ROBOMECH Journal

RESEARCH ARTICLE Open Access

LiDAR DNN based self-attitude estimation with learning landscape regularities

Ryota Ozaki*, Naoya Sugiura and Yoji Kuroda

Abstract
This paper presents an EKF (extended Kalman filter) based self-attitude estimation method with a LiDAR DNN (deep neural network) learning landscape regularities. The proposed DNN infers the gravity direction from LiDAR data. The point cloud obtained with the LiDAR is transformed to a depth image to be input to the network. It is pre-trained with large synthetic datasets. They are collected in a flight simulator because various gravity vectors can be easily obtained, although this study focuses not only on UAVs. Fine-tuning with datasets collected with real sensors is done after the pre-training. Data augmentation is processed during the training in order to provide higher general versatility. The proposed method integrates angular rates from a gyroscope and the DNN outputs in an EKF. Static validations are performed to show the DNN can infer the gravity direction. Dynamic validations are performed to show the DNN can be used in real-time estimation. Some conventional methods are implemented for comparison.

Keywords: Attitude estimation, Mobile robotics, Deep learning, Extended Kalman filter

Introduction

Estimating the attitude of a robot is one of the classic problems in mobile robotics. In particular, real-time estimation is required for real-time attitude control. The attitude is generally estimated with inertial sensors such as accelerometers and gyroscopes. However, mobile robots have their own acceleration. Moreover, on-road robots also receive pulses from the ground, and UAVs suffer from vibration of their multi-rotor. These need to be filtered out from the accelerometer. On the other hand, integration of gyroscopic angular rate has problems of drift and bias. These disturbances worsen the accuracy of the estimation. To complement each other, these inertial data are fused, generally [1]. Nevertheless, dealing with the disturbances with only inertial sensors is quite difficult.

To reduce the influence of these disturbances, many kinds of LiDAR odometry, VO (visual odometry) and SLAM (simultaneous localization and mapping) [2] have been proposed. LiDAR-based methods register point clouds by ICP [3], NDT [4], and so on. Visual methods often track features in image sequences [5, 6]. However, these odometry methods and SLAMs often contain accumulative error since relative pose changes with error are summed up. In order to correct the accumulative error, prior information such as 3D maps is often used [7]. These methods correct the error by matching the prior information against the sensor data. However, they work only in environments where maps are available. Moreover, creating a map is time-consuming, and updating is also required. Some methods [8–10] estimate the attitude under Manhattan world assumption. They assume that planes or edges in the environment are orthogonal to each other. It helps achieving drift-free estimation. However, it is difficult for this kind of method to avoid being affected by objects which do not satisfy the assumption. We presented a method in [11] using looser restraints than Manhattan world assumption. It exploited a regularity which most buildings are built vertically. Therefore, the method can be applied not only to Manhattan world, but also to environments where planes are not orthogonal to each other. Vertical planes are extracted from

*Correspondence: ce192021@meiji.ac.jp
Graduate School of Science and Technology, Meiji University, 1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa 214-8571, Japan

LiDAR point cloud, and the gravity direction is estimated based on normals of the planes. However, the method misses many features even though there are more regularities in environments, since it uses only completely flat planes. Besides, it requires tuning many hyperparameters such as the searching radius for PCA (principal component analysis), and the threshold for the number of the normals, and the threshold for the clustering angle. In addition, the processing of the 3D point clouds takes much time.

Deep learning has been used for attitude estimation in recent years. In [12], IMU-based odometry by end-to-end learning has been proposed. In [13], a deep neural network identifies the measurement noise characteristics of IMU. In [14], a neural network estimates angular rates from sequential images. It was trained with synthetic and real images. The large synthetic dataset was collected in AirSim [15] which offers visually realistic graphics. In [16], a gravity vector is directly estimated from a single shot image. The method can infer the orientation at which a picture was taken simply by looking at the picture, like a human. It implies there are regularities between the gravity direction and landscapes. Since the method is a camera-based one, it is cost effective, but it does not work well in nighttime.

To address these issues above, we present an EKF-based self-attitude estimation method with a LiDAR DNN learning the landscape regularities. By using a LiDAR, the method can estimate the attitude regardless of day or night. Moreover, problems of rule-based methods such as [11] described above can be solved by using deep learning. The main contributions of this paper are summarized below:

- A LiDAR-based DNN which infers an absolute gravity direction which does not contain the robot's own acceleration nor vibration is proposed. Unlike camera-based methods, the DNN can estimate it regardless of day or night.
- The DNN inference is integrated with a gyroscope for real-time estimation.
- A data transformation and augmentation method for the LiDAR and IMU data is proposed. Converting the 3D point cloud into a 2D depth image reduces the computation time in the DNN.
- Pre-training the DNN with large synthetic data before fine-tuning it with real data makes the learning efficient, while the related work [16] uses only real one.

The datasets and the source code used in this paper have been released in open repositories (see 'Availability of data and materials').

DNN estimating gravity vector

The proposed method makes the DNN learn landscape regularities for estimating the gravity vector.

Coordinate definition

A world frame is defined as a standard right-handed coordinate system. A robot frame is defined as a right-handed coordinate system which is fixed on the robot pose. Its x axis is on the robot's heading direction. They are shown in Fig. 1.

Dataset collection

Both synthetic and real data are collected in this study. The synthetic datasets are used for pre-training, and the real ones are used for fine-tuning. The datasets consist of LiDAR data and corresponded gravity vectors $\boldsymbol{g}$ in the robot frame. Technically, lidar data represents 2D depth images transformed from 3D point clouds. The details about the transformation are described in the next section. Fig. 2 shows examples of the datasets.

The synthetic data is collected in AirSim [15]. AirSim is a simulator for drones, cars and more, built on Unreal Engine, which provides visually realistic graphics. An IMU and a LiDAR which has 32 layers are installed to a drone in the simulator. The robot pose and weather parameters are randomized, and the LiDAR data and a gravity vector are recorded at each pose. The range of the random Z is limited as [2 m, 3 m] in this work. The ranges of the random roll ϕ and pitch θ are limited as [−30 deg, 30 deg], respectively. The reason for limiting Z, ϕ and θ is that this study mainly focuses on on-road robots that tilt their bodies on which a LiDAR can be installed.

