Digital Analytics and Robotics for Sustainable Forestry

CL4-2021-DIGITAL-EMERGING-01
Grant agreement no: 101070405

DELIVERABLE 4.4

Semantic mapping system and traversability analysis for under canopy operation

Due date: Month 33 (May 2025)
Deliverable type: R
Lead beneficiary: UBONN

Dissemination Level: PUBLIC

1 Introduction

Robot navigation in forest environments can be very challenging, especially when dense undergrowth and uneven ground can significantly impede mobility. A 3D understanding of the world in the form of accurate maps plays a critical role in making navigation decisions and enabling safe and efficient navigation. Especially in unstructured forest environments, understanding the terrain is critical to inform the robot of which regions are traversable and which not. Only through this information, a robot can then make better informed decisions for autonomous missions. A highly accurate 3D map which encodes traversability information is invaluable for downstream tasks like path planning and exploration.

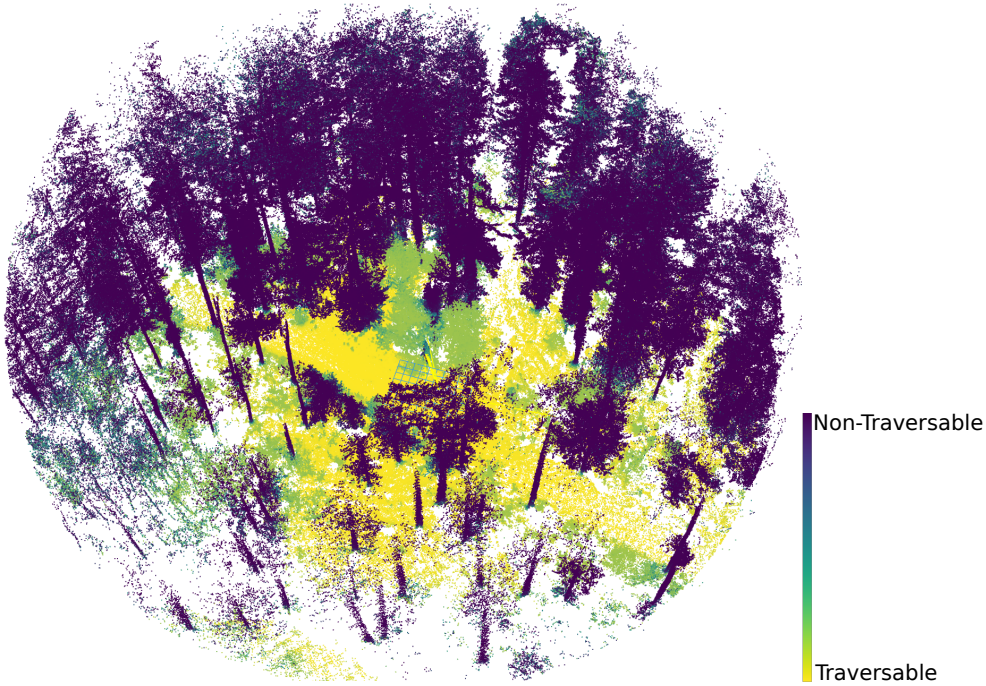

Figure 1: A traversability map produced by the system described in this deliverable for data collected with the Frontier sensor platform used in the DigiForest project. The traversability map of the local surroundings of the platform are colored in a gradient from yellow for traversable regions to dark blue for non-traversable regions.

Task T4.4 of the DigiForest project aims to develop a terrain analysis system that facilitates safe navigation for robots operating under canopy. This report summarizes the developments carried out for Task T4.4, for building a traversability-aware mapping system for under-canopy navigation. An example visualization of the output of this system is shown in Fig. 1. The system estimates a continuous traversability score per observed 3D point ranging from 0 (non-traversable and dark blue in the figure) to 1 (traversable and yellow in the figure). Aligning with the goals of the project, the outputs of this deliverable are then handed over to downstream tasks like Task T3.4 – Traversability-aware Navigation for Ground Robots and Task T3.5 – Mission Planning.

A schematic of the overall system architecture developed in this deliverable is shown in Fig. 2. The system takes IMU measurements and LiDAR scans from

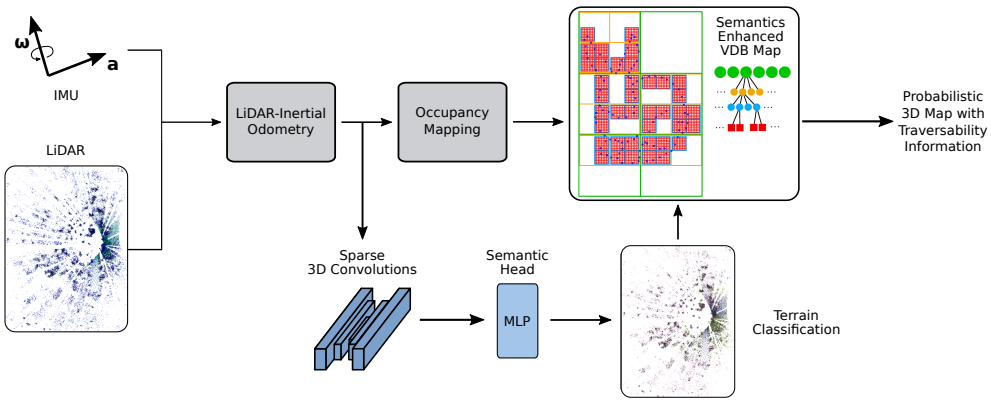

Figure 2: Overview of the traversability-aware mapping system. We use raw LiDAR scans and IMU data as input to a LiDAR-Inertial Odometry system. The LiDAR scans are processed by a MinkUNet-based deep learning model using sparse 3D convolutions to generate per-point traversability scores. The map which uses VDB as a data structure integrates the traversability scores in a probabilistic framework. The output of the system can be used for traversability-aware mission planning and navigation.

sensors onboard the robotic platform as input. Given the unstructured nature of forest scenes, LiDAR odometry methods face challenges both from the sensor noise itself and the sparse features (leaves, branches, and other thin vegetation) present in the environment. We therefore developed a robust LiDAR-Inertial odometry system, building upon our previous developed DigiForest LiDAR-only odometry system [1]. Using proprioceptive IMU measurements along with exteroceptive LiDAR measurements allows our odometry system to perform robust local scan registration and adapt to the complex motion profiles of the multiple robotic platforms used in the DigiForest project.

For mapping the environment, our mapping module performs probabilistic occupancy mapping given the 3D data and poses. Occupancy mapping is probabilistic and can handle sensor noise and to some degree also dynamics in the environment. Since operations like ray casting in occupancy mapping can be expensive, we employ a so-called VDB [2] data structure to ensure efficient performance. VDBs often demonstrates superior performances over other common map representations in robotics applications [3].

