# Tightly-Coupled LiDAR-IMU-Wheel Odometry with an Online Neural Kinematic Model Learning via Factor Graph Optimization⋆

Taku Okawara^{a,*}, Kenji Koide^{b}, Shuji Oishi^{b}, Masashi Yokozuka^{b}, Atsuhiko Banno^{b}, Kentaro Uno^{a} and Kazuya Yoshida^{a}

^{a}The Space Robotics Lab. in the Department of Aerospace Engineering, Graduate School of Engineering, Tohoku University, Sendai, Miyagi, Japan
^{b}National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Ibaraki, Japan

## ARTICLE INFO

## ABSTRACT

Environments lacking geometric features (e.g., tunnels and long straight corridors) are challenging for LiDAR-based odometry algorithms because LiDAR point clouds degenerate in such environments. For wheeled robots, a wheel kinematic model (i.e., wheel odometry) can improve the reliability of the odometry estimation. However, the kinematic model suffers from complex motions (e.g., wheel slippage, lateral movement) in the case of skid-steering robots particularly because this robot model rotates by skidding its wheels. Furthermore, these errors change nonlinearly when the wheel slippage is large (e.g., drifting) and are subject to terrain-dependent parameters. To simultaneously tackle point cloud degeneration and the kinematic model errors, we developed a LiDAR-IMU-wheel odometry algorithm incorporating online training of a neural network that learns the kinematic model of wheeled robots with nonlinearity. We propose to train the neural network online on a factor graph along with robot states, allowing the learning-based kinematic model to adapt to the current terrain condition. The proposed method jointly solves online training of the neural network and LiDAR-IMU-wheel odometry on a unified factor graph to retain the consistency of all those constraints. Through experiments, we first verified that the proposed network adapted to a changing environment, resulting in an accurate odometry estimation across different environments. We then confirmed that the proposed odometry estimation algorithm was robust against point cloud degeneration and nonlinearity (e.g., large wheel slippage by drifting) of the kinematic model.

## 1. Introduction

Accurate and robust odometry estimation is crucial for autonomous robots to accomplish reliable navigation. State-of-the-art odometry algorithms based on Light Detection and Ranging–Inertial Measurement Unit (LiDAR-IMU) odometry algorithms [34, 25, 30] accurately estimate the robot pose owing to the tight fusion of LiDAR and IMU data. The IMU-based constraint can make the odometry estimation more robust against rapid motion and short-term point cloud degeneration. However, it is still difficult for LiDAR-IMU odometry algorithms to overcome long-term point cloud degeneration in featureless environments (e.g., tunnels and long corridors). IMU-based constraints cannot avoid an accumulation of errors that turn into estimation drift and corruption in such challenging environments.

A wheel encoder can provide a reliable constraint in such environments where LiDAR point clouds degenerate because the wheel odometry estimation can provide accurate motion prediction compared with an IMU owing to integral errors. While double integration of the acceleration is needed for calculating translational displacement in the case of IMU-based odometry estimation, wheel encoders measure wheel angular velocities and need only a single integration that accumulates errors more slowly. Therefore,

LiDAR-IMU-wheel odometry has the potential to overcome long-term LiDAR point cloud degeneration. Although the differential drive model is often used as a kinematic model of wheeled robots due to its simplicity, this model ignores both lateral movement and wheel slippage; thus, such complex motions cannot be accurately expressed with this simple model. Incorporating both lateral movement and wheel slippage can reduce the drift; in the case of skid-steering robots particularly, lateral movement and wheel slippage have a large impact on the estimation accuracy [1] because this robot model rotates by skidding its wheels based on different angular velocities of the left and right wheels. Furthermore, wheel slippage depends on terrain conditions, and thus the wheel odometry model must be maintained online to adapt to each environment. Therefore, online calibration of the complex kinematic model is crucial for creating reliable wheel odometry-based constraints.

To estimate an accurate kinematic model of a skid-steering robot, our previous work [21] conducted online calibration of the kinematic model based on the *full linear model* [1]. Specifically, this work jointly solved the online calibration problem and tightly coupled LiDAR-IMU-wheel odometry such that the calibrated model-based robot motion, LiDAR-based motion, and IMU-based motion become consistent. However, this linear model cannot express nonlinearity in the robot model (e.g., large wheel slippage during high-speed operation [3, 28]), because the robot motion model is regarded as linear. A natural approach to overcome the errors caused by non-linearity is to introduce a robot motion model based on a neural network. However, while the rigorous

*Corresponding author.
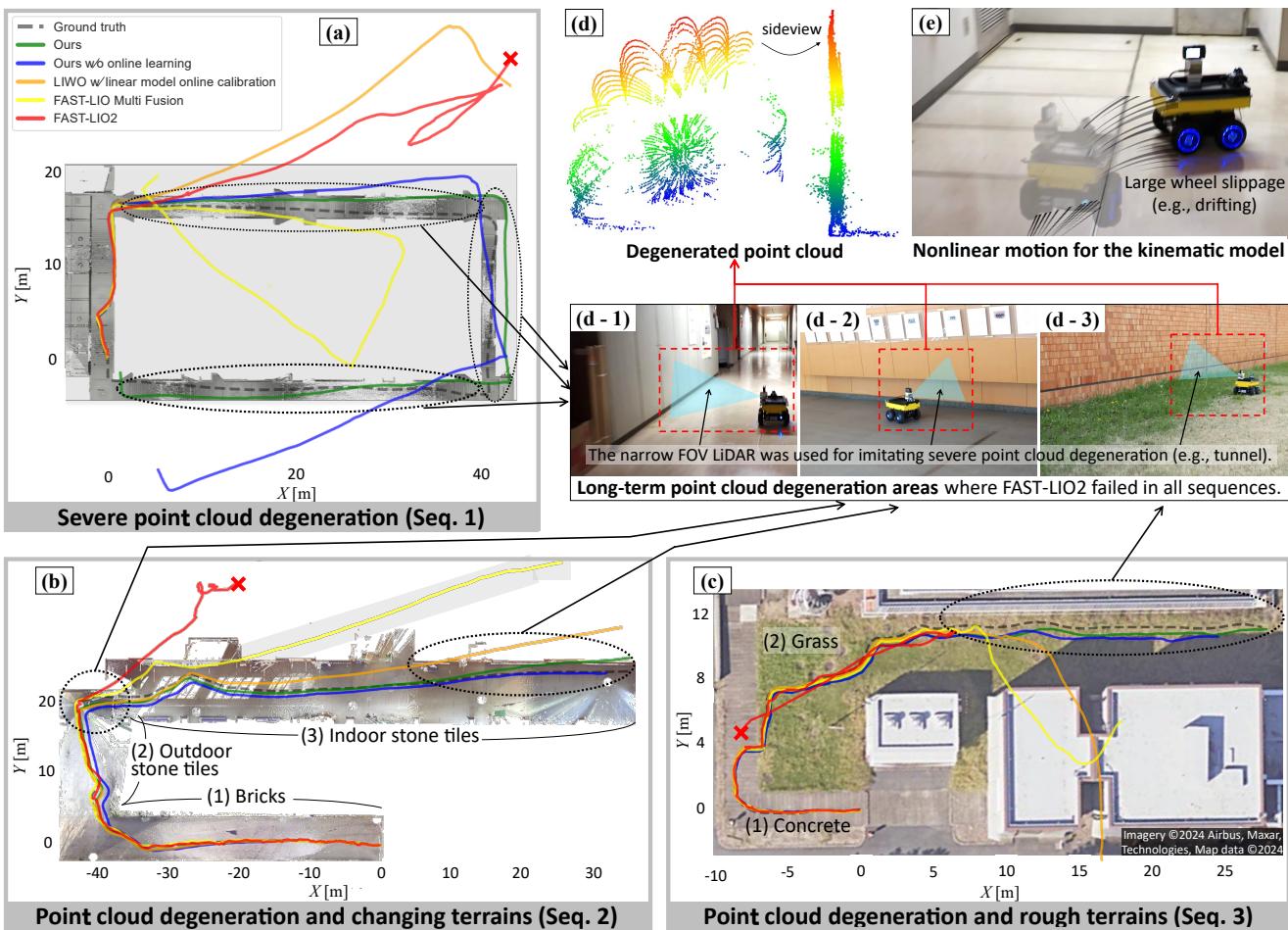✉ okawara.taku.t3@dc.tohoku.ac.jp (T. Okawara)
ORCID(s):

**Figure 1:** Odometry estimation results: (a), (b), and (c) are trajectory comparison results for Seq. 1, Seq. 2, and Seq. 3, respectively. (See Section 4.2 for additional discussion on the evaluation sequences.) The dotted black ellipses indicate areas where point clouds degenerate. The narrow FOV LiDAR was pointed at straight walls in all sequences to validate the robustness to severe point cloud degeneration. (d) Degenerated point clouds were observed when the LiDAR was pointed only at straight walls (d-1, d-2, and d-3). (e) Seq. 1 included complex robot motions involving large wheel slippage (e.g., drifting) in long straight corridors (d-1) where LiDAR point clouds degenerate. In Seq. 2, the robot transited three types of flat hard terrain and faced point cloud degeneration two times (d-2). Furthermore, in the case of Seq. 3, the robot transited from concrete to grass (i.e., rough and soft terrain) and also faced point cloud degeneration on the grass (d-3). The aerial photograph in (c) was obtained by Google Maps.

wheel odometry model depends on terrain-dependent parameters (e.g., friction coefficients and slip ratio), an offline trained network cannot express these parameters to adapt to changes in terrain conditions [22].

In this study, we propose a tightly coupled LiDAR-IMU-wheel odometry algorithm that incorporates online training of the neural network inferencing the kinematic model of a skid-steering robot. Through online training, the proposed network can adapt to the current terrain condition while achieving accurate motion prediction owing to its nonlinearity expression capability. We jointly solve the online training of the network and LiDAR-IMU-wheel odometry on a unified factor graph in a tightly coupled way.

In this work, we focused on applying the proposed method to a skid-steering robot because identifying the complex model of this robot type is difficult and valuable; however, the proposed method can be applied to other robot

platforms to accurately capture the nonlinearity of each kinematic model.

We summarize the contributions of this work as follows:

1. To deal with severe conditions, such as point cloud degeneration and nonlinear kinematic behavior of a skid-steering robot, we proposed a tightly coupled LiDAR-IMU-wheel odometry algorithm incorporating online training of the neural network to describe the kinematic model of a skid-steering robot. The neural network can implicitly express the nonlinearity of the kinematic model; thus, this learning-based motion constraint can compliantly adapt to the current terrain condition.

