longitude and height analytically. So, the error due to integration can be only committed in the first step, i.e. by fitting the splines to the given curve-linear coordinates. On the other side, if any other numerical differentiation method is used, the associated numerical error, beside the error due to interpolation, would be present in both first and second derivative of the trajectory.

The velocity component in the the north direction can be deduced from Eq. 5

$$v^N = \dot{\varphi}\,(R_M + h) \tag{10}$$

therefore the change of v^N is equal to

$$\dot{v}^N = \frac{dv^N}{dt} = \frac{d}{dt}\Big(\dot{\varphi}\,(R_M + h)\Big) = \ddot{\varphi}\,(R_M + h) + \dot{\varphi}\,(\dot{R}_M + v^U) \tag{11}$$

where $\dot{h} = v^U$

Similarly, the rate of change of the longitude λ

$$v^E = \dot{\lambda}\,(R_N + h)\cos\varphi \tag{12}$$

consequently

$$\dot{v}^E = \frac{dv^E}{dt} = \frac{d}{dt}\Big(\dot{\lambda}\,(R_N + h)\cos\varphi\Big) = \ddot{\lambda}\,(R_N + h)\cos\varphi + \dot{\lambda}\,(\dot{R}_N + v^{Up})\cos\varphi - \dot{\varphi}\,\dot{\lambda}\,(R_N + h)\sin\varphi \tag{13}$$

The formula of the time derivative of the radius of curvature in the prime vertical and in the meridian can be found in [16]

In this simulation the height component is assumed to be constant, so that the propagation of the coordinates only in the horizontal plane takes place .

Finally, the attitude information can be obtained from Eq. 2

$$\psi = \arctan\Big(\frac{v^N}{v^E}\Big); \quad \theta = \arctan\Big(\frac{v^U}{\sqrt{{v^N}^2 + {v^E}^2}}\Big); \quad \phi = 0 \tag{14}$$

4. Simulation Results

Using the CARLA's waypoint navigation method, a short trajectory has been generated (see Fig. 6). After applying the previous equations the corresponding error-free accelerometer $\boldsymbol{f}_{ib}^b$ and gyroscope $\boldsymbol{\omega}_{ib}^b$ signals can be simulated, as depicted in Fig. 7. These observation have been transformed from the ENU-frame into the NED navigation frame in order to be compatible with GNSS/INS filter implementation [1]. In the accelerometer z-axis the slight change of gravity can be observed, which cannot be provided by CARLA. Furthermore, the IMU signal is almost spike-free which increases the accuracy of these data during the strapdown process.

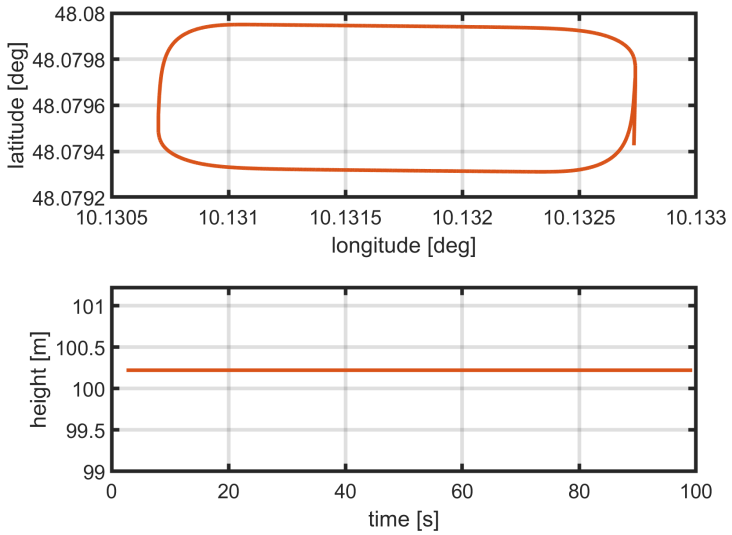

Figure 6: Reference trajectory generated by means of CARLA's waypoint navigation

Before introducing the simulated IMU observation into the GNSS/INS/LiDAR fusion, the accuracy i.e. their error contribution within in the budget of the entire tool chain, should be assessed. This can be achieved, through a closed-loop simulation, where and Input-Output trajectory comparison take place. The output trajectory can be obtained when integrating the simulated observation using pure strapdown computation. In our case (Fig. 8), the closed-error on the position and velocity level is maximal 0.9 m and 0.008 m/s respectively.

IV. LIDAR PROCESSING

In order to obtain accurate estimates about the displacement of the vehicle, one has to process the point clouds recorded by the LiDAR. Our approach consist in removing outliers and keeping points belonging to objects that can be trusted and used for navigation purposes. Such points mostly represent non-moving objects that can be found in the environment. Therefore, clustering of points that represent individual components in the scene is really important. Furthermore, the capability of distinguishing which from those preliminary clusters can be used for the final registration has a very important role in our processing. For the moment, we are only capable of distinguish between vertical and structural elements, as described in the following subsections. Figure 9 shows the general workflow of the LiDAR processing.

1. Ground segmentation

The ground, in our case, is not providing any valuable information. On the contrary if all those points are successfully extracted, the segmentation of the remaining points into different clusters becomes easier.