In parallel, we also process the LiDAR point clouds by a purpose-fit deep learning model for traversability classification. For this we exploit the Minkowski sparse 3D convolution-based [4] deep learning model developed in Task T4.3 (Semantic and Object Mapping) and Deliverable D4.3 (Semantic Segmentation System for Forest Environments). This model has demonstrated strong performance in panoptic segmentation in forest scenes [5]. Since in this task, we are concerned only with estimating traversability, the model architecture from D4.3 was adapted and fine-tuned to prioritize the semantic classes most relevant to traversability analysis and be efficient to execute. This fine-tuned model estimates per-point scores between 0 and 1 that reflect the traversability of the terrain. These scores are integrated into the VDB map, providing a spatially coherent representation of terrain traversability which can for example serve as input in Task T3.4 during robot mission planning. The deep learning architecture developed in this deliverable is also capable of real-time inference owing to its relatively compact model size.

In summary, for this deliverable we developed an efficient local environment mapping system capable of real-time operation that leverages LiDAR-Inertial odometry and a deep learning model to build a map representation encoding both geometric and traversability information. Furthermore, a traversability classified point cloud can be recovered from the map as a final output, allowing for seamless integration with downstream navigation and planning modules, supporting the requirements of T3.4 (Traversability-aware Navigation for Ground Robots) and T3.5 (Mission Planning).

2 LiDAR-Inertial Odometry

A robot needs a robust odometry system to accurately track it's position in an environment. This is especially challenging in forests, given the unstructured and repeating nature of the surroundings. The sparse features in the environment, from thin leaves, branches, etc., make it challenging for typical odometry systems to operate. The problem is further compounded if the motion of the robotic platform is also complicated, like the backpack mounted "Frontier" sensor package used in the DigiForest project. Our previous LiDAR odometry system [1] had space for improvement when deployed on data from the Frontier sensor in forest environments. LiDAR sensors are exteroceptive sensors and are therefore well suited to be paired with inertial measurement units (IMU) which are proprioceptive sensors that measure the motion of the platform. IMUs can also be cheap, especially when compared to LiDAR sensors. However, IMUs by themselves suffer from significant drift over time due to sensor noise and biases and are not suitable to be used alone. Hence, LiDAR-Inertial odometry combines both sensors and is a well known approach [6], [7]. LiDAR-Inertial odometry can handle complex motion dynamics, exploting the IMU, and has the potential for more accurate odometry than just LiDAR odometry.

2.1 IMU Motion Model

An inertial measurement unit typically combines a 3-axis accelerometer and 3-axis gyroscope. They may also include a magnetometer but in this deliverable we consider only IMUs with an accelerometer and a gyroscope. Let $\mathcal{S}$ denote the IMU sensor frame (attached to the physical unit), and $\mathcal{B}$ represent the robot's base link frame or base frame. The inertial frame $\mathcal{I}$ is a frame fixed to the center of the earth, see Fig. 3. The accelerometer measures proper acceleration $\mathbf{a}^{\mathcal{S}}_{\mathcal{SI}} \in \mathbb{R}^3$ – the specific force experienced by the sensor with associated frame $\mathcal{S}$ relative to the inertial frame $\mathcal{I}$. The superscript denotes the frame in which the vector is expressed. Under typical ground vehicle assumptions, we neglect the Earth's curvature and rotation (Coriolis and centrifugal accelerations) in the accelerometer measurement as they are insignificant compared to the typical sensor noise itself. Therefore, we consider the accelerometer to measure only the body's true kinematic acceleration $\mathbf{a}_*$ and the acceleration due to gravity $\mathbf{g}$ and that the IMU measurements are given with respect to the navigation frame $\mathcal{N}$ (see Fig. 3). We drop the subscripts denoting the measured frame ($\mathcal{S}$) and reference frame ($\mathcal{N}$) for notational clarity since they remain the same from here on. The measured acceleration is then given by

$$\mathbf{a}^{\mathcal{S}} = \mathbf{a}^{\mathcal{S}}_{*} - \mathbf{g}^{\mathcal{S}}, \tag{1}$$

where again the superscript denotes the frame in which the vector is expressed. The gyroscope measures angular velocity $\boldsymbol{\omega}^{\mathcal{S}}_{\mathcal{SI}} \in \mathbb{R}^3$ of the sensor frame relative to the inertial frame, expressed in $\mathcal{S}$ coordinates. Again, as we ignore Earth rotation effects,

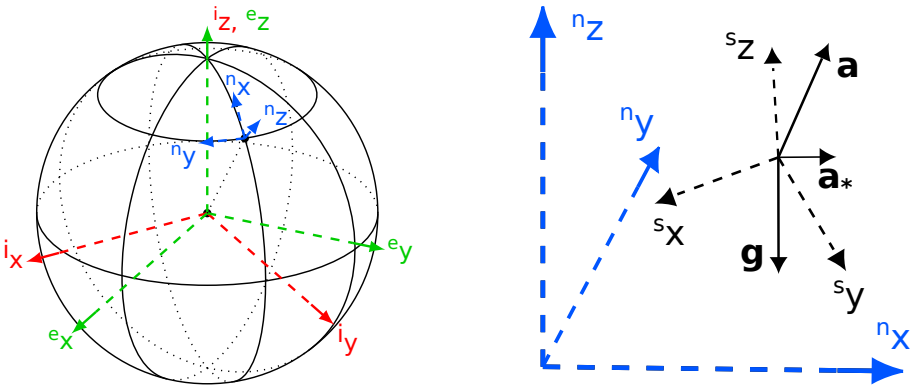

Figure 3: Coordinate frames relevant for IMU motion modeling. Superscripts for the basis vectors indicate the associated frame: i for the fixed inertial frame $\mathcal{I}$, e for the rotating Earth frame $\mathcal{E}$, and n for the navigation frame $\mathcal{N}$, which is used for odometry. Since we neglect the effects of Earth's rotation, on the right is a simplified system with only the navigation frame $\mathcal{N}$ and the sensor frame $\mathcal{S}$. $\mathbf{a}$ is the IMU-measured acceleration vector, $\mathbf{g}$ is gravity, and $\mathbf{a}_*$ is the body acceleration.

we approximate

$$\boldsymbol{\omega}^{\mathcal{S}}_{\mathcal{SI}} \approx \boldsymbol{\omega}^{\mathcal{S}}_{\mathcal{SN}} = \boldsymbol{\omega}^{\mathcal{S}}. \tag{2}$$