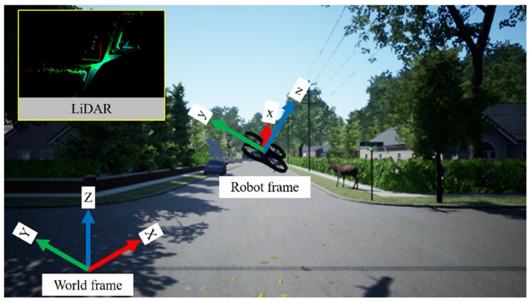


Fig. 1 Screenshot of AirSim with coordinate description. An IMU and a LiDAR are equipped to the drone in the simulator. The purpose of the proposed DNN is estimating a gravity vector in the robot frame from LiDAR data

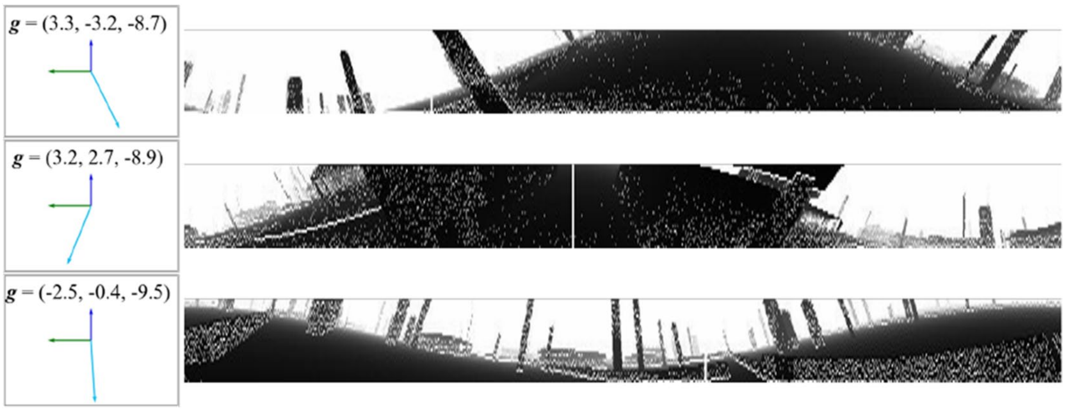


Fig. 2 Examples of datasets. The datasets consist of depth images and corresponded gravity vectors $\boldsymbol{g}[\mathrm{m/s^2}]$ in the robot frame

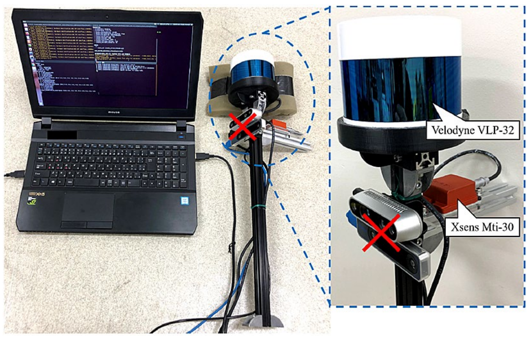


Fig. 3 Sensors installed on stick. Point clouds and acceleration are recorded with this stick when it is still. The judge whether it is static is processed by programing

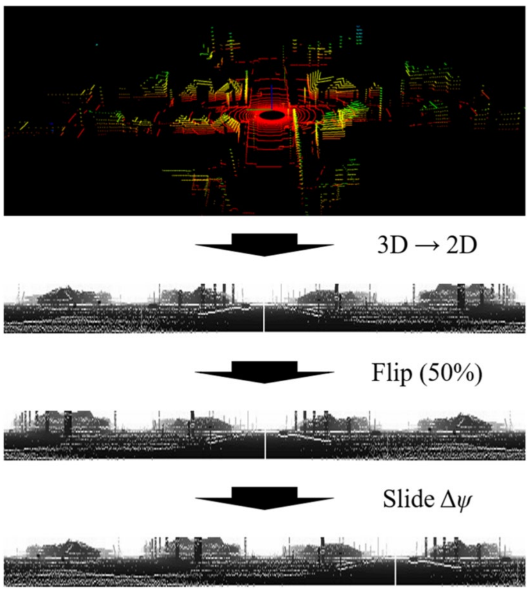


Fig. 4 Preprocessing scheme. Each 3D point cloud is transformed to a 2D depth image. The depth image is randomly flipped and slid. Flipping, rotation, and L2 normalization are also applied to the gravity vector. In this example, the virtual yaw variant is $\Delta\psi = 90$ deg

The real datasets are collected with an IMU (Xsens MTi-30) and a LiDAR (Velodyne VLP-32) installed on a top of a stick (Fig. 3). The stick is hand-carried, and point clouds and linear acceleration vectors are recorded. They are saved only when the stick is shaking less than 0.001 m and 0.1 deg in 0.5 s, and when it is at least 5 deg away from the last saved pose. The IMU is regarded as ground truth because it has enough accuracy (within 0.2 deg) in static according to the specification. Learning the static IMU is valuable because the DNN can reproduce it even in dynamic.

Data preprocessing

Each input and label data are transformed, and are augmented in each epoch of training. Data augmentation is especially important for real data because collecting real data is time-consuming. Figure 4 shows an example of data transformation.

LiDAR data transformation

The point clouds obtained with the LiDAR are transformed to depth images as follows.

$$
\begin{aligned}
\text{pixel}[row, col] &= \begin{cases} \sqrt{p_{i,x}^2 + p_{i,y}^2} \\ -1 & \text{(no return)} \end{cases} \\
row &= \frac{\text{FOV}^{\text{upper}} - \tan^{-1} \frac{p_{i,z}}{\sqrt{p_{i,x}^2+p_{i,y}^2}}}{\text{res}^{\text{v}}} \\
\text{res}^{\text{v}} &= \frac{\text{FOV}^{\text{upper}} - \text{FOV}^{\text{lower}}}{\#\text{row} - 1} \\
col &= (\#\text{col} - 1) - \frac{\tan^{-1} \frac{p_{i,y}}{p_{i,x}} + \pi}{\text{res}^{\text{h}}} \\
\text{res}^{\text{h}} &= \frac{2\pi}{\#\text{col}}
\end{aligned} \tag{1}
$$

where pixel$[row, col]$ denotes a pixel value at $[row, col]$ of the depth image, $\boldsymbol{p}_i$ denotes a point in the cloud, FOV denotes the vertical field-of-view of the LiDAR, res$^{\text{v}}$ and res$^{\text{h}}$ denote the angle resolution of the LiDAR, and #row, #col denote the numbers of the pixels in each row and col, respectively.

Each generated depth image is flipped in 50% of probability for augmenting the data. After the flipping process, the pixels of each depth image are randomly slid horizontally. The virtual yaw variant $\Delta\psi$ is computed as below.

$$
\Delta\psi = 2\pi \frac{\Delta\text{col}}{\#\text{col}} \tag{2}
$$

where Δcol denotes the number of the slid pixels.

IMU data transformation

The gravity vector is also flipped and rotated according to $\Delta\psi$. Since the network does not need to learn the norm of the gravity, L2 normalization is applied to the vector in order to make the training efficient.

$$
\begin{aligned}
\hat{\boldsymbol{g}} &= \begin{cases} \boldsymbol{Rot}^{\text{z}}_{(-\Delta\psi)} \frac{\boldsymbol{g}}{|\boldsymbol{g}|} & \text{(w/o flip)} \\ \boldsymbol{Rot}^{\text{z}}_{(-\Delta\psi)} \frac{(g_x, -g_y, g_z)^{\text{T}}}{|\boldsymbol{g}|} & \text{(w/ flip)} \end{cases} \\
\boldsymbol{Rot}^{\text{z}}_{(\Delta\psi)} &= \begin{pmatrix} \cos(\Delta\psi) & -\sin(\Delta\psi) & 0 \\ \sin(\Delta\psi) & \cos(\Delta\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix}
\end{aligned} \tag{3}
$$

Network

The proposed DNN is shown in Fig. 5. It consists of CNN (convolutional neural network) layers and FC (fully connected) layers. The input to the network is the depth image, and the output is a gravity vector $\boldsymbol{\mu}$. Technically, the output of the FC layers is normalized. The CNN layers are expected to learn extracting features such as edges and planes. The FC layers are expected to learn the regularities between the features and the gravity direction.

$$
\hat{\boldsymbol{\mu}} = \frac{(\mu_x, \mu_y, \mu_z)^{\text{T}}}{|(\mu_x, \mu_y, \mu_z)^{\text{T}}|} \tag{4}
$$

It is expected that the CNN layers learn extracting features such as edges and planes, and the FC layers learn landscape regularities. All layers, except the final output layer, use the ReLU function [17] as an activation function. All FC layers, except the final output layer, use the 10% Dropout [18] to avoid the over-fitting problem.