The method that we chose is based on the work of [18] and [19]. Some of the main reasons of adopting this method are that it targets applications running in real-time, it was designed and tested for outdoor environments and that it only depends in two configurable parameters.

One of the main characteristics of this approach is that it reduces the dimension of the point cloud. This is done by dividing it into several segments along the azimuth angle by $\Delta\alpha$. For each of the segments, a discretization along the range is done by the definition of $Range_{lims}$, having as result the partition of the cloud into several bins. These are the 2 main parameters to tune, depending on the characteristics of the LiDAR sensor (vertical and horizontal angular resolution). From each of the bins belonging to a specific segment, a point is selected. Therefore, one ends up with a 2D data set per segment, consisting of the height, z, and range, ρ, value of the chosen points. To each segment, a 2D line extraction method is applied in order to classify points as ground and non-ground points. All points belonging to the line being extracted, based on certain thresholds and line parameters are said to be part of the ground (See Figure 10 [left]). An example of the detection and extraction of the ground can be seen in 10 [right].

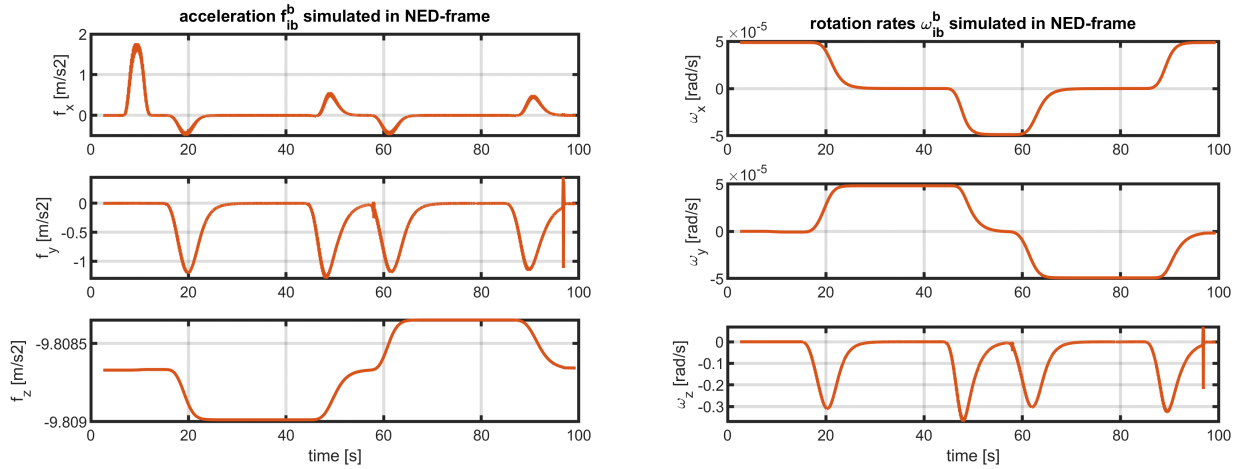

Figure 7: Error-free simulated IMU observation @300 Hz, specific force of the accelerometer (left) and rotation rates of the gyroscope (right)

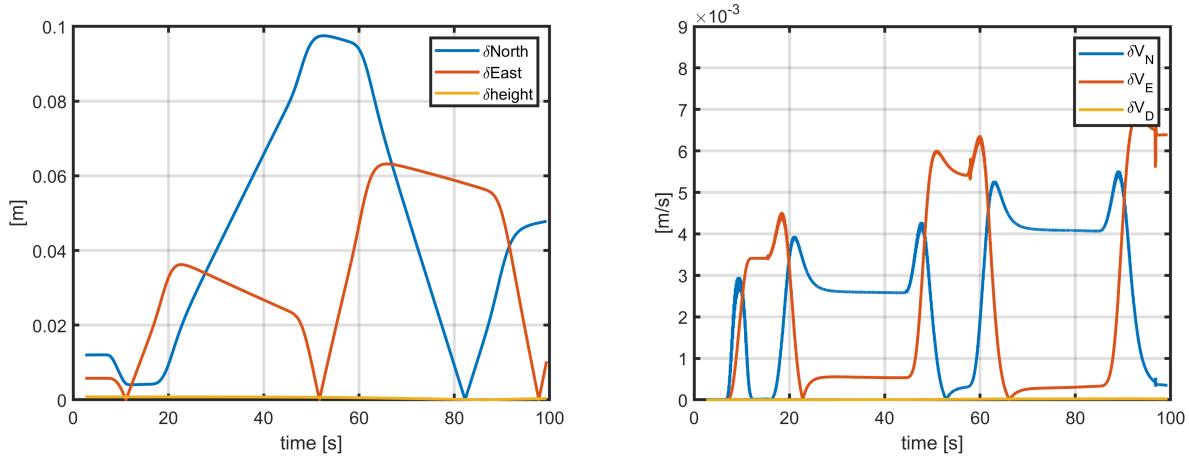

Figure 8: Closed-Loop simulation error for both position (left) and velocity (right) expressed in NED-frame

2. Range image generation and segmentation

Once the ground is extracted, a set of points are left and one has to group them according to a certain criteria. In the best case, each sub-group will represent an object from the real world. There have been many attempts to provide algorithms that accomplish this task. The main challenge is to deal with under and over segmentation. That means, avoiding that a cluster describing an object of the real world is separated falsely into more than one piece or correspondingly, that a cluster is conformed of more that one object. Most of these algorithms ranging from region based, model based, attributes based to graph based [20] highly depend on the definition of complex relationships between neighbors or the quality of derived attributes to obtain good results. Some of them may be inaccurate when dealing with different point cloud sources or they can simply not run in real time.