While the IMU measures acceleration and angular velocity in the sensor frame $\mathcal{S}$, for odometry we are generally interested in estimating the pose of the robot base link frame $\mathcal{B}$. Given the fixed extrinsic calibration between the sensor to base link frames, represented by the rigid body transform $\mathbf{T}^{\mathcal{B}}_{\mathcal{S}} \in \mathrm{SE}(3)$ as

$$\mathbf{T}^{\mathcal{B}}_{\mathcal{S}} = \begin{bmatrix} \mathbf{R}^{\mathcal{B}}_{\mathcal{S}} & \mathbf{t}^{\mathcal{B}}_{\mathcal{S}} \\ \mathbf{0}^{\top} & 1 \end{bmatrix}, \tag{3}$$

the angular velocity transformation is straightforward:

$$\boldsymbol{\omega}^{\mathcal{B}} = \mathbf{R}^{\mathcal{B}}_{\mathcal{S}} \boldsymbol{\omega}^{\mathcal{S}}. \tag{4}$$

While the acceleration transformation must account for the lever arm effect due to the sensor-body displacement $\mathbf{t}^{\mathcal{B}}_{\mathcal{S}} \in \mathbb{R}^3$:

$$\mathbf{a}^{\mathcal{B}} = \mathbf{R}^{\mathcal{B}}_{\mathcal{S}} \mathbf{a}^{\mathcal{S}} - \dot{\boldsymbol{\omega}}^{\mathcal{B}} \times \mathbf{t}^{\mathcal{B}}_{\mathcal{S}} - \boldsymbol{\omega}^{\mathcal{B}} \times \left(\boldsymbol{\omega}^{\mathcal{B}} \times \mathbf{t}^{\mathcal{B}}_{\mathcal{S}} \right). \tag{5}$$

Given the acceleration and angular velocity, the rigid body kinematics are given by

$$\begin{aligned} \dot{\mathbf{v}}^{\mathcal{B}} &= \mathbf{a}^{\mathcal{B}}_{*} \\ \dot{\mathbf{R}}^{\mathcal{N}}_{\mathcal{B}} &= \mathbf{R}^{\mathcal{N}}_{\mathcal{B}} [\boldsymbol{\omega}^{\mathcal{B}}]_{\times}, \end{aligned} \tag{6}$$

where $\mathcal{N}$ is the navigation frame, $[\cdot]_{\times}$ denotes the skew-symmetric matrix of a vector, and $\mathbf{v}^{\mathcal{B}}$ is the base frame velocity expressed in the local frame $\mathcal{B}$. For constant $\mathbf{a}^{\mathcal{B}}$ and $\boldsymbol{\omega}^{\mathcal{B}}$ over a time period Δt, the discrete-time motion model becomes:

$$\begin{aligned} \Delta \mathbf{p} &= \mathbf{v}^{\mathcal{B}}_{0} \Delta t + \frac{1}{2} \mathbf{a}^{\mathcal{B}}_{*} \Delta t^2 \\ \Delta \mathbf{R} &= \exp\left([\boldsymbol{\omega}^{\mathcal{B}} \Delta t]_{\times} \right), \end{aligned} \tag{7}$$

where $\exp(\cdot)$ denotes the SO(3) exponential map, and $\mathbf{v}_0^{\mathcal{B}}$ is the initial base frame velocity. The position of the body as a function of time t is given by $\mathbf{p}(t)$ and the position update by

$$\mathbf{p}^{\mathcal{B}}(\Delta t) = \mathbf{p}^{\mathcal{B}}(0) + \Delta\mathbf{p}. \tag{8}$$

Similarly, $\mathbf{R}_{\mathcal{B}}^{\mathcal{N}}(t)$ denotes the rotation of the body frame $\mathcal{B}$ with respect to the navigation frame $\mathcal{N}$, and the rotation update is

$$\mathbf{R}_{\mathcal{B}}^{\mathcal{N}}(\Delta t) = \mathbf{R}_{\mathcal{B}}^{\mathcal{N}}(0)\,\Delta\mathbf{R}. \tag{9}$$

2.2 LiDAR Scan Registration

Each LiDAR scan provides a time-stamped point cloud in the LiDAR sensor frame. We first transform all points to the robot base frame $\mathcal{B}$ using the known extrinsic calibration, and for the following steps, we assume the LiDAR sensor frame coincides with the base frame. The point cloud is then given by $\mathcal{P}^{\mathcal{B}} = \{(\mathbf{p}_i^{\mathcal{B}}, t_i) \mid \mathbf{p}_i^{\mathcal{B}} \in \mathbb{R}^3,\ t_i \in \mathbb{R}\}$, where $\mathbf{p}_i^{\mathcal{B}}$ represents a 3D range measurement at timestamp t_i.

We follow the general approach of KISS-ICP [1] to register each input LiDAR scan to a local map. Our odometry pipeline maintains a local point map $\mathcal{M} = \{\mathbf{m} \in \mathbb{R}^3\}$ which we use for registering the input scan, to obtain a pose estimate. We note here that this map is for estimating the odometry and is different from what we use later for occupancy and traversability mapping. First we apply sensor motion prediction and motion compensation, often called deskewing, to undo the distortions of the 3D data caused by the sensor's motion during scanning and its non-synchronous data aquisition. We then subsample the deskewed scan and estimate correspondences between it and the local map. Finally we register the input cloud to the local map using the correspondences to obtain the pose $\mathbf{T}_{\mathcal{B}}^{\mathcal{N}} \in \mathrm{SE}(3)$.

2.2.1 Deskewing the Point Cloud

Our LiDAR sensor typically operates at a measurement frequency of 10 Hz, whereas an IMU provides measurements at 100 Hz or higher. For the motion prediction of the LiDAR sensor during scanning, we can therefore leverage the IMU motion model from Eq. (7) to predict sensor poses at each point's acquisition time t_i. A common approach integrates each IMU measurement individually to maintain a motion prediction pose estimate [6]. However, following observations similar to Bloesch et al. [8], averaging the IMU measurements over the LiDAR scan period yields essentially the same results while reducing the computation required. Averaging the IMU measurements also reduces the effect of sensor noise.

Hence, for the LiDAR scan interval from $[t_0, t]$ where t_0 is the timestamp of the first measured point in the scan and t is the timestamp of the last measured point, we compute the average IMU acceleration and angular velocity, denoted by $\hat{\mathbf{a}}$ and $\hat{\boldsymbol{\omega}}$ respectively. Using these averages, the pose at any intermediate timestamp t_i within the scan period can be predicted by applying the discrete-time updates given in Eq. (8) and Eq. (9) using $\Delta t = t_i - t_0$. We deskew the scan to a reference time of t, essentially treating the LiDAR measurement as being obtained entirely at t. The range measurement $\mathbf{p}_i^{\mathcal{B}}(t_i)$ acquired at time t_i can be motion-compensated to the reference time t by

$$\mathbf{p}_i^{\mathcal{B}}(t) = \mathbf{T}(t, t_0)\,\mathbf{T}(t_0, t_i)\,\mathbf{p}_i^{\mathcal{B}}(t_i), \tag{10}$$

where $\mathbf{T}(t_1, t_2) \in \mathrm{SE}(3)$ denotes the transformation of the base frame $\mathcal{B}$ from time t_2 to time t_1. Specifically, $\mathbf{T}(t_0, t_i)$ transforms points from the sensor pose at time t_i to the pose at time t_0, and $\mathbf{T}(t, t_0)$ transforms points from the pose at t_0 to the reference

pose at time t. Both intermediate transformations can be obtained by integrating the IMU motion model Eq. (7) over the corresponding time intervals. This motion compensation effectively removes distortions caused by sensor motion during scan acquisition, improving the accuracy of subsequent scan registration.