Loss function

The MSE (mean square error) between the outputs and labels is used as a loss function of this model.

$$
l_{(\boldsymbol{\Theta})} = \frac{1}{\#D} \sum_{\iota=0}^{\#D} |\hat{\boldsymbol{g}}_\iota - \hat{\boldsymbol{\mu}}_\iota|^2 \tag{5}
$$

where $\boldsymbol{\Theta}$ denotes the parameters of the network, and $\#D$ denotes the number of samples. The network minimizes the loss by updating $\boldsymbol{\Theta}$.

Optimization

Adam (adaptive moment estimation) [19] is used to optimize the parameters. For the training with the synthetic data, the learning rates are set as $lr_{\text{CNN}} = 0.00001$, $lr_{\text{FC}} = 0.0001$, where lr_{CNN} is a value for the CNN layers, lr_{FC} is a value for the FC layers. For the fine-tuning with real data, they are set smaller as $lr_{\text{CNN}} = 0.000001$, $lr_{\text{FC}} = 0.00001$.

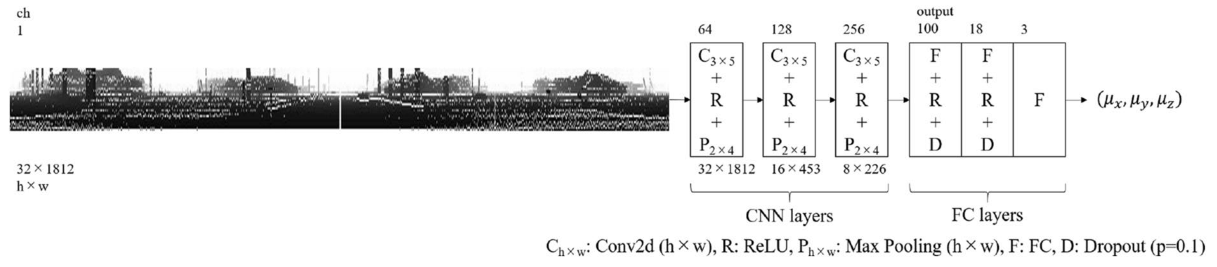


Fig. 5 Proposed network architecture. It consists of CNN layers and FC layers. The MSE between the outputs and the labels is used as a loss function of this model

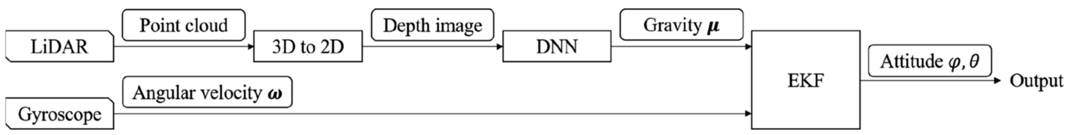

LiDAR
Point cloud
3D to 2D
Depth image
DNN
Gravity $\boldsymbol{\mu}$
EKF
Attitude φ, θ
Output
Gyroscope
Angular velocity $\boldsymbol{\omega}$

Fig. 6 Proposed EKF architecture. Gyroscopic angular rates are integrated in the prediction process in the EKF. The DNN outputs are integrated in the update process in the EKF

EKF-based real-time estimation

The outputs from the DNN are integrated with gyroscopic angular rate in an EKF. The proposed EKF architecture is shown in Fig. 6. It is based on [16] which simplifies the attitude estimator in [20]. The state vector $\boldsymbol{x}$ of the proposed Kalman filter consists of the roll ϕ and pitch θ of the robot pose.

$$\boldsymbol{x} = \begin{pmatrix}\phi & \theta\end{pmatrix}^{\mathrm{T}} \tag{6}$$

Both of the vector $\boldsymbol{x}$ and the covariance matrix $\boldsymbol{P}$ are computed in a prediction process and an update process. The prediction process is computed by integrating angular velocity from a gyroscope. The update process is computed by observing the outputs of the DNN. Note that the covariance matrices for the prediction and observation are determined experimentally. Here, t denotes the time step, S_ϕ, C_ϕ, T_ϕ are short for $\sin\phi$, $\cos\phi$, $\tan\phi$, respectively in the following sections.

Prediction process

The state vector $\boldsymbol{x}$ and the covariance matrix $\boldsymbol{P}$ are respectively computed as follows.

$$\begin{aligned}
\bar{\boldsymbol{x}}_t &= f_{(\boldsymbol{x}_{t-1},\boldsymbol{u}_{t-1})} = \boldsymbol{x}_{t-1} + \boldsymbol{Rot}^{\mathrm{rpy}}_{(\boldsymbol{x}_{t-1})}\boldsymbol{u}_{t-1}\\
\boldsymbol{u}_{t-1} &= \boldsymbol{\omega}_{t-1}\Delta t = \begin{pmatrix}\omega_{\mathrm{x}_{t-1}}\Delta t\\ \omega_{\mathrm{y}_{t-1}}\Delta t\\ \omega_{\mathrm{z}_{t-1}}\Delta t\end{pmatrix}\\
\boldsymbol{Rot}^{\mathrm{rpy}}_{(\boldsymbol{x}_{t-1})} &= \begin{pmatrix}1 & S_{\phi_{t-1}}T_{\theta_{t-1}} & C_{\phi_{t-1}}T_{\theta_{t-1}}\\ 0 & C_{\phi_{t-1}} & -S_{\phi_{t-1}}\end{pmatrix}
\end{aligned} \tag{7}$$

where f is a state transition model, $\boldsymbol{u}$ denotes a control vector, $\boldsymbol{\omega}$ denotes the angular velocity measured with a gyroscope, and $\boldsymbol{Rot}^{\mathrm{rpy}}$ denotes a rotation matrix for angular velocities.

$$\bar{\boldsymbol{P}}_t = \boldsymbol{J}_{f\,t-1}\boldsymbol{P}_{t-1}\boldsymbol{J}_{f\,t-1}^{\mathrm{T}} + \boldsymbol{Q}_{t-1}, \quad \boldsymbol{J}_{f\,t-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{t-1},\boldsymbol{u}_{t-1}} \tag{8}$$

where $\boldsymbol{J}_f$ denotes f Jacobean, and $\boldsymbol{Q}$ denotes a covariance matrix of the process noise.