On the other hand, working with range images (See Figure 11[top]), which are often small, dense, and maintain the neighborhood information implicitly in its 2D structure, gives the opportunity to apply simple methods that give fairly good results. Additionally, operating on them is substantially faster than reasoning on the 3D point cloud.

To segment the range image, we followed [21]. The depth angle, β, is the angle that relates the distance in depth between two points and based on this parameter one decides if it belongs or not to the same object. It is computed as follows:

$$\beta = arctan\left(\frac{d_2 sin(\alpha)}{d_1 - d_2 cos(\alpha)}\right), \tag{15}$$

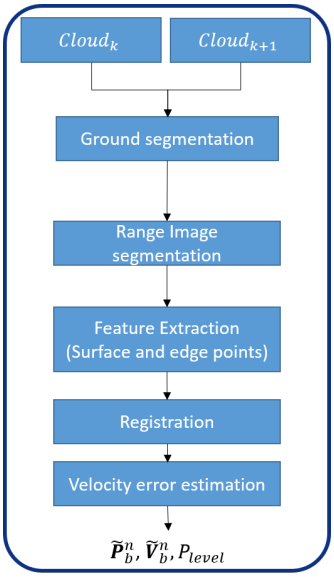

Figure 9: LiDAR processing workflow.

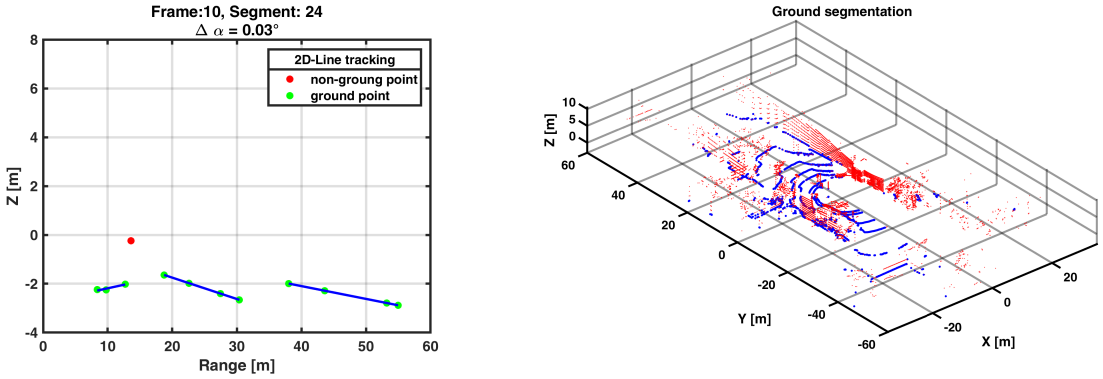

Figure 10: Ground extraction module: Example of line tracking for a given segment in a point cloud [left] and the point cloud classified as ground (blue) and non-ground (red) points [right]

where d_1 and d_2 correspond to the range measurements of the points of interest and α the angle between them.

The depth angle should stay relatively large for most objects. If beta is greater than a certain threshold, $\beta > \theta$, we consider the points to represent one object. But if the change in depth is too large, which means that beta has a small value, we make the decision to separate the points into different segments.

Based on this concept, we can do the segmentation on the range image using the Breath First Search (BFS) algorithm in order to connect components, having in mind the constraint of the β angle. Based in our tests with the LiDAR using the VLP-16 configuration, we have seen that two thresholds are needed to compensate for the difference between the vertical and horizontal angle resolutions. Hence, we setup the $\theta_H = 1.0°$ and $\theta_V = 0.1°$. Results of the segmentation are shown in Figure 11[bottom].

Elongated buildings along the horizontal plane tend to be over segmented. Nevertheless, for the following modules, this does not affect the processing as long as the individual parts do not become too small.

3. Feature extraction

As mentioned previously, from a preliminary pool of clusters, we want to keep the ones we can use in the registration module. In order to classify and keep the elements that we support, we first need to compute the curvature value of each of the points in