2.2.2 Scan Alignment

We will first give a brief overview of the local map structure and correspondence estimation; our approach closely follows our previous DigiForest LiDAR odometry system [1] to which we refer to for further details. Our local map $\mathcal{M}$ is represented as a voxel grid with voxel size $v \times v \times v$, where each voxel stores a certain number of points. For every incoming LiDAR scan, we first downsample the deskewed point coud $\mathcal{P}^*$ to an intermediate point cloud $\mathcal{P}^*_{\text{merge}}$, which is later used to update the map when the relative motion of the robot has been estimated. We then further downsample $\mathcal{P}^*_{\text{merge}}$ to produce a set of points $\mathcal{Q}$ used for registration against the local map $\mathcal{M}$. Correspondences are estimated by associating each point in $\mathcal{Q}$ with points in $\mathcal{M}$.

Initially, for a scan at reference time t, we estimate the pose $\hat{\mathbf{T}}_{\mathcal{B}}^{\mathcal{N}}$ using the IMU motion model in Eq. (7). We then refine this pose estimate by using the point-to-point ICP algorithm. At each iteration, we transform the keypoints into the global (navigation) frame and compute the correspondence set $\mathcal{C} = \{(\mathbf{q}, \mathbf{m}) \mid \mathbf{q} \in \mathcal{Q}, \mathbf{m} \in \mathcal{M}\}$ using a memory-efficient nearest neighbor search with a fixed data association threshold. For a transformation estimate $\mathbf{T} = [\mathbf{R}\ \mathbf{t}] \in \mathrm{SE}(3)$, we define the residual between a map point $\mathbf{m}$ and a transformed source point $\mathbf{q}$ as

$$\mathbf{r}(\mathbf{T}) = \mathbf{T}\mathbf{q} - \mathbf{m}. \tag{11}$$

The point-to-point ICP cost function is then defined as the mean squared residual over the correspondence set $\mathcal{C}$:

$$\chi_{\text{icp}}(\mathbf{T}) = \frac{1}{|\mathcal{C}|} \sum_{(\mathbf{q},\mathbf{m})\in\mathcal{C}} \|\mathbf{r}(\mathbf{T})\|_2^2, \tag{12}$$

where $|\mathcal{C}|$ denotes the cardinality of the correspondence set. While this cost is sufficient for the ICP algorithm, we can further exploit IMU measurements to improve the pose estimation. Referring back to Eq. (1), the accelerometer measures both the body's true kinematic acceleration and the acceleration due to gravity. Neglecting sensor noise, if we assume the accelerometer primarily measures gravity expressed in the sensor frame, we can then estimate an orientation aligning the measured acceleration to the gravity in the global frame. Note, however, that the yaw is unobservable in this alignment. An additional cost term based on this orientation alignment can be defined as

$$\chi_{\text{ori}}(\mathbf{T}) = \left\|\mathbf{R}\hat{\mathbf{a}} + \mathbf{g}^{\mathcal{N}}\right\|_2^2. \tag{13}$$

While χ_{ori} is written as a function of $\mathbf{T}$, it depends only on the rotation $\mathbf{R}$. This cost essentially acts as an additional observation on the roll and pitch angles of the estimate. However, in this cost, we assume the measured acceleration is the gravity, effectively neglecting platform motion. Despite this approximation, prior work [9], [10] shows that it does not significantly degrade results for roll-pitch estimation. Nevertheless, we intend this cost only as a regularization on the roll-pitch estimation, and do not want to drive the optimization to minimize this cost over the ICP cost. This preference can be introduced into the optimization by adding a regularization term to the orientation cost, yielding the combined cost as

$$\chi(\mathbf{T}) = \chi_{\text{icp}}(\mathbf{T}) + \frac{1}{\beta}\chi_{\text{ori}}(\mathbf{T}), \tag{14}$$

where β is inversely proportional to the amount of regularization we want to impose. Following our recently developed Kinematic-ICP approach [11], designed to consider that robots often move on the ground, we set this value in a data-driven fashion based on the initial guess $\hat{\mathbf{T}}_{\mathcal{B}}^{\mathcal{N}}$ from the IMU motion model as:

$$\beta = \chi_{\text{icp}}(\hat{\mathbf{T}}_{\mathcal{B}}^{\mathcal{N}}). \tag{15}$$

We then minimize the total cost $\chi(\mathbf{T})$ in an iterative least squares fashion by solving

$$\begin{aligned} \delta\mathbf{x} &= \arg\min_{\delta\mathbf{x}} \chi(\mathbf{T} \boxplus \delta\mathbf{x}) \\ \mathbf{T} &\leftarrow \mathbf{T} \boxplus \delta\mathbf{x}, \end{aligned} \tag{16}$$

where $\delta\mathbf{x}$ is the correction vector and $\boxplus$ denotes the operator applying this correction to the current pose estimate. This process – including nearest neighbor correspondence search and least squares optimization – is repeated until convergence, resulting in a new pose estimate $\mathbf{T}_{\mathcal{B}}^{\mathcal{N}}$. After convergence, we update the map with $\mathcal{P}^{*}_{\text{merge}}$ transformed by the optimized pose.

3 Traversability Classification

Our odometry system estimates the robot pose through multiple optimization iterations for each input LiDAR scan. We use the resulting optimized poses to re-deskew the raw scans, providing more accurate motion compensation before integrating the data into the traversability map. To account for noisy range measurements close-to and far-away from the LiDAR sensor, we filter points that fall outside a specific minimum and maximum range. For deskewing, our odometry system provides the robot pose at the start and end of each LiDAR scan period. We interpolate between these poses in the $\mathfrak{se}(3)$ Lie algebra – different from and associated with the SE(3) Lie group – to estimate the motion for each point within the scan and follow the approach in Sec. 2.2.1. The resulting filtered and deskewed point cloud, denoted as $\mathcal{D}$, serves as the input for traversability classification and subsequent map integration.