Update process

The observation vector is $\boldsymbol{z}$ as below.

$$\boldsymbol{z} = \hat{\boldsymbol{\mu}} \tag{9}$$

where $\hat{\boldsymbol{\mu}}$ denotes a gravity which is output from the DNN. The observation model is h.

$$\begin{aligned}
&h_{(\boldsymbol{x}_t)} = \boldsymbol{Rot}^{\mathrm{xyz}}_{(-\boldsymbol{x}_t)}\frac{\boldsymbol{g}_{\mathbf{world}}}{|\boldsymbol{g}_{\mathbf{world}}|}, \quad \boldsymbol{g}_{\mathbf{world}} = \begin{pmatrix}0\\0\\g_{\mathrm{world}}\end{pmatrix}\\
&\boldsymbol{Rot}^{\mathrm{xyz}}_{(\boldsymbol{x}_t)} =\\
&\begin{pmatrix}C_{\theta_t}C_{\psi_t} & S_{\phi_t}S_{\theta_t}C_{\psi_t} - C_{\phi_t}S_{\psi_t} & C_{\phi_t}S_{\theta_t}C_{\psi_t} + S_{\phi_t}S_{\psi_t}\\ C_{\theta_t}S_{\psi_t} & S_{\phi_t}S_{\theta_t}S_{\psi_t} + C_{\phi_t}C_{\psi_t} & C_{\phi_t}S_{\theta_t}S_{\psi_t} - S_{\phi_t}C_{\psi_t}\\ -S_{\theta_t} & S_{\phi_t}C_{\theta_t} & C_{\phi_t}C_{\theta_t}\end{pmatrix}
\end{aligned} \tag{10}$$

where $\boldsymbol{g}_{\mathbf{world}}$ denotes a gravity vector in the world frame i.e. $g_{\mathrm{world}} \fallingdotseq 9.8\ \mathrm{m/s^2}$, $\boldsymbol{Rot}^{\mathrm{xyz}}$ denotes a rotation matrix for vectors. The state vector $\boldsymbol{x}$ and the covariance matrix $\boldsymbol{P}$ are respectively computed as follows.

$$\begin{aligned}
&\check{\boldsymbol{x}}_t = \boldsymbol{x}_t + \boldsymbol{K}_t(\boldsymbol{z}_t - h_{(\boldsymbol{x}_t)}), \quad \check{\boldsymbol{P}}_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{J}_{\boldsymbol{h}t})\boldsymbol{P}_t\\
&\boldsymbol{J}_{\boldsymbol{h}t} = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_t}, \quad \boldsymbol{K}_t = \boldsymbol{P}_t\boldsymbol{J}_{\boldsymbol{h}t}^{\mathrm{T}}(\boldsymbol{J}_{\boldsymbol{h}t}\boldsymbol{P}_t\boldsymbol{J}_{\boldsymbol{h}t}^{\mathrm{T}} + \boldsymbol{R})^{-1}
\end{aligned} \tag{11}$$

where $\boldsymbol{J}_{\boldsymbol{h}}$ denotes h Jacobean, $\boldsymbol{K}$ denotes a gain matrix, $\boldsymbol{R}$ denotes the covariance matrix of the process noise, and $\boldsymbol{I}$ denotes an identity matrix.

Validation

Static and dynamic experiments were performed on both synthetic and real data.

Static validation of DNN

The proposed DNN was trained with training datasets, and was evaluated with test datasets.

Table 1 Dataset list

id#	Environment		#Samples	Usage
1	Sim.	AirSim's	10000	Training
2	---"---	'Neighborhood'	1000	Test
3	Real	Area-I	1941	Fine-tuning
4	---"---	Area-II (daytime)	443	Test
5	---"---	Area-II (nighttime)	447	Test
6	---"---	Slope	327	Test

Table 2 Loss after 200 epochs of training

MSE [m^2/s^4]	Training (#1)	Test (#2)
LiDAR DNN (ours)	0.0028	**0.0015**
Camera DNN	**0.0014**	0.0033

Method list

Definitions of methods which were used in this validation are summarized here.

- LiDAR DNN (ours): 'LiDAR DNN (ours)' denotes the proposed method described in the section above.
- Camera DNN: 'Camera DNN' denotes a DNN where the input to it is a color image, and the output is a gravity vector. Its CNN module is the same feature module as VGG16 [21], which means this network is almost the same as the related work [16].
- Statistics: 'Statistics' denotes a method using the average of the label vectors as outputs for all samples, which means $\sum_{\iota=0}^{\#D} \boldsymbol{g}_\iota$ is used for estimating attitudes of all samples. Computing the error of this method is equivalent to calculating the standard deviation of the dataset. This method is regarded as the baseline in this study.

Training

The datasets used in this validation are listed in Table 1. The DNN was trained with 10000 synthetic samples (Dataset#1) with a batch size of 200 samples for 200 epochs. Another 1000 samples (Dataset#2) were used for test. They were collected in 'Neighborhood' of AirSim. The training dataset and the test dataset were not mixed. A computer which has W-2133 CPU and Quadro GV100 GPU with 32 GB memory was used for the training. The training took about 1.1 h with the computer.

The loss values during the training are plotted in Fig. 7(a). Table 2 shows the loss values after 200 epochs of training.

Fine-tuning

Fine-tuning with the real data was done after the training with the synthetic data. The DNN was tuned with 1941 real data samples (Dataset#3) with a batch size of 200 samples for 200 epochs. Another 1217 samples (Dataset#4–6) were used for test. Dataset#3–5 were collected in the same campus of Meiji University, but not in the same area. Dataset#6 was collected on a slope without surrounding buildings. Figure 8 shows pictures of one part of each environment.

The loss values during the fine-tuning are plotted in Fig. 7(b). Table 3 shows the loss values after 200 epochs of the fine-tuning. The loss value on the real dataset became smaller by the fine-tuning. However the loss value on the test dataset is larger than one on the training dataset. To reduce the difference of the results between the training data and the test data, a wider variety of datasets are needed for training.

Attitude estimation

The MAE (mean absolute error) and the variance of the static estimation are computed as below.