2. To strike a balance between the online training speed and estimation accuracy, we designed the network to incorporate both an *online learning model* and *offline*

*learning model*. While the *offline learning model* expresses fixed features (i.e., terrain-independent terms), the *online learning model* describes dynamically changing features (i.e., terrain-dependent terms).

3. We proposed *neural adaptive odometry factor* to directly optimize the *online learning model* of the network on a factor graph such that the online learning-based motion constraint and LiDAR-IMU-wheel odometry become consistent.

## 2. Related work

### 2.1. LiDAR-IMU odometry

The state-of-the-art LiDAR-IMU odometry algorithms enable accurate odometry estimation owing to a tight coupling of the LiDAR and IMU constraints. While a loose coupling approach conducts optimization based on poses estimated by measurements from each sensor (e.g., LiDAR and IMU), the tight coupling approach processes each sensor's raw measurement data to optimize unified constraints defined by each sensor's data. This approach can estimate odometry accurately and robustly because the data from both sensors are directly fused.

LINS [25] was the first tightly coupled LIDAR-IMU odometry algorithm with an iterated error-state Kalman filter. This approach fused the IMU-based motion model and point cloud matching to update states iteratively. LINS iteratively updates point correspondences and the sensor state until the state converges. FAST-LIO2 [34] also uses an iterated Kalman filter for a tight fusion of LiDAR and IMU data. Specifically, this work directly registered raw points of a new scan to a large local map (e.g., 1km) due to the ikd-tree, which efficiently represents a dense and large map because of efficient nearest search and incremental updates of the structure. This registration method and the ikd-tree enabled fast, robust, and accurate odometry estimation, and thus FAST-LIO2 exhibited significantly higher accuracy compared to LINS.

Tightly coupled LiDAR-IMU odometry can estimate accurate robot poses in feature-rich environments where point cloud matching works well. However, point cloud-based constraints become corrupted in featureless environments where point clouds degenerate for long periods (e.g., tunnels). IMU-based constraints also face accumulation errors related to integration errors (particularly for accelerometers). Therefore, a challenge of LiDAR-IMU odometry is robustness to point cloud degeneration.

### 2.2. Learning-based odometry using proprioceptive sensors

To improve robustness in the featureless environments described in Section 2.1, proprioceptive sensors (e.g., wheel encoders and IMUs) are commonly used along with LiDARs because these sensors' data are independent of the geometric features of environments. We can simply calculate relative displacement by integrating a kinematic model formulated by the proprioceptive sensor data. However, this integration

suffers from kinematic model error, bias and noise of the sensor measurements, and thus the odometry estimation has a large drift. Therefore, many researchers have proposed a learning-based approach to implicitly express an accurate kinematic model for reducing the drift.

IONet [5] demonstrated an accurate three-Degrees-of-Freedom (3DoF) odometry estimation due to time-dependent features of IMU measurements and body motions based on the Long Short-Term Memory (LSTM) network. IONet outperformed the traditional model-based method as in strap-down inertial navigation systems (SINS) [29]. In addition, TLIO [16] accurately estimated sensor inertial states, including IMU pose, velocity, and bias, with only IMU measurements through a tight coupling of a neural network-based 3D displacement estimation and a model-based kinematic model with a Kalman filter.

Onyekpe et.al. proposed a Recurrent Neural Network (RNN)-based wheel odometry model named WhONet [22]. WhONet tackled challenges [23] of wheel odometry estimation such as wheel slippage, sharp cornering, and acceleration changes due to the RNN's ability to capture relationships within sequential sensor data. This work suggested that the difficulty stems from a model adaptation to changes in terrain conditions and robot platform because WhONet was trained offline.

The kinematic model of skid-steering robots heavily depends on terrain-dependent parameters (e.g., friction coefficients and slip ratio), and thus online learning is desirable for obtaining an accurate kinematic model. Navone et.al. conducted online training for the kinematic model of a skid-steering robot [20] with IMU and wheel encoders. This work showed that the online learning-based odometry estimation outperformed the traditional extended Kalman filter approach because complex nonlinearity is implicitly expressed. This online learning model had almost the same accuracy as the batch-learning model when the robot moved over the same terrain. Furthermore, Ordonez et.al. trained online the kinematic and dynamic model of a skid-steering robot for expressing terrain-dependent terms [24]. This work validated accurate wheel odometry estimation even in different terrain conditions owing to online learning, which enables the model to adapt to each environment. These approaches only conducted online learning of the kinematic model based on reference data (e.g., visual odometry) obtained by exteroceptive sensors. Therefore, these approaches did not fuse the online trained kinematic model and exteroceptive sensors.

### 2.3. Model-based kinematic model for skid-steering robots

As seen in Figure 2, a skid-steering robot does not have any steering mechanism, and thus rotational motion is produced by skidding its wheels based on different angular velocities between the left and right wheels. This robot model can be used in various fields due to its high road ability and mechanical simplicity. However, this mechanical simplicity leads to the difficulty of modeling the kinematics
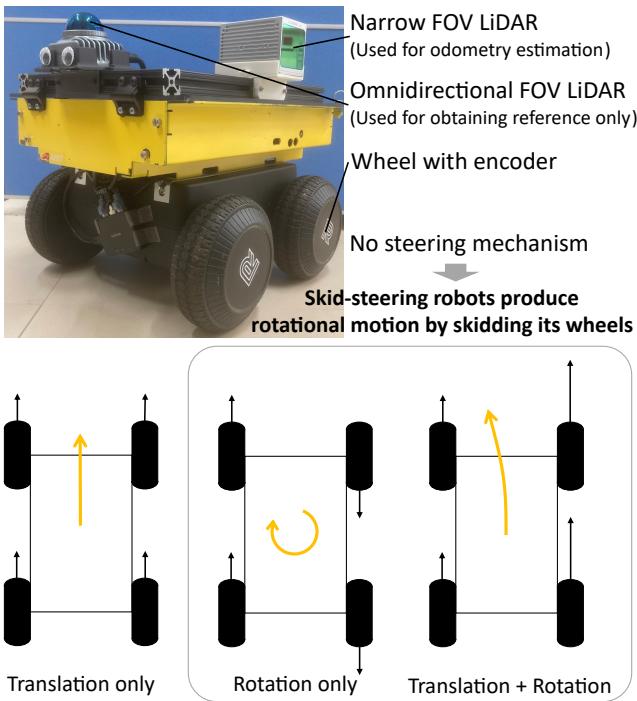
**Figure 2:** Skid-steering robot used as a testbed and diagram of motion of this robot model. Skid-steering robots produce rotational motion by skidding their wheels based on different angular velocities of the left and right wheels, instead of being equipped with a steering mechanism. We must consider the wheel slippage and lateral motion to obtain an accurate kinematic model of a skid-steering robot.

accurately because wheel slippage and lateral motions must be considered in the model. Due to the complexity of the wheel odometry model, various modeling approaches have been proposed [1, 14, 18, 33, 27].

The full linear model [1] is a generic model in that the velocity of the robot is assumed to be proportional to the angular velocity of the wheels. This model does not require explicitly expressing the kinematic model, which is different from the following approaches [18, 33]. Mandow et.al. proposed an extended differential drive model that introduced Instantaneous Centers of Rotation (ICR) parameters to better model the motion of skid-steering robots by extending the ideal differential drive model [18]. Wang et.al. also introduced a Radius Of Curvature (ROC)-based model that captures the relationship between the ICR parameters and ROC of the robot motion [33].

Baril et.al. evaluated the above kinematic models in different environments, such as a flat plane and uneven terrain. This work showed that the full linear model [1] is the most accurate among these models because it has the flexibility to tolerate model errors that result from the difficulty of rigorous expression [2]. However, this work also suggested that the full linear model cannot accurately predict nonlinear motion in rough terrain.

## 2.4. Proprioceptive and exteroceptive sensor fusion-based odometry for wheeled robots

Accuracy of the wheel odometry model suffers from errors in kinematic parameters (e.g., wheel radius) and terrain-dependent parameters (e.g., friction coefficients and slip ratio). Thus, many approaches corrected these errors dynamically based on online calibration by incorporating exteroceptive sensors (e.g., LiDAR and camera). Simultaneous Calibration, Localization, and Mapping [13, 8] solves LiDAR-based SLAM and online calibration problems jointly. Specifically, these works calibrated kinematic parameters (wheel radii and wheelbase of the differential drive robot) and extrinsic parameters (2D relative pose between the LiDAR frame and the robot frame) such that wheel odometry-based and LiDAR-based motions become consistent. Extending these works, Lee et.al. fused camera, IMU, and wheel encoder data to solve SLAM and online calibration jointly for correcting a time offset between IMU and encoder data in addition to kinematic parameters and extrinsic parameters [15]. Murai et.al. conducted distributed simultaneous localization and online calibration of extrinsic parameters via Gaussian belief propagation in a factor graph [19].

The aforementioned online calibration methods aim to correct the kinematic model of a differential drive robot under the ideal condition that wheel slippage and lateral motion are not present. Some approaches conducted online calibration for a kinematic model of skid-steering robots [37, 38, 21]. Zuo et.al. applied an ICR-based model [18] to the kinematic model of a skid-steering robot, and jointly solved online calibration of its model and visual-IMU SLAM [37, 38]. These works enabled a more accurate estimation than the visual-IMU fusion owing to the constraint of wheel odometry incorporating online calibration. Our previous work conducted online calibration of the full linear model [1], which can implicitly express unknown kinematic parameters and environment-dependent values that are difficult to model rigorously [21]. In contrast to [37, 38], our previous work demonstrated accurate odometry estimation even in a featureless environment (e.g., long corridors) by handling degeneracy of LiDAR point clouds.