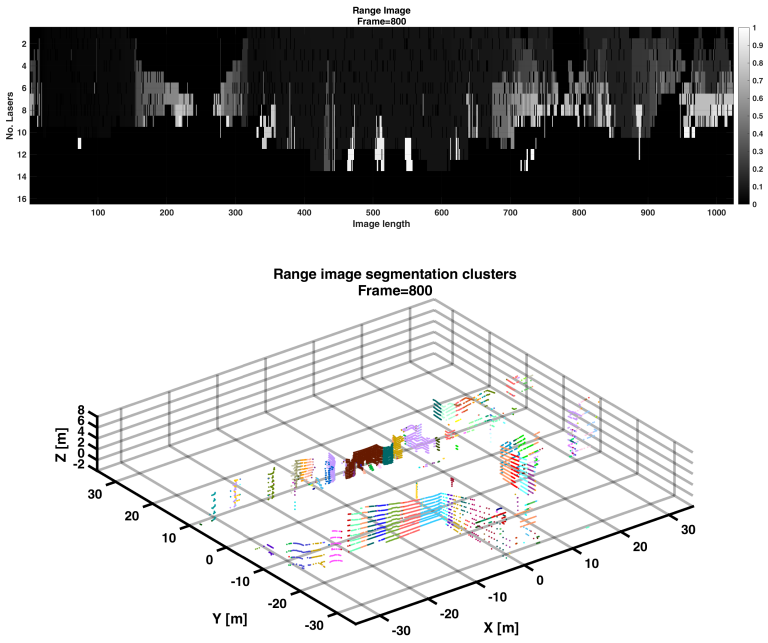

Figure 11: Range image segmentation module: Example of a generated range image [top] and the clusters resulting from the segmentation [bottom]

the clusters as shown in Equation 16.

$$c_k = \frac{1}{\|\boldsymbol{p_k}\|} \left\| \sum_{l \in B, l \neq k} \boldsymbol{p_k} - \boldsymbol{p_l} \right\|, \tag{16}$$

where $\boldsymbol{p_k}$ is the k^{th} point of the LiDAR point cloud and B is the neighborhood of 15 points near k. It is worth to mention that from the initial pool, we discarded clusters that had less that 30 points, as we assume that we cannot derive any classification from those.

As our point cloud is sparse, and the geometry of the simulated LiDAR starts to diverge with respect to the distance, we used different thresholds to classify the points between edge and surface point features. But for all sections the curvature value of a point holds that those close to zero belong to surfaces, otherwise they are labeled as edges. Figure 12 illustrates the values that a specific cluster can get.

Based on the values that each cluster got and some geometrical constraints, we classify the cluster as a *Structural element*, *Vertical element* or *No Class*. Then, we refine the cluster based on its classification, where we down sample, and correct for misclassified points. For the Structural elements for example, as they are composed of edge feature points and surface points (See Figure 13 [right]), we do not only do class correction but we also look for vertical elements. This will later help to compute the correct covariance matrix belonging to this feature.

4. Registration

It is intended to overlap consecutive scans in order to perform the most accurate possible alignment between them, finding the correspondent transformation matrix that allows an accurate computation of the displacement of the vehicle.

The Iterative Closest Point algorithm (ICP) reduces the rotation and translation values in a given number of iterations until the error is an admissible value for the correspondent application. When any of the boundary conditions are met (through differential transformation checkers), the algorithm should provide a rotation matrix, R_{ICP}, and translation vector, t_{ICP}, matching the two point clouds as best as possible. For that purpose, we used the ICP from `libpointmatcher` [22].

The input point clouds for this module already contain the set of clusters that we were able to classify. Nevertheless, it may happen that some of the clusters do not have their counter part in the next frame. This may happen due to several reasons: filtered

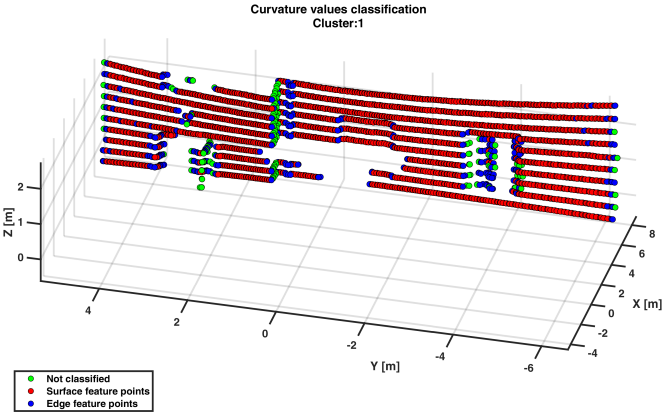

Figure 12: Classification of feature points based on its curvature

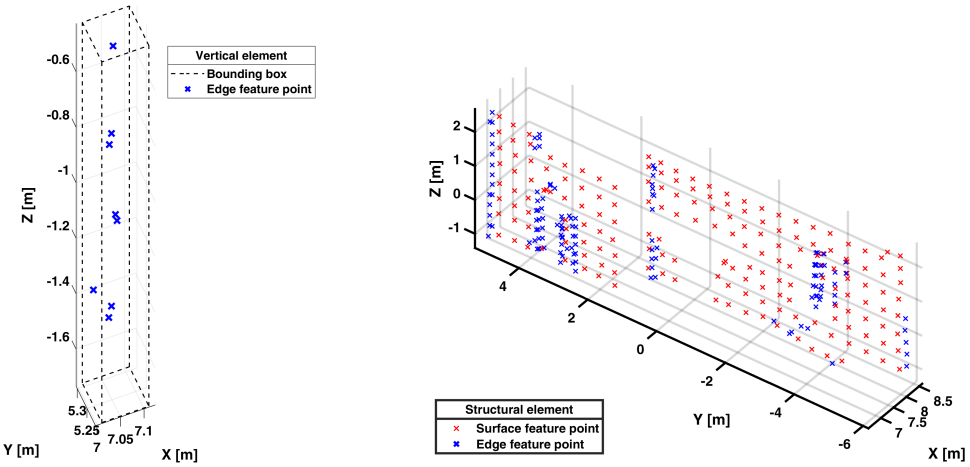

Figure 13: Feature extraction module: Example of a cluster classified as *Vertical Element* [left] and a cluster classified as *Structural Element* [right]

during the processing due to the low density of points, misclassification, occlusion of the object during the data acquisition , etc. Hence we use a filter that ranks the points in the reading cloud by their distance to the reference after a transformation was applied and only a percentage of it is used to continue with the iteration process.