The primary goal of this stage of the pipeline is to assign a traversability score to each point in $\mathcal{D}$, enabling downstream traversability-aware planning and navigation tasks. Our approach builds on prior work developed in Deliverable D4.3, a panoptic segmentation system for forestry environments. The system in D4.3 was designed to compute a semantic label for each LiDAR point and an instance label for each tree point, enabling panoptic segmentation of forest scenes. Specifically, it is capable of segmenting a scene for ground, shrub, and tree classes, segmentating tree instances, and also segment trees into hierarchical semantic classes of stem and canopy. The approach exploits a sparse voxel-based architecture using the the Minkowski engine [4], an auto-differentiation library for sparse tensors suitable for efficiently processing sparse 3D point clouds. The network architecture, based on a MinkUNet backbone [4], produces feature embeddings for each LiDAR point, which are then processed by semantic and instance heads for per-point semantic and instance classification. The training dataset for this system is presented in Deliverable D7.2.

In this deliverable, we aim to perform traversability classification instead of panoptic segmentation, with a particular emphasis on real-time, online inference. Since the key requirement for downstream traversability-aware navigation (Tasks T3.4, T3.5) is to distinguish between traversable and non-traversable regions, the architecture from D4.3 was simplified and fine-tuned for this binary classification task. Specifically, the instance head was removed, and the semantic head was modified to output a

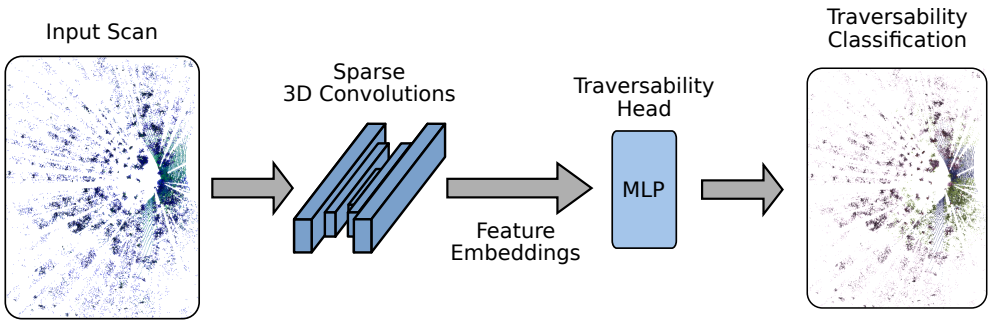

Figure 4: Architecture of the traversability classification network. The MinkUNet backbone extracts features from the input point cloud, which are processed by a multi layer perceptron (MLP) head for traversability classification.

traversability score between 0 and 1, rather than a semantic label. The model size was reduced from 785k to 206k trainable parameters, resulting in a more efficient architecture suitable for deployment on a robot. The revised architecture is illustrated in Fig. 4.

We train the network using the binary cross-entropy loss and use the AdamW [12] optimizer for training with a constant learning rate of 10^{-3} and a weight decay of 10^{-4}. Supervision for training was provided by a modified version of the dataset presented in Deliverable D7.2, which contains only panoptic labels but no traversability labels. To generate traversability supervision, semantic classes were mapped into a traversability label: tree semantics, including stem and canopy, were assigned a score of 0 (non-traversable), while ground and shrub were considered traversable. To distinguish between ground and shrub, ground points were assigned a score of 1.0 for fully traversable and shrub points were assigned a slightly lower traversability score of 0.8. In our evaluations, we observed that the network was able to learn this distinction in the score (see green points in Fig. 5). We note that the dataset in D7.2 provides labels and a pose trajectory for locally aggregated scans, but our system operates on raw scans. To bridge this gap, we used our LiDAR-Inertial odometry along with our previous DigiForest loop closure detection system [13] in a pose-graph optimization framework [14]. With this, we estimated improved poses for each raw scan and associated it with the closest corresponding aggregate cloud by global registration between the dataset trajectory and our estimated trajectory. This was followed by a fine ICP registration, enabling accurate label transfer from aggregated to raw scans. The model was first pre-trained on the D7.2 dataset and subsequently fine-tuned on the label-transferred raw scan data to ensure as little domain-gap as practical during deployment. The resulting model is capable of real-time inference and produces a per-point traversability score, visualized in Figure 5.

4 Probabilistic Mapping

After each incoming scan is classified for traversability, we integrate it into our traversability map. We maintain this map at a much finer resolution than we typically do for odometry, as this enables a more precise reconstruction of the environment that is relevant for mapping but not for odometry estimation. The input scans may contain spurious points due to sensor noise, as well as dynamic obstacles that should not be integrated into the map. Our goal is to maintain a static map for navigation, so we

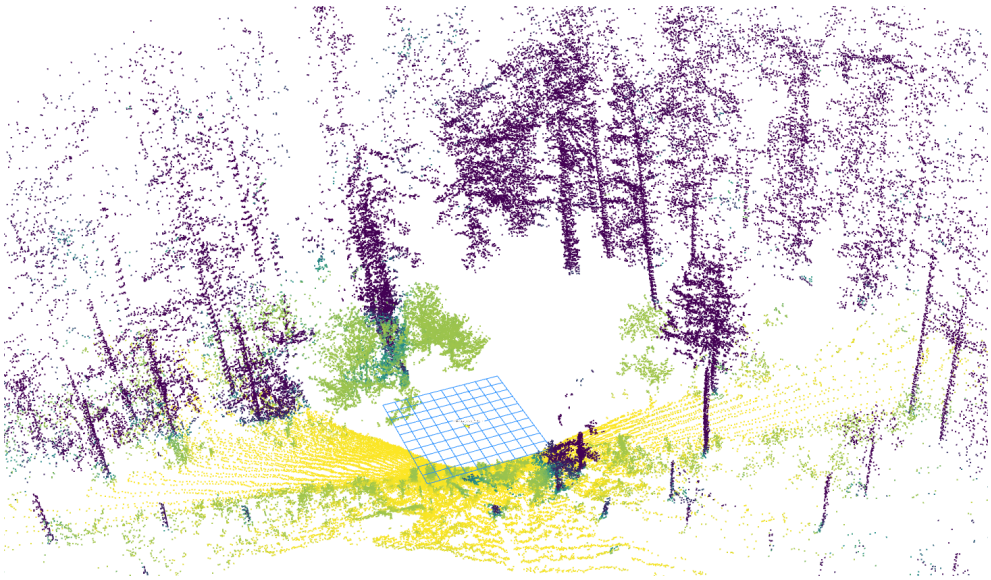

Figure 5: Output of the traversability classification network on a single scan. Points are colored according to a heatmap on the predicted traversability score, with higher values (yellow) indicating traversable regions and lower values (dark blue) indicating non-traversable regions.

employ probabilistic occupancy mapping to handle both sensor noise and dynamic objects. This involves computationally intensive operations like ray casting, so the map representation should lend itself to efficient algorithms. To this end, we use the VDB [2] data structue (see Fig. 2). The VDB representation models a virtually unbounded 3D voxel grid that allows for cache-coherent and fast data access into sparse volumes of high resolution. The VDB data structure is well suited to mapping tasks [3] and once the map is built, downstream components can efficiently query it for occupied voxels and the probability of that voxel being traversable.