### 2.5. Training neural networks on a factor graph

The online calibration methods described in Section 2.4 assume a linear model for skid-steering robots, and thus accuracy is not ensured for nonlinearity. The nonlinearity can be implicitly expressed by a neural network, and thus we extended our previous work [21] by replacing the linear model-based constraint with a neural network-based one. Some odometry algorithms [4, 32] fused a neural network-based motion model and other constraints by factor graph optimization; however, the network was trained offline independent of odometry estimation. The neural networks constrained robot poses based on fixed models and were not directly and dynamically trained on the factor graph. Different from the aforementioned works, the proposed method jointly accomplishes tightly coupled LiDAR-IMU-wheel odometry and online training of the kinematic model for skid-steering
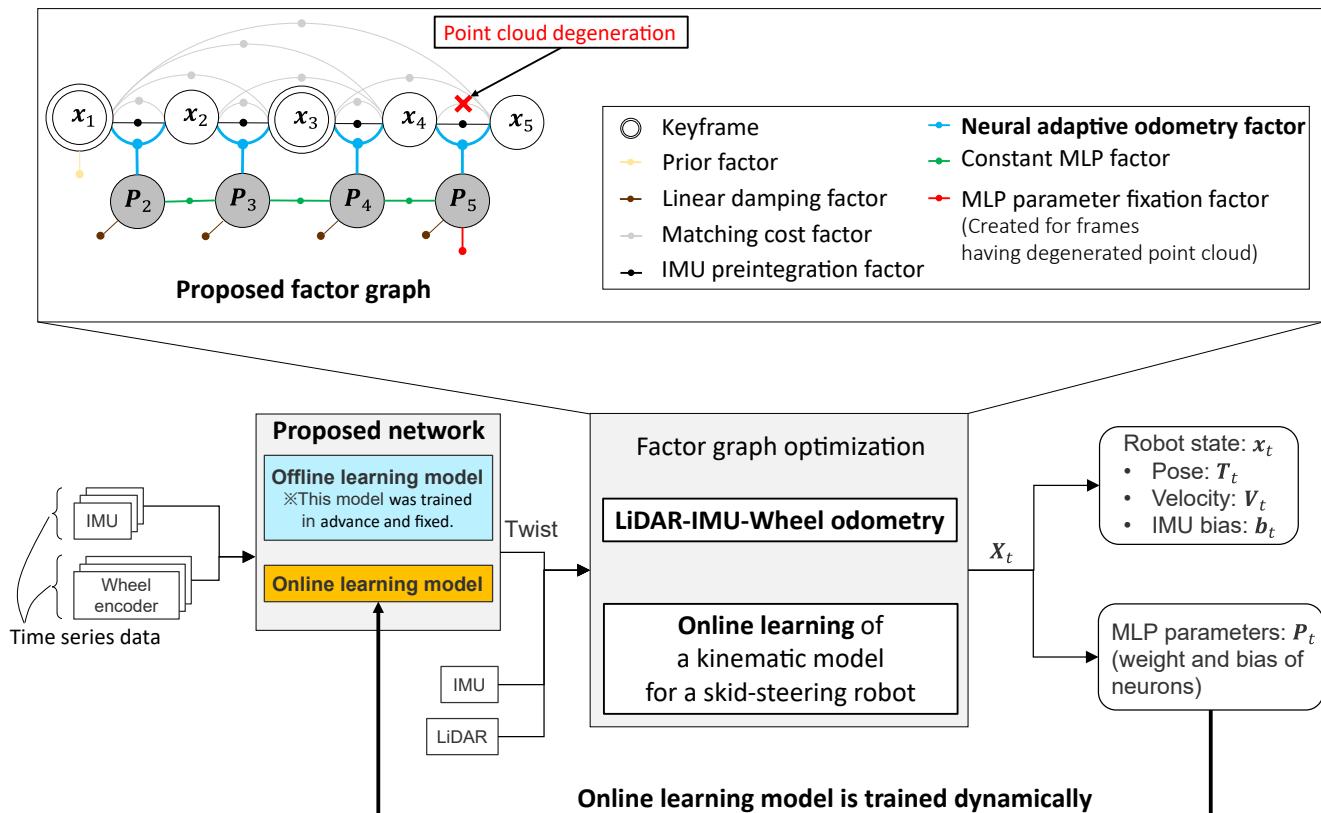
**Figure 3:** Overview of the proposed framework. We simultaneously solve online training of a neural network and LiDAR-IMU-wheel odometry on a unified factor graph based on a tightly coupled way. We designed the neural network to be divided into the *offline learning model* and *online learning model* to balance the computational cost and inference accuracy. We extended our previous work by introducing a *neural adaptive odometry factor*, which provides motion constraints through a learning-based motion model and training of the *online learning model*.

robots. The proposed method trains a network in a factor graph such that the neural network-based constraint and all other constraints (i.e., LiDAR-IMU-wheel odometry) are consistent.

## 3. Methdlogy

### 3.1. System overview

To estimate the time series of robot poses under severe point cloud degeneration and the nonlinear kinematic behavior of a skid-steering robot, we propose a tightly-coupled LiDAR-IMU-wheel odometry algorithm incorporating online training of the kinematic model, as seen in Figure 3. Odometry estimation and online training of the kinematic model are performed simultaneously on a unified factor graph to make all those constraints consistent.

Online training of the neural network enables terrain-dependent features to adapt to the current terrain condition. However, there is a trade-off between computational cost and inference accuracy for training the network; thus, training an accurate and large network online is infeasible. Therefore, we divided the neural network into an *offline learning model* and *online learning model*. The *offline learning model* is trained in advance for expressing static terms (e.g., terrain-independent terms) of the kinematic models. In contrast, the *online learning model* is trained along with LiDAR-IMU-wheel odometry for expressing dynamically changing terms (e.g., terrain-dependent terms) of the kinematic models.

We define the state $X_t$ to be estimated as

$$X_t = [x_t, P_t], \tag{1}$$

$$x_t = [T_t, v_t, b_t], \tag{2}$$

where $x_t$ is the robot state at time $t$; $T_t = [R_t | t_t] \in SE(3)$ and $v_t \in \mathbb{R}^3$ are the robot pose and velocity, respectively. $b_t = [b_t^a, b_t^\omega] \in \mathbb{R}^6$ is the bias of the IMU acceleration $a_t$ and the angular velocity $\omega_t$. $P_t$ indicates the weight and bias of each neuron to be learned online.

We assume that $P_t$ is trained online along with the robot states $x_t$ in feature-rich environments before entering environments where LiDAR point clouds can degenerate. We also assume that the terrain condition does not drastically change while transitioning from feature-rich environments to another environment where LiDAR point clouds can degenerate. Once the online-trained neural network converges to a proper model that adapts to the current terrain condition, this model's inference becomes a reliable constraint for optimization and enables an accurate odometry estimation even in such severe environments.

Specifically, factor graph optimization is used for simultaneously performing LiDAR-IMU-wheel odometry and online training of the learning-based kinematic model, as shown in Figure 3. Details regarding each factor are explained in the following subsections.

### 3.2. Matching cost factor

The matching cost factor [10] references robot poses $T_i$ and $T_j$ to align their point cloud $\mathcal{P}_i$ and $\mathcal{P}_j$ based on voxelized GICP (VGICP) registration with GPU acceleration [11]. The VGICP algorithm treats point $p_k$ in $\mathcal{P}_i$ as a Gaussian distribution; $p_k = (\mu_k, C_k)$, where $\mu_k$ and $C_k$ are the mean and covariance matrix estimated from the neighbors of $p_k$, respectively. The target point cloud $\mathcal{P}_j$ is discretized by voxelization, and each voxel contains the average of the means and covariance matrices within its voxel points. This voxelization process enables a quick nearest-neighbor search through spatial hashing [31]. The matching cost $e^{\mathrm{M}}(T_i, T_j)$ is defined as follows:

$$e^{\mathrm{M}}(T_i, T_j) = \sum_{p_k \in \mathcal{P}_i} e^{\mathrm{D2D}}(p_k, T_i^{-1} T_j), \tag{3}$$

$$e^{\mathrm{D2D}}(p_k, T_{ij}) = d_k^\top (C_k' + T_{ij} C_k T_{ij}^\top)^{-1} d_k, \tag{4}$$

where $e^{\mathrm{D2D}}$ is the distribution-to-distribution Mahalanobis distance between the input point $p_k$ and the corresponding voxel $p_k' = (\mu_k', C_k')$, and $d_k = \mu_k' - T_{ij} \mu_k$ is the residual between $\mu_k$ and $\mu_k'$.

As the implementation details, we conduct a deskewing process of input point cloud $\mathcal{P}_i$ by using IMU measurements for preprocessing. In addition, we manage the keyframes based on an overlap ratio between the input point cloud and the target point cloud. If the overlap ratio is smaller than a threshold (e.g., 90%), we set the frame of the input point cloud as a keyframe. As seen in the proposed factor graph in Figure 3, we constrain each frame by matching cost factors with its last $N$ (e.g., 3) frames and previous keyframes to reduce the estimation drift. More details of this implementation are given in [12].

### 3.3. IMU preintegration factor

The IMU preintegration factor is a constraint for the relative pose and the linear velocity between two consecutive frames by integrating IMU measurements ($a_i$ and $\omega_i$). This factor enables efficient optimization based on the preintegration technique, which avoids the recomputation of IMU measurement integration in every optimization iteration. The IMU data are used to predict the sensor state at a time interval $\Delta t$ (i.e., the frequency of IMU measurements) as follows:

$$R_{t+\Delta t} = R_t \exp\left(\left(\omega_t - b_t^\omega - \eta_k^\omega\right) \Delta t\right), \tag{5}$$

$$v_{t+\Delta t} = v_t + g\Delta t + R_t \left(a_t - b_t^a - \eta_t^a\right) \Delta t, \tag{6}$$

$$t_{t+\Delta t} = t_t + v_t \Delta t + \frac{1}{2} g\Delta t^2 + \frac{1}{2} R_t \left(a_t - b_t^a - \eta_t^a\right) \Delta t^2, \tag{7}$$
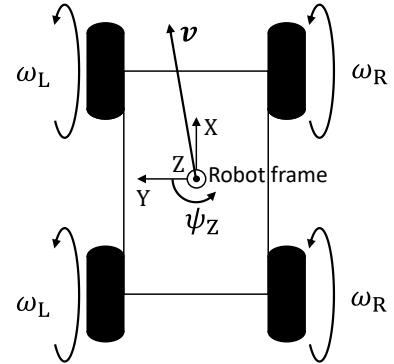


**Figure 4:** Kinematic model of a skid-steering robot.

where $g$ denotes the gravity vector and $\eta_t^a$ and $\eta_t^\omega$ are the white noise of the IMU measurement. We can calculate the relative robot motion $\Delta R_{ij}$, $\Delta v_{ij}$, and $\Delta t_{ij}$ by integrating $R_{t+\Delta t}$, $v_{t+\Delta t}$, and $t_{t+\Delta t}$, respectively, between time steps $i$ and $j$. The error $e^{\mathrm{IMU}}(x_i, x_j)$ between consecutive frame states $x_i$ and $x_j$ is finally defined as follows:

$$e^{\mathrm{IMU}}\left(x_i, x_j\right) = \left\| \log\left(\Delta R_{ij}^\top R_i^\top R_j\right) \right\|^2$$
$$+ \left\| \Delta t_{ij} - R_i^\top \left(t_j - t_i - v\Delta t_{ij} - \frac{1}{2} g\Delta t_{ij}^2\right) \right\|^2$$
$$+ \left\| \Delta v_{ij} - R_i^\top \left(v_j - v_i - g\Delta t_{ij}\right) \right\|^2. \tag{8}$$

More details are given in [6]. The IMU preintegration factor enables robust estimation for rapid motion and short-time degeneracy of point clouds. Furthermore, this factor can reduce the estimation drift of poses from six to four DoFs [26].