The *point to plane minimizer* is highly recommended for the type of features that our scene has. As we are in an urban area, we want to take advantage of the normals of the *Structural elements* as they are mostly planes. Figure 14 shows the reconstructed trajectory using only the obtained displacements, one can notice that they are in the centimeter level of accuracy. The error starts to increase at the end of the trajectory, but this is due to the environmental features of the scene at that part of the path (green area).

5. Position error estimation

It is assumed that the position error coming from the registration heavily depends in the features that surrounds the sensor. For instance, a featureless environment results in a high position error along all axes. Whereas well distributed features along the path in which the vehicle drives should result in a successful registration and hence in an accurate position.

We have adapted the work of [23] to our development. They propose to model the position error as a multivariate Gaussian distribution whose covariance matrix is characterized based on the feature location and orientation with respect to the LiDAR.

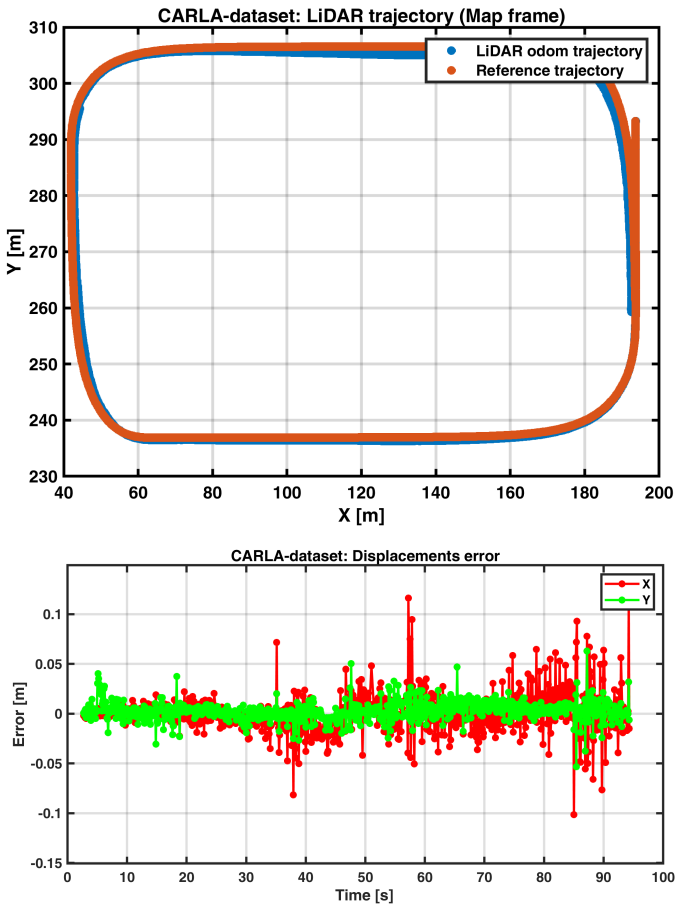

Figure 14: Registration module: Reconstructed trajectory based on the computed displacements [top] and corresponding displacement errors [bottom]

In our adaptation, we compute a covariance matrix per cluster feature. That is to say, that we compute a covariance matrix, $\boldsymbol{R}_s$ (See Equation 17), per surface cluster and another, $\boldsymbol{R}_e$ (See Equation 18), per edge cluster. As mention in the previous section, we have classified the clusters as *vertical elements* and *structural elements* and they, in turn, are composed of surfaces and edges clusters.

$$\hat{\boldsymbol{R}}_i^s = \boldsymbol{V}_i^s \cdot \boldsymbol{L}_i^s \cdot \boldsymbol{V}_i^{s^{-1}} \tag{17}$$

$$\hat{\boldsymbol{R}}_j^e = \boldsymbol{V}_j^e \cdot \boldsymbol{L}_j^e \cdot \boldsymbol{V}_j^{e^{-1}} \tag{18}$$

For a *surface cluster*, one first obtain the surface normal, $\hat{\boldsymbol{s}}_i$, and to create its 3D error ellipsoid an orthonormal basis is created as follows:

$$\hat{\boldsymbol{V}}_i^s = [\hat{\boldsymbol{s}}_i \quad \hat{\boldsymbol{n}}_i^s \quad \hat{\boldsymbol{m}}_i^s] \tag{19}$$

where $\hat{\boldsymbol{n}}_i^s$ is chose to be perpendicular to $\hat{\boldsymbol{s}}_i$ and $\hat{\boldsymbol{m}}_i^s$ is only the cross product of the previous two.