4.1 Traversability Mapping

We first detail our approach to traversability mapping, as our system performs traversability classification on each incoming scan before integrating it into the map. This integration is performed independently of the occupancy state, allowing us to accumulate traversability information over time for each voxel.

Our goal is to fuse per-point traversability predictions into a probabilistic volumetric belief. Fusing multiple independent predictions over time can filter out prediction errors from the neural network [15]. For each deskewed scan at time t (see Sec. 3), we denote the set of N points as $\mathcal{D}_t = \{\mathbf{d}_{t,1}, \dots, \mathbf{d}_{t,N}\}$ with $\mathbf{d}_{t,j} \in \mathbb{R}^3$ for point j, and the corresponding set of predicted logits as $\mathcal{L}_t = \{\ell_{t,1}, \dots, \ell_{t,N}\}$ where $\ell_{t,j} \in \mathbb{R}$ denotes the traversability logit. We estimate the joint probability distribution of the traversability belief for all voxels in the map $\mathcal{M} = \{\mathbf{m}_i\}$, given all observations up to time t, by

$$p(\mathcal{M} \mid \mathcal{D}_{1:t}, \mathcal{L}_{1:t}) = \prod_i p(\mathbf{m}_i \mid \mathcal{D}_{1:t}, \mathcal{L}_{1:t}), \tag{17}$$

where $\mathcal{D}_{1:t}$ and $\mathcal{L}_{1:t}$ denote the sets of all measured points and predicted logits up to time t, respectively. According to the derivation of the static-state binary Bayes filter,

we obtain a recursive estimation scheme for each voxel:

$$l(\mathbf{m}_i \mid \mathcal{D}_{1:t}, \mathcal{L}_{1:t}) = l(\mathbf{m}_i \mid \mathcal{D}_{1:t-1}, \mathcal{L}_{1:t-1}) + l(\mathbf{m}_i \mid \mathcal{D}_t, \mathcal{L}_t) - l(\mathbf{m}_i), \tag{18}$$

where $l(x) = \log\left(\frac{p(x)}{1-p(x)}\right)$ denotes the log-odds, $l(\mathbf{m}_i \mid \mathcal{D}_{1:t-1}, \mathcal{L}_{1:t-1})$ is the current belief stored in the voxel, $l(\mathbf{m}_i \mid \mathcal{D}_t, \mathcal{L}_t)$ is the update from the current scan, and $l(\mathbf{m}_i)$ is the log-odds of the prior probability p_0. Since we do not assume prior knowledge about traversability, we set $p_0 = 0.5$. To compute the per-voxel update $l(\mathbf{m}_i \mid \mathcal{D}_t, \mathcal{L}_t)$, we take the arithmetic mean of the logits assigned to voxel i [15], following

$$l(\mathbf{m}_i \mid \mathcal{D}_t, \mathcal{L}_t) = \frac{1}{|\mathcal{V}_{t,i}|} \sum_{j \in \mathcal{V}_{t,i}} \ell_{t,j}, \tag{19}$$

where $\mathcal{V}_{t,i} = \{j \mid \mathbf{d}_{t,j} \in v_i\}$ is the set of point indices falling into voxel v_i at time t, and $|\mathcal{V}_{t,i}|$ its cardinality.

When querying the map for the traversability probability of a particular voxel v_i, the log-odds belief $l(\mathbf{m}_i \mid \mathcal{D}_{1:t}, \mathcal{L}_{1:t})$ can be converted to a posterior probability p using

$$p(x) = \frac{e^{l(x)}}{1 + e^{l(x)}}. \tag{20}$$

Our system can either return this probability, or threshold it to consider a voxel traversable if $p > 0.5$. We show examples of both in Fig. 7 and Fig. 8 respectively.

4.2 Occupancy Mapping

Similar to the traversability, we model a traditional occupancy state for each voxel as a log-odds value, representing the belief in whether the voxel is occupied or free. However, the difference here is that the occupancy log-odds are updated based on direct sensor observations rather than network predictions. Furthermore, there can be objects that robots can drive over, i.e., occupancy is not equal to non-traversability. For occupancy mapping, each incoming point increases the occupancy log-odds of the corresponding voxel by a predefined amount. To model free space, we employ ray casting from the sensor origin to each detected point using an efficient 3D Bresenham line algorithm. Voxels traversed by the ray (but not containing a detected point) are updated as free, reducing their occupancy log-odds. This process, called free space carving, is essential for handling sensor noise and dynamic obstacles. Each voxel maintains its log-odds value using a fixed-point integer representation for memory efficiency and computational speed. We also clamp the log-odds values to predefined minimum and maximum bounds to avoid numerical instability and overconfidence.

When querying the map for occupied voxels and their traversability probability, we first collect occupied voxels using Eq. (20) with the occupancy log-odds, thresholding voxels with probability greater than 0.5 as occupied. We then recover the posterior traversability probability for those occupied voxels using again Eq. (20) with the traversability log-odds.

5 Evaluation

We evaluate our system across three components: odometry, traversability classification, and the full system. We report quantitative results for the odometry and for the classification modules. We show qualitative results for demonstrating the full mapping pipeline on data from the Deliverable D7.2 and field data from the SAHA harverster platform.

Sequence	RPE (%)	ATE (m)
M1	0.62	0.08
M2	0.70	0.74
M3	0.88	0.10
M4	1.11	0.13
C1	0.81	0.09
D2	0.72	0.08

Table 1: Quantitative results of our odometry system, without loop closures or pose-graph optimization, compared to VILENS SLAM on the dataset from Deliverable D7.2 (March 2023 session). We report the RPE in [%] and the ATE in [m].

5.1 Odometry

We evaluate our odometry module on two datasets: quantitatively on the D7.2 dataset and qualitatively on field data from the SAHA harverster. The DigiForest D7.2 dataset includes reference trajectories for each sequence generated by the VILENS SLAM system [7]. To assess odometry accuracy, we use two widely adopted metrics. The first is the relative pose error (RPE), which measures the average translational error between estimated and reference trajectories over various segment lengths, reported as a percentage. This approach is standard in benchmarks such as KITTI [16], though we use shorter intervals of 1, 2, 5, 10, 20, 50, and 100 meters to better match the scale of our dataset collected in a forest following similar work [11]. The second metric is the absolute trajectory error (ATE) after alignment, which provides a measure of global drift in the estimated trajectories.

VILENS is a multi-modal SLAM system which integrates loop closures in a pose-graph optimization framework. Our system, however, is LiDAR-Inertial odometry only, without loop closures or pose-graph optimization. We compare our odometry system against a SLAM system, given the unavailability of ground truth, to evaluate the consistency of our odometry estimation. Tab. 1 presents results for the March 2023 recording sessions from the D7.2 dataset. As shown, assuming the VILENS SLAM results to be ground truth, our odometry results are only slightly worse in both RPE and ATE. The average RPE across the six sequences is 0.8% and the average ATE is just 20.33 cm. The poses from our odometry system are used for motion-compensating the raw LiDAR scans (see Sec. 3), and accurate motion-compensation is critical for good mapping quality when classifying and integrating the scans in subsequent parts of our pipeline.