### 3.4. Neural adaptive odometry factor

We propose a neural network that infers the 2D twist of a skid-steering robot with proprioceptive sensors, namely encoder and IMU. This neural network consists of two sub-networks: 1) *offline learning model* describing static features (e.g., terrain-independent parameters) of the kinematic model and 2) *online learning model* expressing dynamically changing features (e.g., terrain-dependent terms). The *offline learning model* is obtained by batch learning before odometry estimation is conducted. In contrast, the *online learning model* is trained online by factor graph optimization along with the robot state $x$. To conduct the online learning by factor graph optimization, we propose the neural adaptive odometry factor to constrain the robot pose $T$, the linear velocity $v$, and the parameter vector $P$ concatenating the weight and bias of each neuron to be trained online.

#### 3.4.1. Fundamental kinematic model for wheeled robots

First, we introduce the fundamental formulation of a kinematic model for wheeled robots. In the case of an ideal
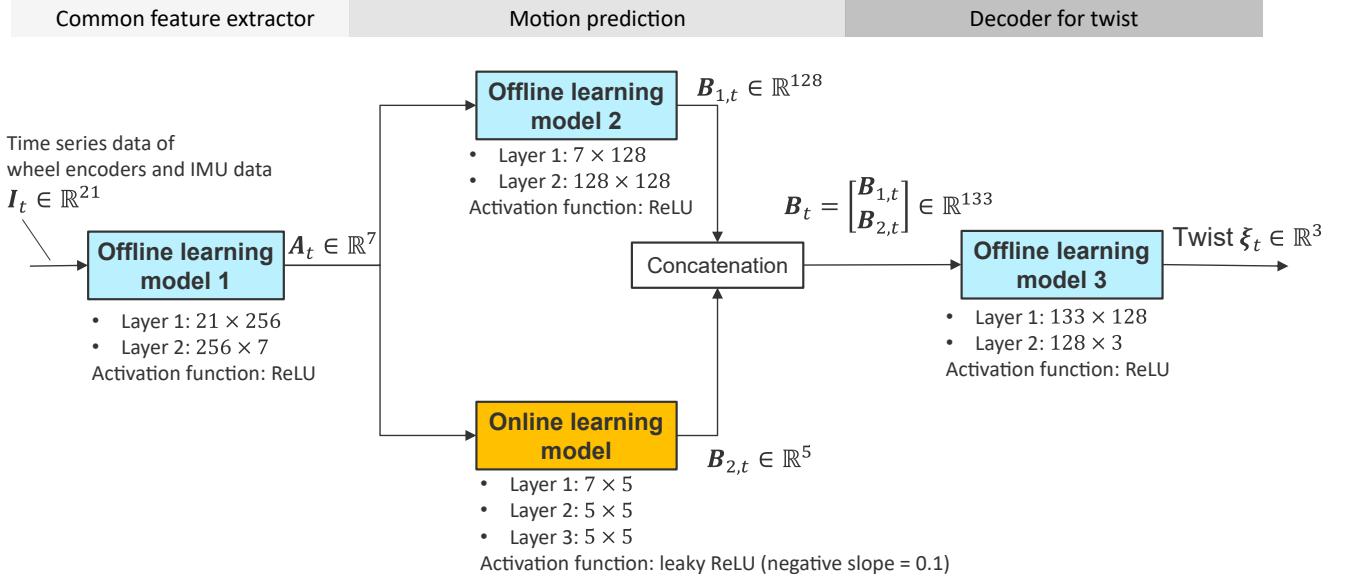
**Figure 5:** Proposed neural network structure. The network outputs twist from the time series data of wheel encoders and IMU data, through the following three layers: 1) *Common feature extractor*, 2) *Motion prediction*, and 3) *Decoder for twist*. We divided the second layer into two blocks to be trained online and offline for capturing terrain-dependent and -independent properties of the kinematic model, respectively. The number of neurons in the *online learning model* was set to 100 ((7×5+5)+(5×5+5)+(5×5+5)); thus, the dimension of MLP parameter $P$ is 100 in our implementation.

condition (e.g., wheel slippage and lateral motion do not occur, and wheels and terrain are not deformed), the robot twist motion is described by a simple kinematic equation:

$$\begin{bmatrix} v_x \\ v_y \\ \psi_z \end{bmatrix} = J\boldsymbol{\omega} = \begin{bmatrix} R/2 & R/2 \\ 0 & 0 \\ -R/B & R/B \end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix}, \quad (9)$$

where $\boldsymbol{v} = [v_x \ v_y]^T$ is the 2D translational velocity; $\psi_z$ is the angular velocity around the z-axis of the robot frame; $R$ and $B$ indicate the wheel radius and wheelbase, respectively; $\boldsymbol{\omega} = [\omega_L \ \omega_R]^T$ is the angular velocity of the left and right wheels, as shown in Figure 4. $J$ is a $3 \times 2$ matrix that describes the relationship between $[v_x \ v_y \ \psi_z]^T$ and $\boldsymbol{\omega}$. $J$ is constant in the case of ideal conditions. As seen in Eq. 9, the relationship between the robot twist $[v_x \ v_y \ \psi_z]$ and the wheels' angular velocity $\boldsymbol{\omega}$ is clearly linear.

In the case of skid-steering robots, Eq. 9 does not fully capture the robot motion, because the skid-steering robot rotates by skidding its wheels depending on the difference between the angular velocities of the left and right wheels. Thus, wheel slippage and lateral motion $v_y$ must be considered for skid-steering robots. However, incorporating these terms makes the robot twist motion model more complex because this rigorous model depends on terrain-dependent parameters (e.g., friction coefficients). Many works modeled the $J$ matrix for expressing the complex motion for skid-steering robots[1, 14, 18, 33, 27]; however, these linear models cannot capture nonlinear kinematic behavior (e.g., large wheel slippage such as drifting).

### 3.4.2. *Proposed neural network structure*

We implicitly express the complex motion model with nonlinearity based on a nonlinear neural network. The proposed neural network must be trained online to adapt to each terrain condition because the kinematic model is subject to terrain-dependent parameters. As shown in Figure 5, we designed the network with three layers: 1) the first layer extracts common features for motion prediction from the time series data of wheel encoders and IMU data, 2) the second layer predicts motion in the latent space, and 3) the third layer decodes the features into the twist. We divided the second motion prediction layer into two blocks to be trained online and offline for capturing terrain-dependent and -independent properties of the kinematic model, respectively. We designed the proposed neural network such that the *online learning model* represents terrain-dependent properties with fewer neurons compared to those in the *offline learning models* for balancing inference accuracy and computational costs. In our implementation, the number of neurons in the *online learning model* was set to 100; that is, the dimension of MLP parameters vector $\boldsymbol{P}_t$ is 100.

The input data of the proposed neural network is the time series data measured by the wheel encoders and IMU, namely, all-wheel rotational velocities $\omega_t^{LF}, \omega_t^{LH}, \omega_t^{RH}, \omega_t^{RF}$, incremental velocity $\Delta v_{x,t}^{IMU}, \Delta v_{y,t}^{IMU}$, and an angular velocity $\omega_{z,t}^{IMU}$. Note that $\Delta v_{x,t}^{IMU}, \Delta v_{y,t}^{IMU}$, and $\omega_{z,t}^{IMU}$ are described in the robot frame. We define $\boldsymbol{i}_t$ as a concatenated vector of those seven types of sensor data as follows:

$$\boldsymbol{i}_t = [\omega_t^{LF}, \omega_t^{LH}, \omega_t^{RH}, \omega_t^{RF}, \Delta v_{x,t}^{IMU}, \Delta v_{y,t}^{IMU}, \omega_{z,t}^{IMU}].$$
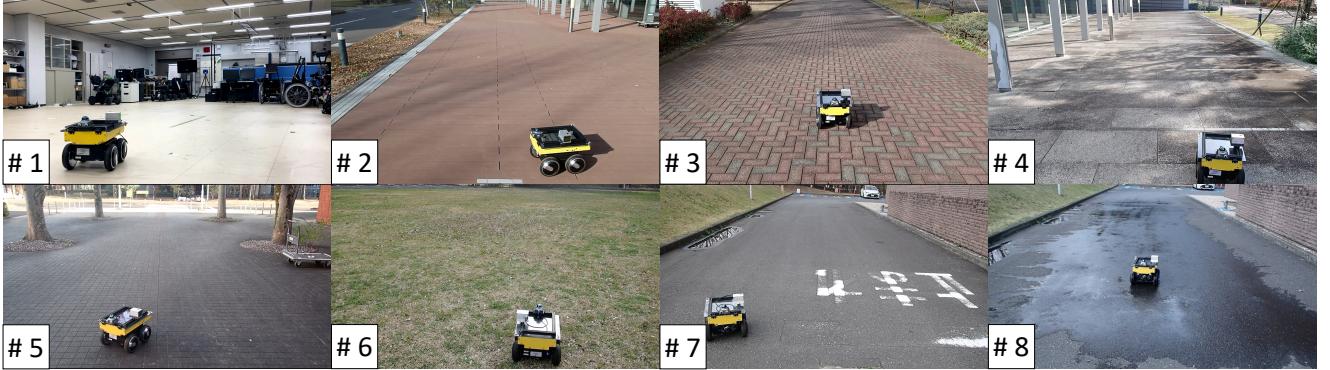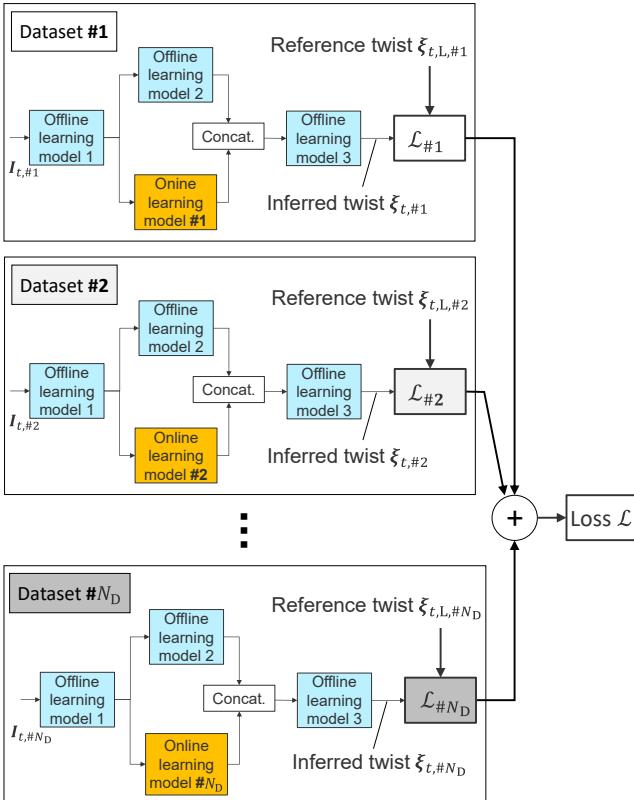
$$(10)$$