Then we use Equation 20 to define the eigenvalues corresponding to the eigenvectors in Equation 19 as follows:

$$\hat{\boldsymbol{L}}_i^s = \begin{bmatrix} a^s & . & . \\ . & b^s & . \\ . & . & b^s \end{bmatrix} \tag{20}$$

As each *surface cluster* contributes reducing position error in the direction of the corresponding surface normal, we set $a^s << b^s$. Both are constants and in our implementation, we decided to set the value of a^s to the range noise specification of the sensor $4\,cm$ and b^s to $3.5\,m$.

Similarly with the *vertical clusters*, one makes the hypothesis that the cluster helps in reducing position error in the directions perpendicular to the edge vector, $\boldsymbol{n}_j^e$, or in other words, the horizontal position error. Hence, we use as eigenvectors the following,

$$\hat{\boldsymbol{V}}_i^e = [\hat{\boldsymbol{e}}_i \quad \hat{\boldsymbol{n}}_i^e \quad \hat{\boldsymbol{m}}\boldsymbol{n}_i^e] \tag{21}$$

and as eigenvalues,

$$\hat{\boldsymbol{L}}_i^e = \begin{bmatrix} a^e & . & . \\ . & b^e & . \\ . & . & b^e \end{bmatrix} \tag{22}$$

In this case, $a^e >> b^e$, where a^e is equal to $3.5\,m$, and b^e is $4\,cm$.

Finally, once all covariances are computed for the respective type of feature clusters, one can obtain the covariance matrix that represents the estimated position error for that specific frame, $\hat{\boldsymbol{R}}_{adaptative}$, by combining all the individual contributions as follows:

$$\hat{\boldsymbol{R}}_{adaptative} = \left(\sum_{i=1}^{n} (\boldsymbol{R}_s)^{-1} + (\boldsymbol{R}_e)^{-1} \right)^{-1}. \tag{23}$$

6. LiDAR update

The LiDAR update that is fed into the filter consist in the estimated velocity, a protection level and a validity flag. The first is easily computed by:

$$\tilde{\boldsymbol{V}}_b = \frac{\Delta \boldsymbol{X}_{ICP}}{\Delta t}, \tag{24}$$

where $\Delta \boldsymbol{X}_{ICP}$ is the displacement computed in the registration module.

The protection levels are bounds over the velocity error and they give a measure to the user of how much they can trust the measurement its being given. To compute it, we take $3\,\sigma$ of the computed velocity error estimate

$$\boldsymbol{P_L} = \alpha \sqrt{diag\left(\frac{\hat{\boldsymbol{R}}_{adaptative}}{\Delta t} \right)}, \tag{25}$$

where α=3 and Δt is the time span between two acquired point clouds.

Figure 15 shows an integrity plot corresponding to the trajectory, subject of study in this work. As one can see most of the measurements are contained within the computed bounds, i.e. $\boldsymbol{V}_\varepsilon < \boldsymbol{P}_L$. Nevertheless, one can notice some longitudinal velocities that were estimated incorrectly. They correspond to a section at the end of the trajectory, where the environment was composed of features that were not supported or difficult to detect (green areas). Due to this reason an inaccurate displacement was computed and hence the velocity estimate. In future developments, we seek to use the estimated attitude computed by the filter to help the ICP to converge to the correct solution, as we have seen that the main source of error does not come from the computed translation, but from the rotation matrix that the algorithm provides.

Lastly the validity flag, allows the filter to distinguish between a successful or unsuccessful registration result. This is done by checking the RMSE of the distances of the matched points between the registered point clouds. If the resulting RMSE is greater than a certain threshold, it is assumed that the ICP was not able to find the correct transformation that aligns accurately the point clouds. Therefore, even if the sensor had nice features in the surroundings in a particular moment and thus a small estimated velocity error, the validity flag will set to false and the update will be dropped.

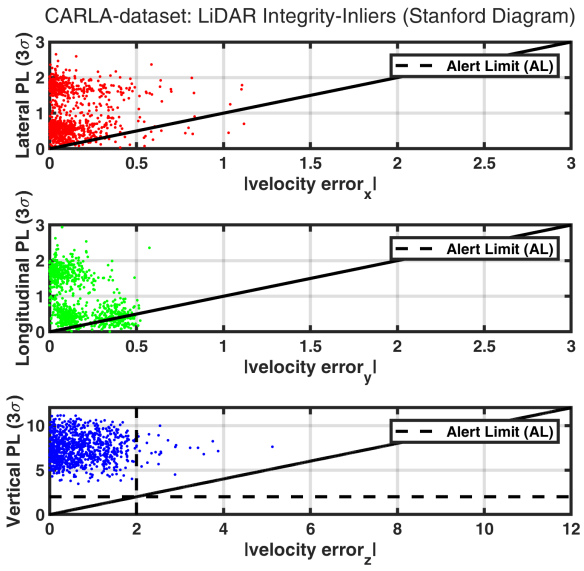

Figure 15: Integrity of LiDAR velocity estimates (Standford-like integrity plot)

V. INS/LIDAR-EKF AND SIMULATION RESULTS

In a previous work [3], we introduced our GNSS/INS/LiDAR navigation engine, implemented within the MuSNAT [24]. This navigation module is able to receive position updates from the GNSS, velocity updates from the LiDAR and perform the strapdown computation using the IMU measurements.