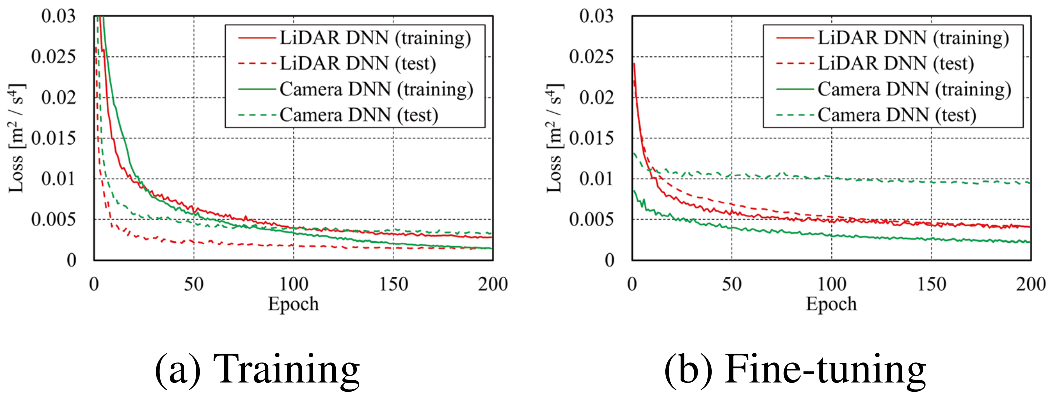


Fig. 7 Loss plotting. The DNNs converged the loss values by deep learning. The fine-tuning made the loss values on the real data smaller

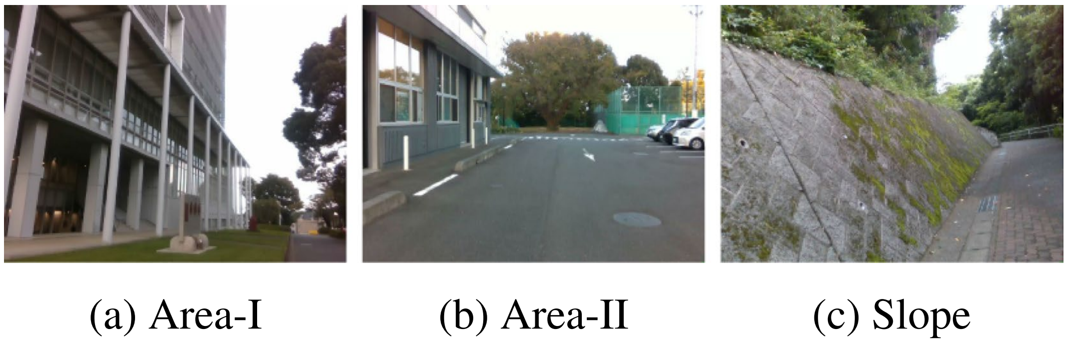


Fig. 8 Environments for collecting datasets. Dataset#3 was collected in (**a**). Dataset#4, 5 were collected in (**b**). Dataset#6 was collected in (**c**)

Table 3 Loss after 200 epochs of fine-tuning

MSE [m^2/s^4]	Training	Test	
	(#3)	(#4+#5)	(#6)
LiDAR DNN (ours)	0.0040	**0.0041**	0.0337
Camera DNN	**0.0023**	0.0095	**0.0239**

Bold value represents the best result in each experiment/validation

Table 4 MAE and variance of static estimation on synthetic data

MAE [deg] {Var. [deg^2]}	Dataset#	
	1	2
LiDAR DNN (ours)	**1.82** {9.87}	**1.86** {11.41}
Camera DNN	3.57 {24.98}	3.18 {19.83}
Statistics	22.91 {71.54}	22.82 {71.74}

Bold value represents the best result in each experiment/validation

Table 5 MAE and variance of static estimation on real data

MAE [deg] {Var. [deg^2]}		Dataset#			
		3	4	5	6
Before fine-tuning	LiDAR DNN (ours)	11.68 {108.84}	13.71 {105.20}	11.29 {77.18}	24.31 {157.94}
---"---	Camera DNN	6.67 {25.92}	6.42 {22.90}	8.72 {83.84}	14.95 {38.24}
After fine-tuning	LiDAR DNN (ours)	**4.08** {6.88}	5.27 {8.65}	**5.72** {11.80}	16.60 {61.97}
---"---	Camera DNN	4.82 {18.13}	**4.83** {16.03}	5.83 {55.13}	**14.28** {33.51}
Statistics		23.61 {96.13}	22.28 {98.97}	27.33 {86.82}	15.91 {85.83}

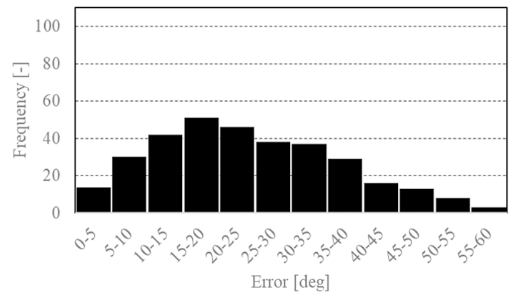
Bold value represents the best result in each experiment/validation

$$e_\iota = \cos^{-1}\frac{\hat{\boldsymbol{g}}_\iota \cdot \hat{\boldsymbol{\mu}}_\iota}{|\hat{\boldsymbol{g}}_\iota||\hat{\boldsymbol{\mu}}_\iota|}$$

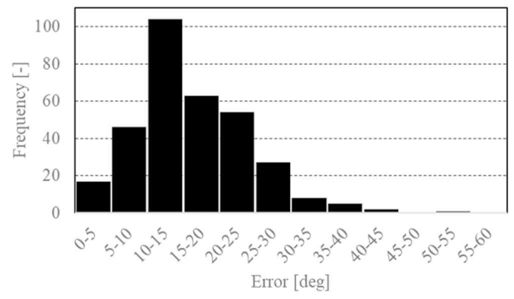
$$MAE = \frac{1}{\#D}\sum_{\iota=0}^{\#D} e_\iota, \quad Var. = \frac{1}{\#D}\sum_{\iota=0}^{\#D} |e_\iota - MAE|^2 \tag{12}$$

The MAE and the variance of the estimation on the synthetic datasets is shown in Table 4. Those on the real datasets are shown in Table 5. Both of 'LiDAR DNN (ours)' and 'Camera DNN' inferred the attitude with small errors, except with Dataset#6. The DNNs did not work well on Dataset#6. The distribution of error in Fig. 9(a) implies that the LiDAR DNN before fine-tuning underfit the dataset. On the other hand, Fig. 9(b) implies that the inferences given by LiDAR DNN after fine-tuning are biased. We consider it had an illusion that the slope is horizontal because the training datasets do not contain many samples of slopes. It might also have an illusion that the wall (in Fig. 8(c)) is vertical. Therefore, it should be noted that the DNNs do not perform well in situations which are not contained in the datasets. At the point in Fig. 8(c), the ground truth is $\phi_{gt} = 9.23$ deg, $\theta_{gt} = -11.51$ deg, and the inference given by LiDAR DNN after fine-tuning is $\phi_{est} = 3.83$ deg, $\theta_{est} = 1.50$ deg. According to this result, establishing a way to collect more data including slopes is our future work. On the other hand, the DNNs perform well even in unknown environments when regularities such as vertical buildings exist.

Focusing on the data collected in the nighttime (dataset #5), 'Camera DNN' showed a good result thanks to streetlights, but it had larger error than in the daytime. The variance is also much larger. On the other hand, 'LiDAR DNN' is not affected by the light condition.