**Figure 6:** Each environment of datasets used for offline learning: #1, indoor flat floor; #2, wood tiles; #3, bricks; #4, stone tiles; #5, cement tiles; #6, grass; #7, dry asphalt; and #8, wet asphalt. The robot was operated with various speeds (0 to 3 m/s) and trajectories (e.g., straight, curved, and pivot turns) in each dataset. Environments in all datasets were surrounded by many features, and thus the result of LiDAR-IMU odometry can be used as reference data for offline learning.



**Figure 7:** Offline learning procedure for obtaining *offline learning model 1, 2, and 3*. The loss function $\mathcal{L}$ is defined as the sum of all loss functions from $\mathcal{L}_{\#1}$ to $\mathcal{L}_{\#N_D}$ obtained with all datasets. In our implementation, we set the number of datasets $N_D$ to 8, as shown in Figure 6.

We concatenate consecutive $N_w$ (e.g., 3) frames of $\boldsymbol{i}_t$ to compose an input vector $\boldsymbol{I}_t = \rho([\boldsymbol{i}_t, \boldsymbol{i}_{t-1}, ...])$, where $\rho$ is the standardization operation. The proposed neural network outputs twist $\boldsymbol{\xi}_t \in \mathbb{R}^3$ through the input data $\boldsymbol{I}_t$ as shown in Figure 5.

### 3.4.3. Offline learning procedure

In the offline training phase, we aim to train *offline learning model 1, 2, and 3* in Figure 5 to capture the robot behavior while conditioning them to the *online learning model*, which will be trained online to encode dynamic environment information. We assume that the parameters of the *online learning model* do not drastically change when the environment is the same. Thus, we conduct offline learning such that, while *offline learning model 1, 2, and 3* consist of the same parameters for all environments, the *online learning model* consists of different parameters for each environment.

To conduct offline learning in this framework, we prepared a dataset including $N_D$ (e.g., 8) sequences in different environments, such as those shown in Figure 6. As seen in Figure 7, each loss function $\mathcal{L}_{\#n}$ is defined for each dataset, and the offline learning is executed to minimize the sum of the loss functions $\mathcal{L}$. We train the proposed network so that, while *offline learning model 1, 2, and 3* are common for all sequences, the *online learning model* is different for each sequence. Specifically, $\mathcal{L}$ is defined as follows:

$$\mathcal{L} = \sum_{n=1}^{N_D} \mathcal{L}_{\#n}, \tag{11}$$

$$\mathcal{L}_{\#n} = \frac{1}{n_D} \sum_{t=1}^{n_D} (e_{t,\text{trans}} + w e_{t,\text{rot}}), \tag{12}$$

where $\mathcal{L}_{\#n}$ is defined as the Mean Square Error (MSE) of the estimated twist for each dataset; $n_D$ is the number of reference data in each dataset; $e_{t,\text{trans}}$ and $e_{t,\text{rot}}$ are the norm of error of the translation and rotation components of the twist $\boldsymbol{\xi}_t$, respectively; and $w$ is a weight (e.g., 32) for adjusting the scale of translation and rotation.

We used the Adam optimizer [9] with a learning rate of $10^{-4}$ and 3000 epochs. The batch size was set to 128 in our implementation. The number of layers and activation functions for each MLP are shown in Figure 5. We utilized

LiDAR-IMU odometry with an omnidirectional field-of-view (FOV) LiDAR (Livox MID-360) to obtain the reference data of the twist $\xi_{t,\mathrm{L}}$. LiDAR-IMU odometry is highly accurate in feature-rich environments, such as those in the datasets we used (Figure 6). The robot also moved locally in each environment to always remain on the local map, and thus estimation drift did not occur in LiDAR-IMU odometry. In each dataset, we manually operated the robot at a range of speeds (0 to 3 m/s) and various motions (straight and curved trajectories, and pivot turns).

### 3.4.4. Error derivation of neural adaptive odometry factor for online learning

Unlike the *offline learning model*, the *online learning model* is adaptively updated to better capture the dynamic environment features obtained by the *neural adaptive odometry factor* through factor graph optimization. This factor provides not only motion constraints but also training for the *online learning model*, and thus the parameter vector $\boldsymbol{P}$ concatenating weights and biases of neurons is trained online via factor graph optimization on a unified factor graph along with robot state $\boldsymbol{x}$.

First, we describe the pose error $e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{P}_i)$. We extend the output twist $\xi_i \in \mathbb{R}^3$ into $\xi_{i,\mathrm{6DoF}} \in \mathbb{R}^6$ by substituting 0 into the translational z and rotational x, y of $\xi_{i,\mathrm{6DoF}}$. We calculate the displacement by integrating $\xi_{i,\mathrm{6DoF}}$ in the time step $\Delta t_i$. The pose error $e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{P}_i)$ is defined as follows:

$$e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{P}_i) = \boldsymbol{r}_i^{\mathrm{NN_p}\top} \boldsymbol{C}_i^{\mathrm{NN}-1} \boldsymbol{r}_i^{\mathrm{NN_p}}, \tag{13}$$

$$\boldsymbol{r}_i^{\mathrm{NN_p}} = \log(\boldsymbol{T}_i^{-1}\boldsymbol{T}_i \exp(\xi_{i,\mathrm{6DoF}}\Delta t_i)^{-1}), \tag{14}$$

where $\boldsymbol{C}_i^{\mathrm{NN}-1}$ is the covariance matrix of the displacement. We estimate $\boldsymbol{C}_i^{\mathrm{NN}-1}$ online with the Kalman filter (as described in our previous work [21]) to obtain the uncertainty of the 3D motion. This method can adaptively set the covariance matrix based on terrain roughness.

Next, we define the linear velocity error $e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i)$. We extend the twist $\xi_i \in \mathbb{R}^3$ into the 3D linear velocity $\boldsymbol{v}_i^{\mathrm{NN}} \in \mathbb{R}^3$ by substituting 0 into the z element of $\boldsymbol{v}_i^{\mathrm{NN}}$. The linear velocity error $e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i)$ is defined as follows:

$$e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i) = \boldsymbol{r}_i^{\mathrm{NN_v}\top} \boldsymbol{C}_i^{\mathrm{NN_v}-1} \boldsymbol{r}_i^{\mathrm{NN_v}}, \tag{15}$$

$$\boldsymbol{r}_i^{\mathrm{NN_v}} = \boldsymbol{v}_{i-1} - \boldsymbol{R}_{i-1}\boldsymbol{v}_i^{\mathrm{NN}}, \tag{16}$$

where $\boldsymbol{C}_i^{\mathrm{NN_v}-1}$ is the covariance matrix of the linear velocity. We set all diagonal elements of $\boldsymbol{C}_i^{\mathrm{NN_v}-1}$ to $10^{-6}$.

Finally, the error of the neural adaptive odometry factor $e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i)$ is defined as the sum of $e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{P}_i)$ and $e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i)$ as follows:

$$e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i) = e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{P}_i)$$
$$+ e^{\mathrm{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i). \tag{17}$$

### 3.5. Constant MLP parameter factor

We assume that dynamic environment parameters do not change drastically over short time periods and create a *constant MLP parameter factor* between consecutive states with zero-mean Gaussian noise. The *constant MLP parameter factor* constrains the MLP parameters of consecutive time steps to become mostly identical under the random walk assumption. The error $e^{\mathrm{C}}(\boldsymbol{P}_{i-1}, \boldsymbol{P}_i)$ of this factor is defined as follows:

$$e^{\mathrm{C}}(\boldsymbol{P}_{i-1}, \boldsymbol{P}_i) = \boldsymbol{r}_i^{\mathrm{C}\top} \boldsymbol{C}_i^{\mathrm{C}-1} \boldsymbol{r}_i^{\mathrm{C}}, \tag{18}$$

$$\boldsymbol{r}_i^{\mathrm{C}} = \boldsymbol{P}_i - \boldsymbol{P}_{i-1}. \tag{19}$$

In our implementation, we set all diagonal elements of the covariance matrix $\boldsymbol{C}_i^{\mathrm{c}-1}$ of this factor to $10^{-6}$.

### 3.6. MLP parameter fixation factor

The neural adaptive odometry factor facilitates the online learning for $\boldsymbol{P}_i$ and serves as a motion constraint. Therefore, online learning can be unstable when point cloud-based constraints become unreliable due to point cloud degeneration. To resolve this problem, we introduce a *MLP parameter fixation factor* for $\boldsymbol{P}_i$ whenever point cloud degeneration is detected in the current frame.