Some of the most interesting tasks for which we have developed the Extended SIMSS are to try the diverse situations that can be encounter within the filter. For example, having available updates from all sensors and accepting or declining one over another. Up to now, in our implementation, a valid GNSS update is accepted over a LiDAR update. But it is of our interest to enhance the detection of false fixes coming from GNSS or outliers velocities coming from the LiDAR that were not correctly detected within its module.

For the purpose of the present work, we want to introduce how the Extended SIMSS tool is capable of aiding in such analysis. For that reason, we have chosen a scenario where only LiDAR updates and IMU measurements are available. Given the presented results in previous sections, the expected outcome from the filter is a very accurate estimation of the position, hence a successful reconstruction of the vehicle's trajectory. As both sensors are setup to be error free, the only source of position error comes from the environment itself, where the LiDAR processing is tested to retrieve the correct displacements.

Due to the architecture of our engine, which is thought to work with real applications, the filter starts with a static alignment. Therefore, we simulate a static phase, where the car is not moving. Usually this phase is not only used for the initialization of the filter but also for the synchronization of the sensors' measurements.

We take the first GNSS measurement coming from the generated RINEX file as a valid and accurate position and we assume that the filter has already converged due to previous RTK fixed position updates. As depicted in Figure 16, this is followed by several seconds of IMU free-run (red dots) until the first LiDAR velocity update (blue dots) is available (we remove some of the acquired point clouds intentionally in order to have this behaviour).

The validity flag computed in the LiDAR module is used to accept or reject the measurement as well as the P_L, which additionally sets its accuracy for the LiDAR velocity filter update. As seen in the trajectory plot in Figure 16, not all the LiDAR updates were used. They correspond to unsuccessful registration of point clouds. Turns in the trajectory and featureless environments are situations that are particularly challenging in this LiDAR module.

In general, it is difficult to obtain accurate velocity estimates along the z-axis using this type of 3D-LiDAR, as it is challenging to detect features that can constraint movement in that axes. A LiDAR with a higher resolution in the vertical component may be able to overcome this problem. In our case, we have set the z velocity in the vehicle frame to zero and assumed to be very accurate.

The final consideration in the setup of our filter is to set the process noise values such as the filter accepts every incoming LiDAR velocity update.

As one can see, the trajectory was successfully recovered. In a future work, we will provide an in-depth analysis of the position error and filter states (biases, attitude estimation, position error, etc.).

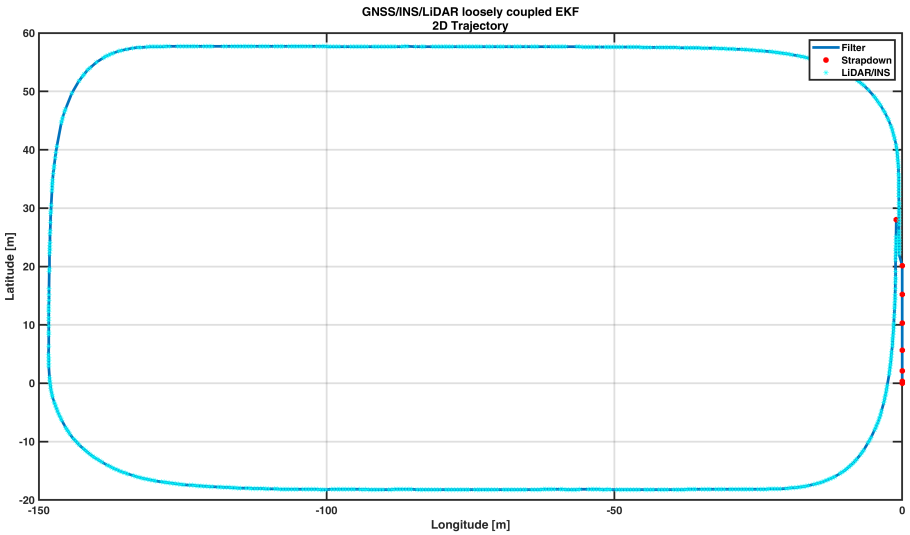

Figure 16: Sensor fusion through a loosely coupled Extended Kalman Filter

VI. CONCLUSIONS AND FUTURE WORK

In this work, we have introduced the new updates of the SIMSS-tool, which is now capable of integrating LiDAR data. Due to this new feature, modifications in the input of the IMU and RINEX simulators were done. We briefly described the new workflow and the modules that compose it. Specifically we focused in the process to generate IMU measurements and our approach to process the LiDAR point clouds. We have also successfully shown the applicability of the tool by designing a specific route in a simulated environment and processing the synthetic data generated by the Extended SIMSS-tool to process it in our navigation module within the MuSNAT. We obtained satisfactory results, which will be discussed in more detail in a future publication.