We present also qualitative results on field data collected from the SAHA harvester platform. Fig. 6 illustrates the mapping consistency achieved by our odometry module on this platform.

5.2 Traversability Classification

In the absence of ground truth labels for traversability, we evaluate our approach indirectly by performing semantic segmentation on the Deliverable D7.2 dataset. We adapt the deep learning architecture described in Sec. 3 by modifying only the output layer in the traversability head (see Fig. 4) to predict four semantic classes: ground, shrub, stem and canopy. The network is trained with cross-entropy loss using the semantic labels from the D7.2 dataset for supervision. We follow the same train, validation, and test splits as provided in D7.2, and report results on the test split.

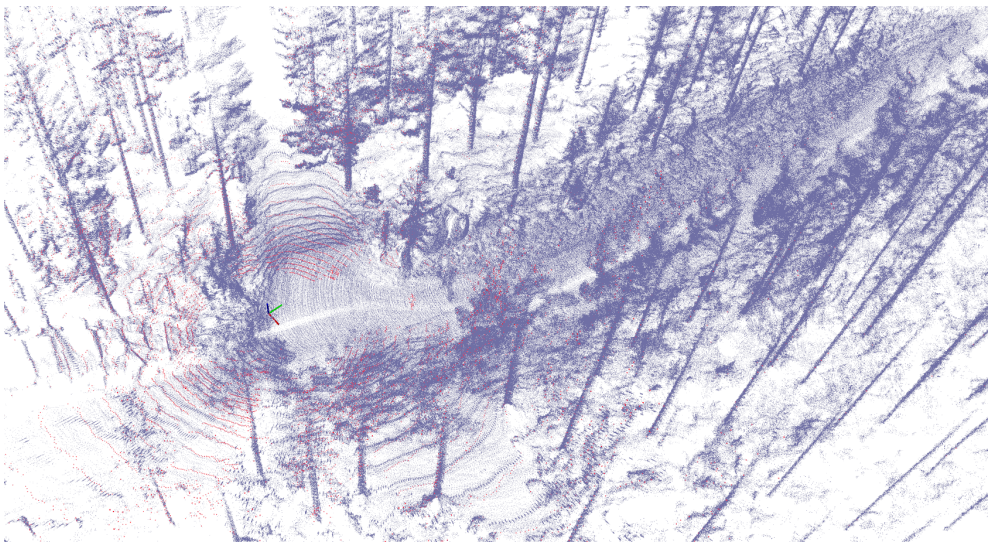

Figure 6: Odometry results on data from the SAHA platform. In dark blue is a map built from accumulating past scans transformed by the estimated odometry poses. The 3D axes denote the location of the robot, and in red is the current input scan.

Method	Param	Ground	Shrub	Stem	Canopy	mIoU
D4.3 (Pan. Seg.)	785K	79.5	72.7	80.8	48.2	70.3
Ours (Sem. Seg.)	206K	83.0	78.2	80.5	66.9	77.1

Table 2: Semantic segmentation performance (IoU) on Deliverable D7.2 dataset's test set. The system from Deliverable D4.3 is capable of performing panoptic segmentation. "Ours" is a simplified architecture from D4.3, targeting only semantic segmentation. "Param" is the number of learnable parameters in the deep learning model.

As shown in Tab. 2, our lightweight architecture (206K parameters vs. 785K) achieves a mean IoU of 77.1%, improving over the 70.3% for the D4.3 system. The D4.3 system was designed for panoptic segmentation, which is a combination of two tasks: semantic segmentation and instance segmentation. However, in this deliverable, we aim to perform traversability classification instead of panoptic segmentation so we additionally adapted the D4.3 system and reduced the model size. We suppose the improvement in results in Tab. 2 is due to the removal of the instance segmentataion head and simplification of the task: semantic segmentation is easier than panoptic segmentation [17]. Similar observations have been reported previously [18], [19], where multi-task learning or more complex objectives can sometimes degrade performance on individual tasks. Additionally, we note that our actual task of traversability classification is still simpler than multi-class semantic segmentation. The model only needs to regress a traversability likelihood rather than classify points into four distinct semantics. We demonstrate qualitative results of this in the following section.

5.3 Probabilistic Traversability Mapping

Figure 7 presents qualitative results of our full system on the Deliverable D7.2 dataset. As described in Sec. 4.1, our system can produce a continuous traversability probability

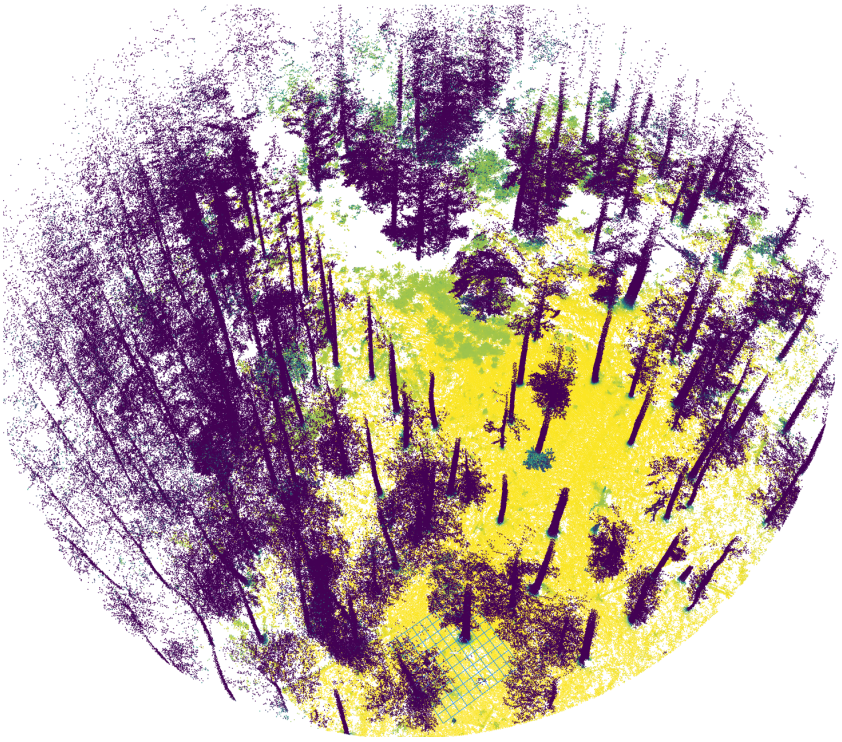

Figure 7: Traversability mapping results on sequence M4 from Deliverable D7.2 data. We visualize the traversability probability for each voxel, without thresholding, as a heatmap.

for each voxel, ranging from 0 (non-traversable) to 1 (traversable). Fig. 7 shows the traversability probability of the voxels as a heatmap, where dark blue represents non-traversable regions such as trees, and yellow indicates traversable ground regions. Our deep learning model is able to learn to distinguish between ground and shrub, which is reflected in the heatmap as intermediate colors for shrubs, indicating lower traversability compared to ground.