After the factor graph is optimized, we detect point cloud degeneration by analyzing the linearized system of the matching cost factor (point cloud-based constraint). This procedure involves two steps: 1) calculating the Hessian matrix of a matching cost factor between the latest and last frames and 2) determining degeneration if the minimum eigenvalue of the Hessian matrix is smaller than a certain threshold (i.e., the Hessian matrix is not positive definite). If the current point cloud $\mathcal{P}_i$ is determined as degenerate, we create the *MLP parameter fixation factor* to ensure that $\boldsymbol{P}_i$ is retained to the values ($\tilde{\boldsymbol{P}}_{i-1}$) just before degeneration occurs to avoid numerically unstable optimization. The error $e^{\mathrm{F}}(\boldsymbol{P}_i)$ of this factor is defined as follows:

$$e^{\mathrm{F}}(\boldsymbol{P}_i) = \begin{cases} \boldsymbol{r}_i^{\mathrm{F}\top} \boldsymbol{C}_i^{\mathrm{F}-1} \boldsymbol{r}_i^{\mathrm{F}}, & \text{if } \mathcal{P}_i \text{ is degenerated} \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

$$\boldsymbol{r}_i^{\mathrm{F}} = \boldsymbol{P}_i - \tilde{\boldsymbol{P}}_{i-1}. \tag{21}$$

Note that $e^{\mathrm{F}}(\boldsymbol{P}_i)$ become 0 when the current point cloud $\mathcal{P}_i$ is not detected as degenerated. $\tilde{\boldsymbol{P}}_{i-1}$ is a fixed MLP parameter at time $i-1$, and not optimized. All diagonal elements of the covariance matrix $\boldsymbol{C}_i^{\mathrm{F}-1}$ of this factor are set to $10^{-12}$ to represent hard constraints in our implementation. We also use similar factors for the IMU bias to ensure stable optimization.

### 3.7. Implementation details

In summary, the objective function $e^{\mathrm{ALL}}$ of our system is defined as follows:

$$e^{\mathrm{ALL}} = \underbrace{e^{\mathrm{Prior}}(\boldsymbol{x}_1)}_{\text{Prior factor}} + \sum_{j \in \mathcal{K},\, i=2} \underbrace{e^{\mathrm{M}}(\boldsymbol{T}_j, \boldsymbol{T}_i)}_{\text{Matching cost factor}}$$

$$+ \sum_{i=2} \underbrace{e^{\text{IMU}}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i)}_{\text{IMU preintegration factor}}$$

$$+ \sum_{i=2} \underbrace{e^{\text{NN}}(\boldsymbol{T}_{i-1}, \boldsymbol{T}_i, \boldsymbol{v}_{i-1}, \boldsymbol{P}_i)}_{\text{Neural adaptive odometry factor}}$$

$$+ \sum_{i=3} \underbrace{e^{\text{C}}(\boldsymbol{P}_{i-1}, \boldsymbol{P}_i)}_{\text{Constant MLP parameter factor}}$$

$$+ \sum_{i=3} \underbrace{e^{\text{F}}(\boldsymbol{P}_i)}_{\text{MLP parameter fixation factor}} . \tag{22}$$

As seen in the factor graph in Figure 3, the error $e^{\text{Prior}}(\boldsymbol{x}_1)$ of the prior factor constrains the initial robot state $\boldsymbol{x}_1$ to the fixed reference frame. Similar to the Levenberg-Marquardt algorithm, we add a constant damping element to the Hessian matrix related to $\boldsymbol{P}$ at each timestep to help the optimization become more stable because $\boldsymbol{P}$ is a large variable with 100 dimensions, which can cause numerically unstable optimization. As stated in Section 3.2, $\mathcal{K}$ is a set of point clouds as follows: 1) the last three point clouds $\mathcal{P}_{i-1}, \mathcal{P}_{i-2}, \mathcal{P}_{i-3}$ and 2) keyframes.

We used GTSAM to implement the factor graph and the iSAM2 [7] optimizer to incrementally optimize the factor graph (i.e., $e^{\text{ALL}}$). We also used PyTorch and LibTorch for implementing offline learning procedures (Figure 7) and the neural adaptive odometry factor, respectively.

## 4. Experimental results

First, we verified that the network we developed can properly adapt to each terrain condition (Section 4.1). Second, we demonstrated that the proposed odometry estimation algorithm can deal with severe point cloud degeneration and large wheel slippage (i.e., nonlinearity in the kinematic model of the skid-steering robot) (Section 4.2).

### 4.1. Verification evaluation of the proposed neural network

To demonstrate that the proposed network (Figure 5) and training approach (Figure 7) enable robust odometry estimation in different terrains, we conducted an ablation study that evaluated the wheel odometry estimation of the network only, that is, without LiDAR and IMU constraints. We compared the proposed trained network (*Ours*) with other networks trained through two offline learning procedures as follows:

1. *Identical network*: In this model, a single *online learning model* (i.e., simple batch learning-based model) was fitted to all the datasets; thus, features depending on terrain types could not be expressed accurately.
2. *Improper dataset pair*: This setting used the parameters of the *online learning model* that was trained on a sequence different from that of the testing environment. We used the following two combinations of the *online learning model*:

(a) Case 1: Training=indoor flat floor, Testing=wood tiles
(b) Case 2: Training=indoor flat floor, Testing=grass

As seen in Figure 6, in Case 1 (indoor flat floor (#1) and wood tiles (#2)), the terrain differences were slight, whereas in Case 2 (indoor flat floor (#1) and grass (#6)), the differences in terrain conditions were significant. The indoor flat floor and wood tiles were flat and hard, whereas the grass was uneven and soft.

We used a Rover mini (Rover Robotics) equipped with two types of LiDAR, as shown in Figure 2. To conduct the offline learning procedure, we recorded the point cloud and IMU data using omnidirectional FOV LiDAR (Livox Mid-360), and the wheel angular velocities using each wheel encoder. Specifically, point clouds, IMU measurements, and the wheel angular velocities were recorded at 10, 200, and 60 Hz, respectively. In this experiment, IMU measurements and wheel angular velocities were used as input data for the proposed neural network inferring the 2D twist of a skid-steering robot. Whereas, point clouds and IMU measurements were used for creating reference data of the 2D twist based on tightly coupled LiDAR-IMU odometry in feature-rich environments. We used 70 % and 30 % of the dataset (Figure 6) for training and validation, respectively.

Table 1 shows the relative trajectory errors (RTEs) for the proposed method (Ours) and *Identical network*. The translation RTEs of the proposed method (Mean: 0.021 m) were about 1.5 times as accurate as those of the *Identical network* (Mean: 0.031 m) in all datasets because translational accuracy depends on terrain condition changes. The difference in rotational accuracy between these methods was slight (0.02 deg). This result shows the proposed network can properly adapt to each terrain condition.

Table 2 presents the RTEs of the proposed method and the *Improper dataset pair* case for three types of datasets (indoor flat floor, wood tiles, and grass). This result also shows that the proposed method (Ours) is the most accurate translation RTEs in all cases. The average of translation RTEs of our method in terrains #1 and #2 (0.017 m) was twice as accurate as that of Case 1 (0.034 m). In addition, the average of translation RTEs of our method in terrains #1 and #6 (0.028 m) was about six times as accurate as that of Case 2 (0.158 m). Whereas, the accuracy of rotation RTEs is almost the same in all cases. Furthermore, the translation RTEs of Case 1 are more accurate than those of Case 2 because, while the difference between *indoor flat floor* and *wood tiles* (Case 1) is slight, the difference between *indoor flat floor* and *grass* (Case 2) is significant. Hence, this result implies that the network trained for an *indoor flat floor* was almost the same as the network trained for *wood tiles* compared with another network trained for *grass*. Clearly, *indoor flat floor* was more similar to another flat and hard terrain (*wood tiles*) than to grass (i.e., rough and soft terrain). Therefore, this result demonstrates that the proposed network was trained to properly represent features that vary with the environment.

**Table 1**
RTEs per 1 m for the ablation study on the *Identical network* case on eight terrains. While the proposed method (Ours) trained the proposed neural network to adapt to the current environment (i.e., the *online learning model* was trained for each different dataset), the *Identical network* case trained its model to fit to all the datasets.

| Method/Seq. | #1 | | #2 | | #3 | | #4 | | #5 | | #6 | | #7 | | #8 | | Mean | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $t_{\mathrm{RTE}}$[1] | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ |
| Ours | **0.017** | 0.24 | **0.016** | 0.25 | **0.016** | 0.21 | **0.016** | 0.20 | **0.019** | 0.28 | **0.038** | 0.31 | **0.021** | 0.24 | **0.025** | 0.23 | **0.021** | 0.25 |
| Identical network | 0.025 | 0.22 | 0.023 | 0.20 | 0.034 | 0.20 | 0.018 | 0.17 | 0.027 | 0.36 | 0.051 | 0.27 | 0.030 | 0.21 | 0.038 | 0.18 | 0.031 | 0.23 |

[1] $t_{\mathrm{RTE}}$ and $r_{\mathrm{RTE}}$ are the translation [m] and rotation [deg/m] RTEs, respectively.

**Table 2**
RTEs per 1 m for the ablation study on the *Improper dataset pair* case for three types of datasets (#1, indoor flat floor; #2, wood tiles; and #6, grass in Figure 6). Note that the results of the proposed method (Ours) in this table are the same as those in Table 1. While the *Improper dataset pair* case cannot express features fitting to proper terrain conditions, our network was trained to adapt to the current environment. Hence, this result shows that our network was trained properly.

| Method/Seq. | #1 | | #2 | | #6 | |
|---|---|---|---|---|---|---|
| | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ | $t_{\mathrm{RTE}}$ | $r_{\mathrm{RTE}}$ |
| Ours | **0.017** | 0.24 | **0.016** | 0.25 | **0.038** | 0.31 |
| Case 1[1] | 0.033 | 0.36 | 0.034 | 0.24 | | |
| Case 2[2] | 0.15 | 0.36 | | | 0.165 | 0.34 |

[1] Training: indoor flat floor; Testing: wood tiles.
[2] Training: indoor flat floor; Testing: grass.



**Figure 8:** Example of situations where point clouds are unavailable due to the distance between the LiDAR and walls being too small. For the Livox AVIA LiDAR used for odometry estimation in this experiment, the minimum observation range is 1 m. Thus, we also verified the robustness of our odometry estimation algorithm to point cloud absence in addition to point cloud degeneration.