Nevertheless, many questions are still open. The trade-off between the quality of the IMU and the LiDAR observations, i.e. which IMU classes in which applications can provide the LiDAR sensor a reliable support. On the other hand, is also of our interest to test our own developed maps and create new according to the case scenarios we want to target (e.g. feature-rich/ featureless environments). Another open problem that should be also addressed in the future, is the time synchronisation issue, especially if working with Low-cost sensors, where the internal time latency/jitters is not known or if the time stamping source is not available for short time. In this case SIMMS-tool offer the opportunity to test the impact of this parameter, since we have the fully control on the timing source, i.e. GNSS and also the time stamping of each sensor. A well-know issue that could also be investigates, is the impact of the drop-off of the LiDAR point clouds on both position and velocity accuracy. Additionally, in order to reduce the computation load, while keeping the accuracy of the simulation tool chain, the processing of the point cloud at different sampling rates could be investigated. Finally, to explorer the the potential and the limitation of our tool chain, an advanced simulation with more realistic car behavior and population of the simulation with realistic urban traffic conditions can be adapted.

ACKNOWLEDGEMENTS

The results presented in this work were developed within the project “OPtimal Assistierte, hoch Automatisierte, Autonome und cooperative Fahrzeugnavigation und Lokalisation (short: OPA3L)”, funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) and administered by the Project Management Agency for Aeronautics Research of the German Space Agency (DLR) in Bonn, Germany (grant no. 50NA1910). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of UniBwM nor DLR.

REFERENCES

[1] A. Schütz, M. Bochkati, D. Maier, and T. Pany, "Closed-loop GNSS/INS simulation chain with RTK-accuracy for sensor fusion algorithm verification," in *Proceedings of the 33rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2020)*, 2020, pp. 2867–2877.

[2] (2017) CARLA open-source simulator for autonomous driving research. [Online]. Available: https://carla.org/

[3] A. Schütz, D. E. Sánchez-Morales, and T. Pany, "Precise positioning through a loosely-coupled sensor fusion of GNSS-RTK, INS and LiDAR for autonomous driving," in *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 2020, pp. 219–225.

[4] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.

[5] S. Bechtold and B. Höfle, "Helios: A multi-purpose LiDAR simulation framework for research, planning and training of laser scanning operations with airborne, ground-based mobile and stationary platforms." *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 3, no. 3, 2016.

[6] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[7] "Lidarmodul," 9/18/2021. [Online]. Available: https://www.dspace.com/de/gmb/home/products/sw/sensor_sim/lidar_module.cfm#175_48449

[8] Ondulus, "Ondulus LiDAR sensor simulation software," 2021. [Online]. Available: https://www.presagis.com/en/product/ondulus-lidar/

[9] S. Manivasagam, S. Wang, K. Wong, W. Zeng, M. Sazanovich, S. Tan, B. Yang, W.-C. Ma, and R. Urtasun, "LiDARsim: Realistic LiDAR Simulation by Leveraging the Real World," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 167–11 176.

[10] P. Vacek, O. Jašek, K. Zimmermann, and T. Svoboda, "Learning to predict LiDAR intensities," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[11] B. Hurl, K. Czarnecki, and S. Waslander, "Precise synthetic image and LiDAR (presil) dataset for autonomous vehicle perception," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 2522–2529.

[12] "RoadRunner, MATLAB." [Online]. Available: https://de.mathworks.com/products/roadrunner.html

[13] D. Titterton, J. Weston, J. Weston, I. of Electrical Engineers, A. I. of Aeronautics, and Astronautics, *Strapdown Inertial Navigation Technology*, ser. IEE Radar Series. Institution of Engineering and Technology, 2004.

[14] Y. Yang, "Tightly coupled MEMS INS/GPS integration with ins aided receiver tracking loops," Ph.D. dissertation, University of Calgary, 2008. [Online]. Available: https://prism.ucalgary.ca/handle/1880/103131

[15] N. El-Sheimy, "The postential of partial IMUs for land vehicle navigation," *InsideGNSS*, pp. 16–25, 2008.

[16] C. Jekeli, *Inertial Navigation Systems with Geodetic Applications*. De Gruyter, 2012. [Online]. Available: https://doi.org/10.1515/9783110800234

[17] "Numerical Integration. First Order. Lecture 13A." May 2012.

[18] M. Himmelsbach, F. V. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3D point clouds for ground vehicles," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 560–565.

[19] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart, "A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 1929–1934.

[20] A. Nguyen and B. Le, "3D point cloud segmentation: A survey," in *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*. IEEE, 2013, pp. 225–230.

[21] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3D laser scans for online operation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 163–169.

[22] F. Pomerleau, F. Colas, and R. Siegwart, "A review of point cloud registration algorithms for mobile robotics," *Foundations and Trends in Robotics*, vol. 4, no. 1, pp. 1–104, 2015.

[23] A. Shetty and G. X. Gao, "Adaptive covariance estimation of LiDAR-based positioning errors for UAVs," *NAVIGATION, Journal of the Institute of Navigation*, vol. 66, no. 2, pp. 463–476, 2019.

[24] T. Pany, D. Dötterböck, H. Gomez-Martinez, M. S. Hammed, F. Hörkner, T. Kraus, D. Maier, D. Sánchez-Morales, A. Schütz, P. Klima *et al.*, "The Multi-Sensor Navigation Analysis Tool (MuSNAT)–Architecture, LiDAR, GPU/CPU GNSS Signal Processing," in *Proceedings of the 32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2019)*, 2019, pp. 4087–4115.