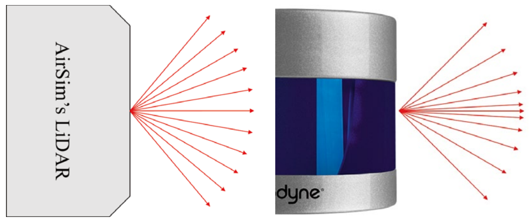


(a) Underfitting (LiDAR DNN before fine-tuning on Dataset#6)



(b) Biased inferences (LiDAR DNN after fine-tuning on Dataset#6)

Fig. 9 Distribution of inference error



(a) Linear spacing FOV (b) Non-linear spacing FOV

Fig. 10 LiDAR FOV. The LiDAR in the simulator does not reproduce non-linear spacing of FOV which the real sensor has

Comparing 'before fine-tuning' and 'after fine-tuning', the fine-tuning with the real datasets makes the error smaller. The number of the samples for the fine-tuning is not large, but it worked enough. It implies the pre-training with the large synthetic dataset is valid. Before the fine-tuning, the error of 'LiDAR DNN' is large. It is considered to be because the LiDAR in the simulator does not fully reproduce specifications of the real one, such as non-linear spacing of FOV (Fig. 10). This problem is solved by the fine-tuning.

Validation of real-time estimation in simulator

The proposed EKF-based real-time estimation was validated on synthetic flight data of a drone since ground truth is available in the simulator. Videos of the experiments have been released in public (see 'Availability of data and materials').

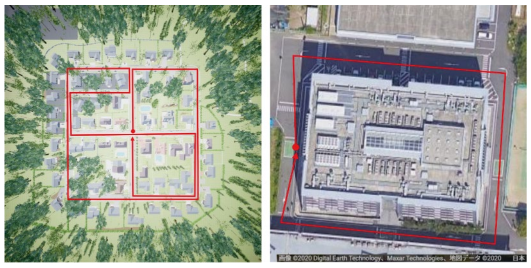
Method list

Definitions of methods which were used in this validation are summarized here.

- Gyro: 'Gyro' denotes an estimation method integrating angular velocity from a gyroscope.
- Gyro+Acc: 'Gyro+Acc' denotes an EKF-based estimation method integrating angular velocity and linear acceleration from an IMU.
- Gyro+NDT: 'Gyro+NDT' denotes NDT SLAM [4] using 32 layers of LiDAR. Angular velocity from a gyroscope, linear velocity of ground truth, and the NDT output are integrated in an EKF. Note that linear velocity of the ground truth is available because the environment is a simulator.
- Gyro+DGSphere [11]: 'Gyro+DGSphere' denotes a method described in [11]. Vertical planes are extracted from the LiDAR point cloud, and the cross product of the planes' normals is used as an estimated gravity direction. 'DGSphere' is short for 'depth-Gaussian sphere'.
- DNN: 'DNN' denotes a method using the proposed DNN directly without EKF.
- Gyro+DNN (ours): 'Gyro+DNN (ours)' denotes the proposed method described in the section above.

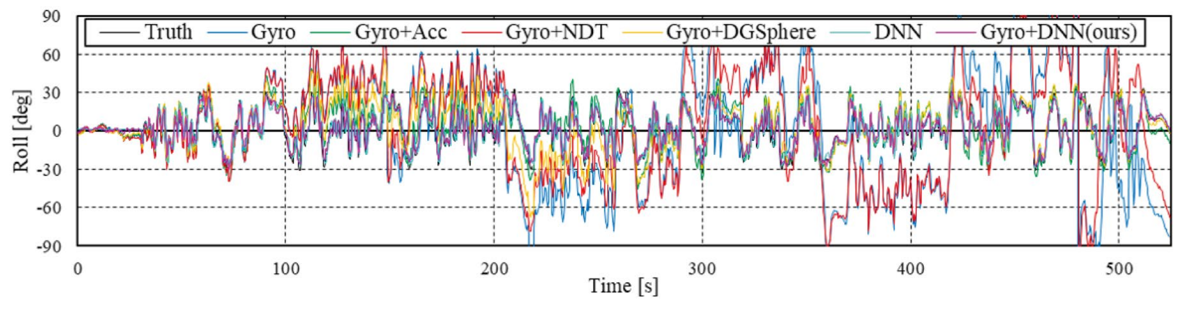
Experimental conditions

Flight data of a drone was recorded in 'Neighborhood' of AirSim. The sampling frequency of the IMU and the LiDAR are approximately 100 Hz, 20 Hz, respectively.

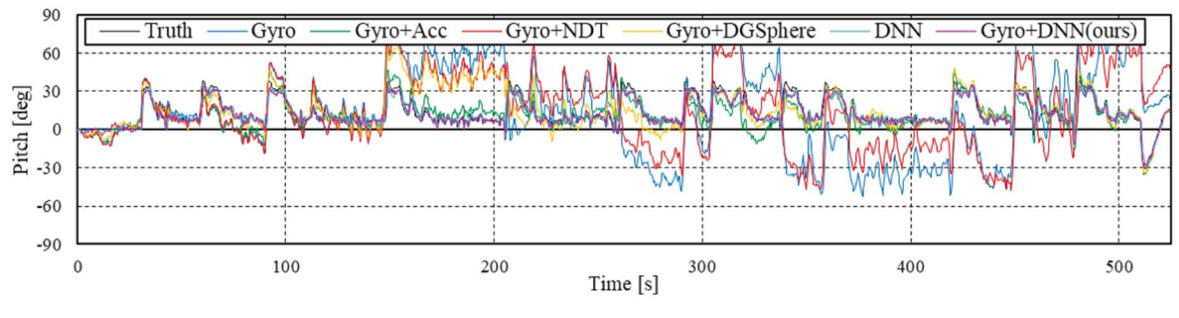


(a) Simulator (b) Real world

Fig. 11 Driving courses. The AirSim's drone flew for about 9 min in (**a**). The sensors were carried for about 5 min in (**b**)



(a) Roll



(b) Pitch

Fig. 12 Real-time plotting in 'Neighborhood'. 'Gyro+DNN (ours)' suppressed accumulative error by observing the DNN outputs

Table 6 MAE of dynamic estimation in simulator

	Roll [deg]	Pitch [deg]
Gyro	36.786	28.473
Gyro+Acc	6.451	5.387
Gyro+NDT	32.514	23.995
Gyro+DGSphere [11]	9.272	7.534
DNN	3.148	1.748
Gyro+DNN (ours)	**2.865**	**1.973**

Bold value represents the best result in each experiment/validation

Table 7 MAE of dynamic estimation in mocap area

	Roll [deg]	Pitch [deg]
Gyro	6.012	5.100
Gyro+Acc	2.509	**1.648**
Gyro+DGSphere	4.272	3.147
DNN	6.153	3.494
Gyro+DNN (ours)	**2.506**	1.854

Bold value represents the best result in each experiment/validation

Virtual noise was added to the IMU's 6-axis data. It was randomly added following a normal distribution with a mean of 0 rad/s, 0 m/s^2 and a standard deviation of 0.5 rad/s, 0.5 m/s^2, respectively. Note that the simulator does not reproduce the motion distortion of the LiDAR data. The flight course is shown in Fig. 11. A computer which has i7-6700 CPU and GTX1080 GPU with 16 GB memory was used for the estimation. The DNN inference computation takes around 0.005 s with the computer, while 'DGSphere' takes around 0.4 s every step.