## 4.2. Accuracy evaluation of the proposed odometry estimation algorithm

We conducted experiments to evaluate the robustness of the proposed method against severe point cloud degeneration under conditions producing large wheel slippage (i.e., nonlinear kinematic behavior of the skid-steering robot). We also evaluated adaptability of our network to changing terrain conditions. We operated the robot testbed (Figure 2) at a maximum speed of 3 m/s (the maximum rotational speed of each wheel is 36 rad/s) to induce significant wheel slippage. Refer to Video 1 [1] for an understanding of the maximum wheel speed as a reference. In contrast to the evaluation described in Section 4.1, we applied the narrow FOV LiDAR (Livox AVIA) to the proposed odometry estimation algorithm to imitate severe point cloud degeneration in, for example, tunnels. Furthermore, we placed the narrow FOV LiDAR toward the Y direction (side) of the robot frame (Figure 4). Hence, degenerated point clouds were obtained when the robot moved near a wall as shown in Figure 1-(d). In this experiment, we input all wheel encoder values, IMU measurements, and the point cloud measured by the narrow FOV LiDAR into the proposed odometry estimation algorithm for estimating the robot state $X_t$ and MLP parameters $P_t$ on a unified factor graph, as shown in Figure 3.

The proposed method was evaluated for three sequence types. *Seq. 1* included severe degeneracy of point clouds

in straight long corridors and drifting (Figure 1-(e)). In the case of *Seq. 2*, the robot transited flat hard terrain, such as bricks, outdoor stone tiles, and indoor stone tiles, as shown in Figure 1-(b). *Seq. 3* included drastic terrain changes from concrete to grass, as shown in Figure 1-(c).

Table 3 summarizes the number of frames where the point cloud data became 1) degenerate or 2) unavailable (fewer than 100 points in a frame). The narrow FOV LiDAR was always pointed at the walls in all sequences; thus, the observed point clouds were mostly degenerated. Furthermore, the point cloud measurements became completely unavailable when the sensor approached the wall (Figure 8) because the LiDAR cannot observe points closer than the minimum observation range (1 m). *Seq. 1* was an especially difficult condition compared with the other sequences because the majority of point clouds (92 %) were degenerated or unavailable.

Note that the omnidirectional FOV LiDAR (Livox MID-360) and its IMU data were used only for estimating the reference trajectory as ground truth. We manually aligned point clouds measured by the omnidirectional FOV LiDAR with a previously prepared map built by a 3D laser scanner (FARO Focus3D X330). The ground truth trajectory was estimated by batch optimization based on the scan-to-map registration errors and IMU motion errors. We set initial values of $P_t$ (weights and biases of the *online learning model*) by using the corresponding values obtained via the offline learning procedure. In our implementation, we set

[1] https://youtu.be/ZkGO21Q_paY

**Table 3**

Ratio of frames facing degeneration and absence of the point clouds. Especially, *Seq. 1* was a difficult condition because the majority of point clouds (92 %) were degenerated or unavailable.

| Sequences/Problems | Point cloud degeneracy | | Point cloud absence | | Total time [s] |
|---|---|---|---|---|---|
| | Ratio to all frames | Duration [s] | Ratio to all frames | Duration [s] | |
| Seq. 1 | 69 % (572/828) | 57.2 | 23 % (189/828) | 18.9 | 85.7 |
| Seq. 2 | 26 % (110/419) | 11.1 | 2 % (9/419) | 0.9 | 41.9 |
| Seq. 3 | 30 % (227/760) | 22.7 | 8 % (60/760) | 6.0 | 76.3 |

**Table 4**

ATEs and RTEs of the odometry estimation algorithms. The proposed method (*Ours*) was twice as accurate as *Ours w/o online learning* owing to online training. *LIWO w/ linear model online calibration* and *FAST-LIO-Multi-Fusion* suffered from complex robot motions (e.g., large wheel slippage) because these methods regarded the kinematic model as linear. *FAST-LIO2* failed in all sequences because all environments included locations where point clouds degenerate.

| Method/Sequence | Seq. 1 | | Seq. 2 | | Seq.3 | |
|---|---|---|---|---|---|---|
| | ATE [m] | RTE [m] | ATE [m] | RTE [m] | ATE [m] | RTE [m] |
| Ours | **1.07** ± 0.35 | **0.08** ± 0.05 | **0.25** ± 0.10 | **0.12** ± 0.07 | **0.41** ± 0.18 | **0.09** ± 0.07 |
| Ours w/o online learning | 3.29 ± 2.42 | 0.09 ± 0.06 | 0.59 ± 0.53 | 0.16 ± 0.07 | 0.99 ± 0.51 | 0.11 ± 0.07 |
| LIWO w/ linear model online calibration [21] | 18.80 ± 14.40 | 0.08 ± 0.07 | 2.23 ± 2.16 | 0.12 ± 0.08 | 0.98 ± 0.50 | 0.09 ± 0.05 |
| FAST-LIO-Multi-Fusion [36] | 7.99 ± 3.10 | 0.36 ± 0.31 | 2.41 ± 2.17 | 0.49 ± 0.46 | 2.49 ± 1.63 | 0.17 ± 0.22 |
| FAST-LIO2 [34] | Error >20 m | Error >20 m | Error >20 m | Error >20 m | Error >20 m | Error >20 m |

these parameters trained for the indoor flat floor (i.e., #1 in Figure 6) as the initial value of $P_1$.

### 4.2.1. Comparison with state-of-the-art methods

We compared the proposed odometry estimation algorithm (*Ours*) with the following methods:

- *Ours w/o online learning*: We used the fixed neural network (i.e., the *Identical network* case) described in Section 4.1 instead of the proposed network for conducting an ablation study. We expected that the fixed network cannot adapt to a proper terrain condition.

- *LIWO w/ linear model online calibration* [21]: This method is our previous work, which is tightly-coupled LiDAR-IMU-wheel odometry with online calibration of the full linear model (kinematic model of a skid-steering robot). The full linear model cannot express nonlinearity (e.g., large wheel slippage).

- *FAST-LIO2* [34]: FAST-LIO2 is the state-of-the-art LiDAR-IMU odometry based on tightly-coupled LiDAR and IMU fusion, as described in Section 2.1.

- FAST-LIO-Multi-Sensor-Fusion [36]: Zhao et.al. fused wheel odometry-based constraints [36] and FAST-LIO2 [34] to improve estimation performance. They considered only longitudinal motion in the kinematic model, without taking into account lateral and rotational movements, even for a skid-steering robot. Furthermore, the wheel odometry-based constraint was directly defined by the longitudinal velocity of the robot, not the kinematic model (i.e., the wheel odometry-based constraint was incorporated by the loose coupling way).

Table 4 shows the absolute trajectory errors (ATEs) and relative trajectory errors (RTEs) [35] for all methods. We also show the trajectory comparison results in Figure 1. According to these results, in terms of ATEs and RTEs, the proposed method (*Ours*) outperformed all other methods in all sequences owing to the neural adaptive odometry factor, which can capture the nonlinearity of the kinematic model to adapt it to the current terrain condition. *FAST-LIO2* failed due to point cloud degeneration when the LiDAR pointed only at the wall in all sequences. *LIWO w/ linear model online calibration* and *FAST-LIO-Multi-Sensor-Fusion* suffered from nonlinearities such as drifting (executed at each corner of corridors in Figure 1-(a)) and successive left and right turning in rough terrain (executed in the grass, Figure 1-(c)) because these methods create constraints based on the linear kinematic model of the skid-steering robot. *Ours w/o online learning* approximately captured the nonlinearities compared with other comparative methods. However, *Ours w/o online learning* could not adapt to changes in terrain, and thus scales of the estimated trajectories were not consistent (especially, Figure 1-(b), (c)) in contrast to the proposed method with online learning. The ATEs of our method (Seq. 1: 1.07 m; Seq. 2: 0.25 m; Seq. 3: 0.41 m) were more than twice as accurate as a method incorporating a fixed network (i.e., batch learning rather than online learning)-based constraints (Seq. 1: 3.29 m; Seq. 2: 0.59 m; Seq. 3: 0.99 m) in all sequences.

In addition, Video 2 [2], Video 3 [3], and Video 4 [4] show supplementary material for understanding each terrain's characteristics and the robot's speed for Seq. 1, Seq. 2, and Seq. 3, respectively.

---

[2] https://youtu.be/IdvmBCDAtfw
[3] https://youtu.be/5BzZB0vO9jc
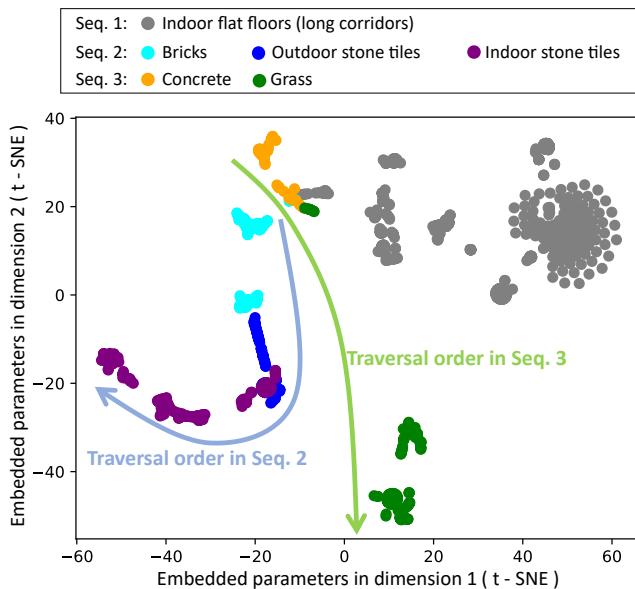[4] https://youtu.be/_TzNPNX9Gys

**Figure 9:** Visualization of the time histories of MLP parameters $P$ (100 dimensions) for all sequences with 2D t-SNE embedding. In *Seq. 1*, the robot moved on similar terrains consistently. *Seq. 2* included terrain changes from bricks to outdoor stone tiles to indoor stone tiles. *Seq. 3* also included changes in terrain conditions from asphalt to grass. The embedded parameters of grass (uneven and soft) were isolated from other terrains (flat and hard). Both Seq. 3 and Seq. 2 included terrain condition changes; however, the embedded parameters in Seq. 3 changed significantly compared to those in Seq. 2.