We show qualitative results on data from the SAHA platform in Fig. 8. Here, we threshold the predicted traversability probabilities, visualizing the map as either traversable or non-traversable voxels. Although occasional misclassifications occur at the point level due to the domain gap, our probabilistic mapping framework helps to filter out these errors over time, resulting in a more consistent and reliable traversability map. Overall, our full system can run faster than the LiDAR frame rate when tested on a laptop with an Intel Core i9-13950HX CPU and an Nvidia RTX 3500 GPU.

Full integration and testing of the traversability mapping system on the SAHA platform, as part of its navigation pipeline, will be conducted during the upcoming forest field trials in June 2025. The results will be reported in upcoming periodic reports and in Deliverable D1.3 (Final periodic review report 2).

6 Conclusion

In this deliverable, we presented a robust, efficient, and platform-agnostic traversability-aware online mapping system developed for Task T4.4 of the DigiForest project. The

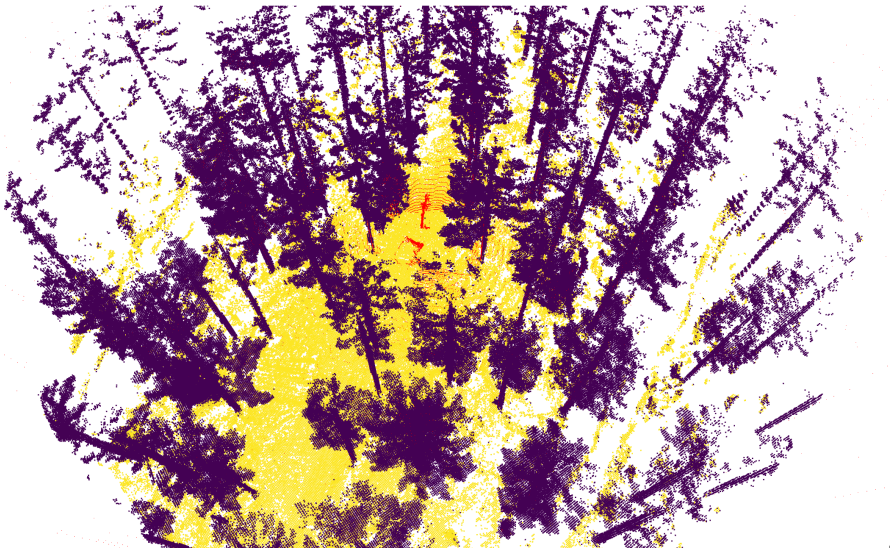

Figure 8: Traversability mapping results on data from the SAHA harvester platform. We threshold the traversability probability, such that voxels with probability > 0.5 are traversable (yellow) and otherwise are non-traversable (dark blue). In red, we visualize the raw LiDAR scan from the sensor, which shows the location of the harvester.

system leverages LiDAR-inertial odometry and deep learning-based traversability classification to provide high-resolution maps encoding both geometric and traversability information. The outputs of this deliverable can be used in downstream tasks like planning and navigation, supporting the requirements of Tasks T3.4 (Traversability-aware Navigation for Ground Robots) and T3.5 (Mission Planning).

References

[1] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 2, pp. 1029–1036, 2023.

[2] K. Museth, J. Lait, J. Johanson, J. Budsberg, R. Henderson, M. Alden, P. Cucka, D. Hill, and A. Pearce, "OpenVDB: An Open-source Data Structure and Toolkit for High-resolution Volumes," in *ACM SIGGRAPH 2013 courses*, 2013.

[3] I. Vizzo, T. Guadagnino, J. Behley, and C. Stachniss, "VDBFusion: Flexible and Efficient TSDF Integration of Range Sensor Data," *Sensors*, vol. 22, no. 3, p. 1296, 2022.

[4] C. Choy, J. Gwak, and S. Savarese, "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[5] M. Malladi, N. Chebrolu, I. Scacchetti, L. Lobefaro, T. Guadagnino, B. Casseau, H. Oh, L. Freissmuth, M. Karppinen, J. Schweier, S. Leutenegger, J. Behley, C. Stachniss, and M. Fallon, "DigiForests: A Longitudinal LiDAR Dataset for Forestry Robotics," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2025.

[6] W. Xu and F. Zhang, "FAST-LIO: A Fast, Robust LiDAR-Inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, pp. 3317–3324, 2021.

[7] D. Wisth, M. Camurri, and M. Fallon, "VILENS: Visual, Inertial, Lidar, and Leg Odometry for All-Terrain Legged Robots," *IEEE Trans. on Robotics (TRO)*, vol. 39, no. 1, pp. 309–326, 2023.

[8] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *Intl. Journal of Robotics Research (IJRR)*, vol. 36, no. 10, pp. 1053–1072, 2017.

[9] J. K. Lee, E. J. Park, and S. N. Robinovitch, "Estimation of attitude and external acceleration using inertial sensor measurement during various dynamic conditions," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 8, pp. 2262–2273, 2012.

[10] H. Luinge and P. H. Veltink, "Measuring orientation of human body segments using miniature gyroscopes and accelerometers," *Medical and Biological Engineering and Computing*, vol. 43, no. 2, pp. 273–282, 2005.

[11] T. Guadagnino, B. Mersch, I. Vizzo, S. Gupta, M. Malladi, L. Lobefaro, G. Doisy, and C. Stachniss, "Kinematic-ICP: Enhancing LiDAR Odometry with Kinematic Constraints for Wheeled Mobile Robots Moving on Planar Surfaces," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2024.

[12] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2019.

[13] S. Gupta, T. Guadagnino, B. Mersch, I. Vizzo, and C. Stachniss, "Effectively Detecting Loop Closures using Point Cloud Density Maps," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2024.

[14] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011, pp. 3607–3613.

[15] B. Mersch, T. Guadagnino, X. Chen, I. Vizzo, J. Behley, and C. Stachniss, "Building Volumetric Beliefs for Dynamic Environments Exploiting Map-Based Moving Object Segmentation," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 8, pp. 5180–5187, 2023.

[16] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[17] A. Kirillov, K. He, R. Girshick, C. Rother, and P. D. r, "Panoptic Segmentation," in *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[18] C. Fifty, E. Amid, Z. Zhao, T. Yu, R. Anil, and C. Finn, "Efficiently identifying task groupings for multi-task learning," in *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

[19] T. Standley, A. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, "Which tasks should be learned together in multi-task learning?" In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2020.