Experimental results

The estimated attitudes in 'Neighborhood' are plotted in Fig. 12. Table 6 shows the MAE of the estimated attitude. The MAE of 'Gyro+DNN (ours)' is smaller than ones of the other methods. 'Gyro' had large accumulative error. That is natural because noise was added and the method does not have any other observation. 'Gyro+Acc' did not have accumulative error. However it had error constantly, since the acceleration values of the sensor contained own acceleration of the robot and noise. On the other hand, the proposed method can observe the gravity vector which does not contain them. 'Gyro+NDT' accumulated error slower than 'Gyro' did by using the LiDAR, but it could not remove the accumulation. 'Gyro+DGSphere' and 'Gyro+DNN' corrected the accumulative error by observing the estimated gravity. Comparing their MAE, the deep learning surpasses the rule-based method. Moreover, the DNN outputs the estimation much faster than 'DGSphere' which processes 3D point clouds.

Validation of real-time estimation in real world

To see the fine-tuned DNN can work in real world, two types of experiments with the real sensors (Fig. 3) were performed.

Indoor experiment with motion capture

The sensors were hand-carried in an indoor environment of 4.5 m $\times$ 6 m for about 23 min. Motion capture cameras (Vicon Vero v1.3X) were used for measuring the ground truth. Note that the DNN was not trained in this area.

Table 7 shows the MAE of the estimated attitude. The proposed method suppressed accumulation of error also in the real world. In the flat indoor environment, the MAE given by the proposed method is almost the same as that of 'Gyro+Acc'. The acceleration measured with the IMU is not integrated in the proposed EKF in this paper just for making the validation simple, but it actually can be integrated, and it would be a more stable estimation. For reference, the error given by 'Gyro+Acc+DNN' was $\phi_{\text{error}} = 2.503$ deg, $\theta_{\text{error}} = 1.637$ deg. 'DNN' without EKF showed worse performance compared with 'Gyro+DNN (ours)' although they performed similarly in the simulator. One of the reasons may be because the sensors moved more intensely in this real experiment. In order to see this, Fig. 13 plots the sensor attitude during the experiment. The DNN had larger error when the sensors moved rapidly. Motion distortion of the LiDAR data might affect the DNN in the real world, while the simulator does not reproduce the distortion. Another reason may be because 'DNN' does not interpolate the state between inferences unlike 'Gyro+DNN (ours)' .

Outdoor experiment

The motion capture cameras measure the attitude accurately, but the captured area is limited. To complement that, a long distance experiment was also performed. Detailed quantitative evaluation of the accuracy was done in the previous section, thus this section is just for seeing that the proposed method also be able to work outdoors.

The sensors were hand-carried for around 5 min in Area-II (Fig. 11(b)) where the DNN was not trained. Since the ground truth is not available while the sensors are being carried, the estimated attitude at the end

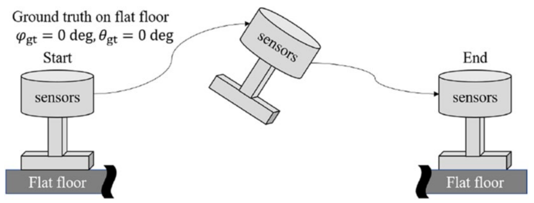

Ground truth on flat floor
$\varphi_{gt} = 0$ deg, $\theta_{gt} = 0$ deg
Start
sensors
sensors
End
sensors
Flat floor
Flat floor

Fig. 14 Dynamic experiment. The ground truth on the flat floor is assumed to be $\phi_{\text{gt}} = 0$ deg, $\theta_{\text{gt}} = 0$ deg

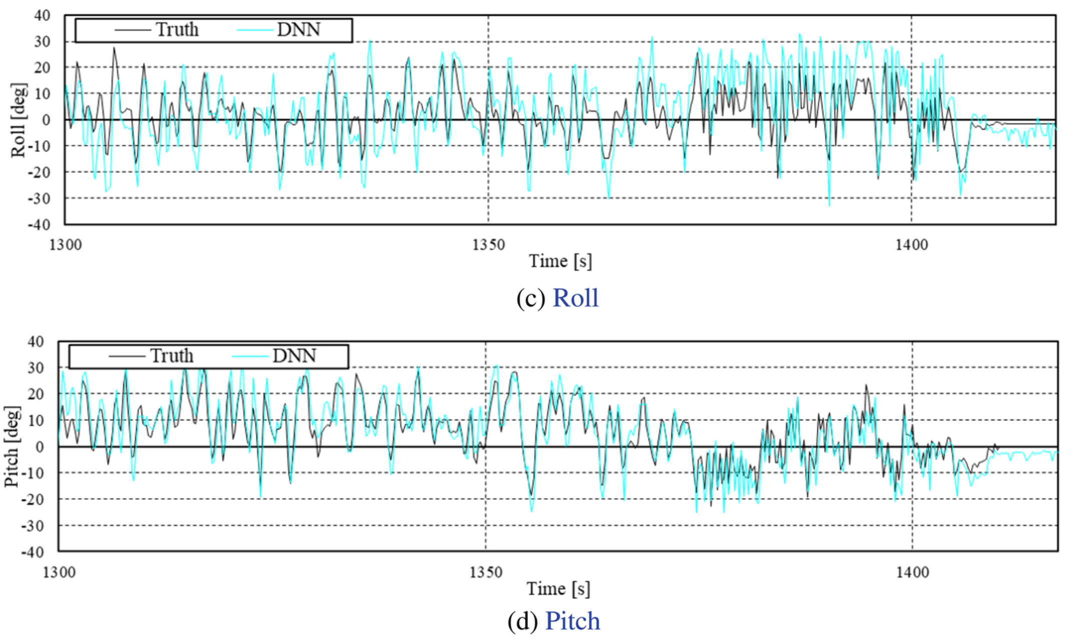


Fig. 13 Real-time plotting in indoor experiment. The DNN had larger error when the sensors moved intensely

Table 8 Error of estimated attitude at last pose in outdoor experiment

	Roll [deg]	Pitch [deg]
Gyro	+5.268	−5.047
Gyro+Acc	−0.269	+0.303
Gyro+DNN (ours)	−1.347	−0.654

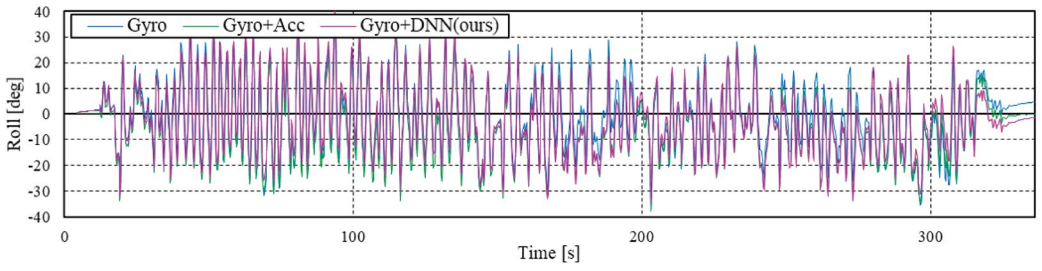
of carrying was evaluated to see error accumulation. The sensors were placed on a flat floor at the start and end of the experiment as Fig. 14, and the ground truth was assumed as $\phi_{gt} = 0$ deg, $\theta_{gt} = 0$ deg. This evaluation method is based on the related study [16].