### 4.2.2. *Adaptability evaluation of the proposed network with online learning*

To verify adaptability of the proposed network online trained by the *neural adaptive odometry factor* with LiDAR-IMU-wheel odometry, we visualized the trained weight and bias of neurons in each sequence. We summarized those values by using t-distributed Stochastic Neighbor Embedding (t-SNE) [17], which embeds high-dimensional data into low-dimensional data through manifold learning. Figure 9 shows the embedded parameters (2 dimensions) from time histories of MLP parameters $P$ (100 dimensions) for all sequences. We can see that the embedded parameters for grass are isolated from the parameters of other terrains. This result is reasonable because the grass was uneven and soft terrain that was distinct from other flat and hard terrains. In the case of *Seq. 3* (transition from concrete to grass), the embedded parameters drastically changed during this transition because this difference between these terrains was significant. In contrast, in *Seq. 2* (transition from bricks to outdoor stone tiles to indoor stone tiles), the embedded parameters gradually changed because the differences between these terrains were slight. Similarly, in *Seq. 1* (indoor flat floors as in long corridors), the embedded parameters' behavior differed from those in each terrain to describe the proper terrain-dependent features.

Accordingly, these results indicate that the proposed network can adapt to the current terrain condition owing to online learning by *neural adaptive odometry factor*.

## 5. Conclusion

We proposed the LiDAR-IMU-wheel odometry algorithm incorporating online training of the neural network to infer the kinematic model of the skid-steering robot. We designed the proposed network to include an *offline learning model* and *online learning model* (Figure 5). The *offline learning model* was trained offline with datasets for eight types of terrain (Figure 6) to learn features related to terrain-independent parameters (Figure 7). In contrast, the *online learning model* was trained via the *neural adaptive odometry factor* with LiDAR-IMU-wheel odometry on a unified factor graph to ensure consistency with all those constraints.

We showed that the proposed network was more accurate than other networks not properly adapted to each terrain condition owing to our network design. We compared the proposed odometry estimation algorithm with the state-of-the-art methods [34, 36, 21]. The proposed method outperformed those methods owing to neural network-based constraints, which can express the complicated nonlinearity (e.g., large wheel slippage such as drifting) of the kinematic model of a skid-steering robot.

In future work, we plan to improve versatility of our network to adapt to any type of wheeled robot model (e.g., different size and mechanism from those used during the offline training phase) in addition to terrain condition changes. We need to enlarge the size of the network to enhance its expression by implementing the online adaptive odometry factor (i.e., online learning) with GPU acceleration, as in [10].

## References

[1] Anousaki, G., Kyriakopoulos, K.J., 2004. A dead-reckoning scheme for skid-steered vehicles in outdoor environments, in: 2004 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 580–585.

[2] Baril, D., Grondin, V., Deschênes, S.P., Laconte, J., Vaidis, M., Kubelka, V., Gallant, A., Giguere, P., Pomerleau, F., 2020. Evaluation of skid-steering kinematic models for subarctic environments, in: 2020 17th Conference on Computer and Robot Vision (CRV), IEEE. pp. 198–205.

[3] Brach, R., Brach, M., 2011. The tire-force ellipse (friction ellipse) and tire characteristics. Technical Report. SAE Technical Paper.

[4] Buchanan, R., Camurri, M., Dellaert, F., Fallon, M., 2022. Learning inertial odometry for dynamic legged robot state estimation, in: Conference on robot learning, PMLR. pp. 1575–1584.

[5] Chen, C., Lu, X., Markham, A., Trigoni, N., 2018. Ionet: Learning to cure the curse of drift in inertial odometry, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 6468–6476.

[6] Forster, C., Carlone, L., Dellaert, F., Scaramuzza, D., 2016. On-manifold preintegration for real-time visual–inertial odometry. IEEE Transactions on Robotics, 33 (1), pp. 1–21.

[7] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., Dellaert, F., 2012. isam2: Incremental smoothing and mapping using the bayes tree. The International Journal of Robotics Research, 31 (2), pp. 216–235.

[8] Kümmerle, R., Grisetti, G., Burgard, W., 2012. Simultaneous parameter calibration, localization, and mapping. Advanced Robotics, 26 (17), pp. 2021–2041.

[9] Kingma, D., Ba, J., 2015. Adam: A method for stochastic optimization, in: 2015 3rd International Conference on Learning Representations (ICLR), pp. 1–15.

[10] Koide, K., Yokozuka, M., Oishi, S., Banno, A., 2021a. Globally consistent 3d lidar mapping with gpu-accelerated gicp matching cost factors. IEEE Robotics and Automation Letters, 6 (4), pp. 8591–8598.

[11] Koide, K., Yokozuka, M., Oishi, S., Banno, A., 2021b. Voxelized gicp for fast and accurate 3d point cloud registration, in: 2021 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 11054–11059.

[12] Koide, K., Yokozuka, M., Oishi, S., Banno, A., 2022. Globally consistent and tightly coupled 3d lidar inertial mapping, in: 2022 International Conference on Robotics and Automation (ICRA), IEEE. pp. 5622–5628.

[13] Kümmerle, R., Grisetti, G., Burgard, W., 2011. Simultaneous calibration, localization, and mapping, in: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE. pp. 3716–3721.

[14] Le, A.T., Rye, D.C., Durrant-Whyte, H.F., 1997. Estimation of track-soil interactions for autonomous tracked vehicles, in: 1997 International Conference on Robotics and Automation (ICRA), IEEE. pp. 1388–1393.

[15] Lee, W., Eckenhoff, K., Yang, Y., Geneva, P., Huang, G., 2020. Visual-inertial-wheel odometry with online calibration, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE. pp. 4559–4566.

[16] Liu, W., Caruso, D., Ilg, E., Dong, J., Mourikis, A.I., Daniilidis, K., Kumar, V., Engel, J., 2020. Tlio: Tight learned inertial odometry. IEEE Robotics and Automation Letters, 5 (4), pp. 5653–5660.

[17] Van der Maaten, L., Hinton, G., 2008. Visualizing data using t-sne. Journal of machine learning research, 9 (11), pp. 2579–2605.

[18] Mandow, A., Martinez, J.L., Morales, J., Blanco, J.L., Garcia-Cerezo, A., Gonzalez, J., 2007. Experimental kinematics for wheeled skid-steer mobile robots, in: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE. pp. 1222–1227.

[19] Murai, R., Alzugaray, I., Kelly, P.H., Davison, A.J., 2024. Distributed simultaneous localisation and auto-calibration using gaussian belief propagation. IEEE Robotics and Automation Letters, 9 (3), pp. 2136–2143.

[20] Navone, A., Martini, M., Angarano, S., Chiaberge, M., 2023. Online learning of wheel odometry correction for mobile robots with attention-based neural network, in: 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), IEEE. pp. 1–6.

[21] Okawara, T., Koide, K., Oishi, S., Yokozuka, M., Banno, A., Uno, K., Yoshida, K., 2024. Tightly-coupled lidar-imu-wheel odometry with online calibration of a kinematic model for skid-steering robots. arXiv preprint arXiv:2404.02515 .

[22] Onyekpe, U., Palade, V., Herath, A., Kanarachos, S., Fitzpatrick, M.E., 2021a. Whonet: Wheel odometry neural network for vehicular localisation in gnss-deprived environments. Engineering Applications of Artificial Intelligence, 105, pp. 1–19.

[23] Onyekpe, U., Palade, V., Kanarachos, S., 2021b. Learning to localise automated vehicles in challenging environments using inertial navigation systems (ins). Applied Sciences, 11 (3), pp. 1–23.

[24] Ordonez, C., Gupta, N., Reese, B., Seegmiller, N., Kelly, A., Collins Jr, E.G., 2017. Learning of skid-steered kinematic and dynamic models for motion planning. Robotics and Autonomous Systems, 95, pp. 207–221.

[25] Qin, C., Ye, H., Pranata, C.E., Han, J., Zhang, S., Liu, M., 2020. Lins: A lidar-inertial state estimator for robust and efficient navigation, in: 2020 IEEE International Conference on Robotics and Automation (ICRA), IEEE. pp. 8899–8906.

[26] Qin, T., Li, P., Shen, S., 2018. Vins-mono: A robust and versatile monocular visual-inertial state estimator. IEEE Transactions on Robotics, 34 (4), pp. 1004–1020.

[27] Rabiee, S., Biswas, J., 2019. A friction-based kinematic model for skid-steer wheeled mobile robots, in: 2019 International Conference on Robotics and Automation (ICRA), IEEE. pp. 8563–8569.

[28] Reichensdörfer, E., Odenthal, D., Wollherr, D., 2018. On the stability of nonlinear wheel-slip zero dynamics in traction control systems. IEEE Transactions on Control Systems Technology, 28 (2), pp. 489–504.

[29] Savage, P.G., 1998. Strapdown inertial navigation integration algorithm design part 1: Attitude algorithms. Journal of guidance, control, and dynamics, 21 (1), pp. 19–28.

[30] Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., Rus, D., 2020. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping, in: 2020 IEEE/RSJ international conference on intelligent robots and systems (IROS), IEEE. pp. 5135–5142.

[31] Teschner, M., Heidelberger, B., Müller, M., Pomerantes, D., Gross, M.H., 2003. Optimized spatial hashing for collision detection of deformable objects., in: Vmv, pp. 47–54.

[32] Van Nam, D., Gon-Woo, K., 2022. Learning observation model for factor graph based-state estimation using intrinsic sensors. IEEE Transactions on Automation Science and Engineering, 20 (3), pp. 2049–2062.

[33] Wang, T., Wu, Y., Liang, J., Han, C., Chen, J., Zhao, Q., 2015. Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor. Sensors, 15 (5), pp. 9681–9702.

[34] Xu, W., Cai, Y., He, D., Lin, J., Zhang, F., 2022. Fast-lio2: Fast direct lidar-inertial odometry. IEEE Transactions on Robotics, 38 (4), pp. 2053–2073.

[35] Zhang, Z., Scaramuzza, D., 2018. A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE. pp. 7244–7251.

[36] Zhao, H., Ji, X., Wei, D., 2023. Vehicle-motion-constraint-based visual-inertial-odometer fusion with online extrinsic calibration. IEEE Sensors Journal, 23 (22), pp. 27895–27908.

[37] Zuo, X., Zhang, M., Chen, Y., Liu, Y., Huang, G., Li, M., 2019. Visual-inertial localization for skid-steering robots with kinematic constraints, in: The International Symposium of Robotics Research, Springer. pp. 741–756.

[38] Zuo, X., Zhang, M., Wang, M., Chen, Y., Huang, G., Liu, Y., Li, M., 2022. Visual-based kinematics and pose estimation for skid-steering robots. IEEE Transactions on Automation Science and Engineering, 21 (1), pp. 91–105.