Table 8 shows the error of the estimation at the last pose. The proposed method suppressed accumulation of error during the driving outdoor. 'Gyro+Acc' had very small error at the final pose because the sensor was still. The intermediate estimation results are also shown in Fig. 15 as reference.

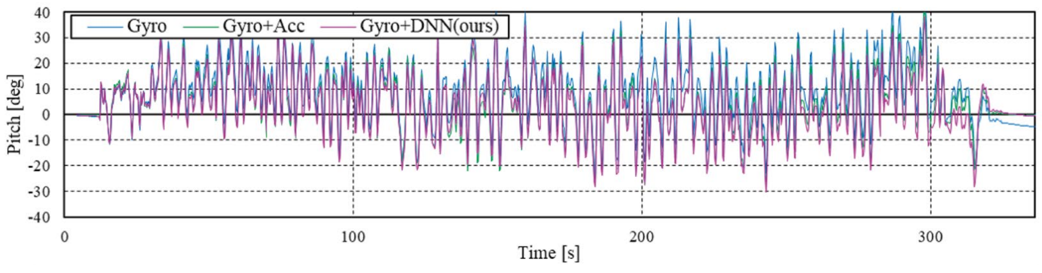
Conclusions and future work

The proposed method integrates a gyroscope and the DNN for estimating self-attitude in real-time. The proposed network estimates the gravity direction from LiDAR data. It was trained with synthetic data, and was fine-tuned with real data. Pre-training with the large synthetic data and augmenting the data help making the learning efficient. The static experiment showed the DNN can infer the gravity direction from only single shot LiDAR data. It showed good results regardless of day or night. For dynamic estimation, angular rates from a gyroscope and the DNN's outputs are integrated in the EKF. The dynamic experiments showed the proposed method can be used for real-time estimation.

However, it should be noted that the proposed DNN did not perform well in the situations which are not contained in the training datasets, especially without buildings. A way to collect more variety of data or judging the difficulty of inferences is necessary in our future work. Besides, the proposed method does not cope with the distortion of the LiDAR data in this paper. It worked well in the experiments, but the distortion may affect the inference when the LiDAR moves much faster. Testing the effects and coping with it are our future work. As our other future work, adopting the invariant extended Kalman filter (IEKF) [22] as the estimator instead of the EKF should be considered. Combining the camera DNN and the LiDAR DNN, or using other sensors for estimating the attitude is another future work.



(a) Roll



(b) Pitch

Fig. 15 Real-time plotting in outdoor experiment. Note that the ground truth while the sensors were being carried is not available in the outdoor environments

Acknowledgements
The authors are thankful for the generous support from the New Energy and Industrial Technology Development Organization (NEDO) for this study. This study was conducted under 'Autonomous Robot Research Cluster' at Meiji University.

Authors' contributions
RO proposed the method described in this paper, implemented all the programing, conducted some of the experiments, and drafted the manuscript. NS conducted one of the experiments. YK provided the inspiration for this study, provided advice, and checked and corrected the manuscript. All authors read and approved the final manuscript.

Funding
This study was supported by the New Energy and Industrial Technology Development Organization (NEDO).

Availability of data and materials
Source code and dataset: https://github.com/ozakiryota/depth_image_to_gravity. The code is implemented using Python, C++, PyTorch API and ROS API. Video of experiments: https://photos.app.goo.gl/m1F2v9taKw9sEjRz6.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 9 January 2021 Accepted: 5 November 2021
Published online: 20 December 2021

References
1. Vaganay J, Aldon MJ, Fournier A (1993) Mobile robot attitude estimation by fusion of inertial data. In: Proceedings of 1993 IEEE International Conference on Robotics and Automation (ICRA), pp. 277–282
2. Thrun S, Burgard W, Fox D (2005) Probabilistic robotics. The MIT Press, Cambridge, pp 309–336
3. Rusinkiewicz S, Levoy M (2001) Efficient variants of the icp algorithm. In: Proceedings of Third International Conference on 3-D Digital Imaging and Modeling, pp. 145–152
4. Biber P, er WS (2003) The normal distributions transform: a new approach to laser scan matching. In: Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
5. Engel J, Stueckler J, Cremers D (2014) Lsd-slam: large-scale direct monocular slam. In: Proceedings of European Conference on Computer Vision (ECCV), pp. 834–849
6. Mur-Artal R, Montiel JMM, Tardós JD (2015) Orb-slam: a versatile and accurate monocular slam system. IEEE Trans Robotics 31(5):1147–1163
7. Quddus MA, Ochieng WY, Noland RB (2007) Current map-matching algorithms for transport applications: state-of-the art and future research directions. Transportation Res Part C Emerg Tech 15(5):312–328
8. Kim P, Coltin B, Kim HJ (2018) Linear rgb-d slam for planar environments. In: Proceedings of European Conference on Computer Vision (ECCV), pp. 333–348
9. Hwangbo M, Kanade T (2011) Visual-inertial uav attitude estimation using urban scene regularities. In: Proceedings of 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 2451–2458
10. Goto T, Pathak S, Ji Y, Fujii H, Yamashita A, Asama, H (2018) Line-based global localization of a spherical camera in manhattan worlds. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 2296–2303
11. Ozaki R, Kuroda Y (2019) Real-time 6dof localization with relative poses to walls of buildings. Trans JSME 85(875):19–00065 (**in Japanese**)
12. do Lima JPSM, Uchiyama H, Taniguchi RI (2019) End-to-end learning framework for imu-based 6-dof odometry. Sensors 19(17):3777
13. Al-Sharman MK, Zweiri Y, Jaradat MAK, Al-Husari R, Gan D, Seneviratne LD (2020) Deep-learning-based neural network training for state estimation enhancement: application to attitude estimation. IEEE Trans Instrum Meas 69(1):24–34
14. Mérida-Floriano M, Caballero F, Acedo D, García-Morales D, Casares F, Merino L (2019) Bioinspired direct visual estimation of attitude rates with very low resolution images using deep networks. In: Proceedings of 2019 IEEE International Conference on Robotics and Automation (ICRA), pp. 5672–5678
15. Shah S, DeyChris D, Kapoor L (2017) Airsim: high-fidelity visual and physical simulation for autonomous vehicles. Field Serv Robotics 5:621–635
16. Ellingson G, Wingate D, McLain T (2017) Deep visual gravity vector detection for unmanned aircraft attitude estimation. In: Proceedings of 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
17. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of ICML 2010, pp. 807–814
18. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
19. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference for Learning Representations (ICLR)
20. Beard RW, McLain TW (2012) Small unmanned aircraft: theory and practice. Princeton University Press, Princeton, NJ
21. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. In: arXiv Preprint, pp. 1409–1556
22. Bonnable S, Martin P, Salaün E (2009) Invariant extended kalman filter: theory and application to a velocity-aided attitude estimation problem. In: Proceedings of the 48h IEEE Conference on Decision and Control (CDC), pp. 1297–1304