Ozaki and Kuroda *Robomech J* (2021) 8:9
https://doi.org/10.1186/s40648-021-00196-3

ROBOMECH Journal

RESEARCH ARTICLE Open Access

EKF-based self-attitude estimation with DNN learning landscape information

Ryota Ozaki* and Yoji Kuroda

Abstract
This paper presents an EKF-based self-attitude estimation with a DNN (deep neural network) learning landscape information. The method integrates gyroscopic angular velocity and DNN inference in the EKF. The DNN predicts a gravity vector in a camera frame. The input of the network is a camera image, the outputs are a mean vector and a covariance matrix of the gravity. It is trained and validated with a dataset of images and corresponded gravity vectors. The dataset is collected in a flight simulator because we can easily obtain various gravity vectors, although the method is not only for UAVs. Using a simulator breaks the limitation of amount of collecting data with ground truth. The validation shows the network can predict the gravity vector from only a single shot image. It also shows that the covariance matrix expresses the uncertainty of the inference. The covariance matrix is used for integrating the inference in the EKF. Flight data of a drone is also recorded in the simulator, and the EKF-based method is tested with it. It shows the method suppresses accumulative error by integrating the network outputs.

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

Introduction
Estimating attitude of a robot or a UAV is one of the classic problems of mobile robotics. Especially, 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 rotors. 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 the disturbances with only inertial sensors is quite difficult.

To reduce the influence of these disturbances, many kinds of SLAM (Simultaneous Localization And Mapping) [2] have been proposed. SLAM with LiDAR registers point clouds by ICP [3], NDT [4], and so on. Many visual SLAM with cameras have also been proposed [5, 6]. SLAM often contains accumulative error since relative pose changes with error are summed up. Some methods use prior information such as 3D maps for estimating self-pose [7]. These methods correct the error by matching the prior information against data from the sensor. However, they work only in environments where maps are available. Moreover, creating a map is time-consuming and also requires update. Some methods [8, 9] estimate attitude under Manhattan world assumption. They assume that planes and edges in the environment are orthogonal to each other. It helps achieving drift-free estimation. However, it is difficult for this kind of methods to avoid being affected by objects which do not satisfy the assumption.

Deep learning has been used for attitude estimation in recent years. In [10], IMU-based odometry by end-to-end learning has been proposed. In [11], a deep neural network identifies the measurement noise characteristics of IMU. In [12], 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 [13] which offers visually realistic

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

graphics. In [14], a gravity vector is directly estimated from a single shot image. This is based on expectation that the network can learn edge, context, and landscape information; for example, most of all artificial buildings should be built vertically, the sky should be seen when the camera orients upper, and so on. The method does not depend on time sequence since only a single shot image is used to estimate the attitude. It helps suppressing drift, noise, and accumulative error. This method is the most similar to our proposed method. However, this method contains some problems. It cannot express uncertainty of the prediction; for instance, the network outputs estimation even when the camera is all covered by obstacles, when less features are captured, and so on. These outputs with large error worsen estimation when it is used in filter functions such as Kalman filter [15]. Therefore, they should be detected and be rejected before the integration.

To address these issues above, this paper presents self-attitude estimation with DNN which predicts a gravity vector from a single shot image, where the outputs are mean and variance. Only outputs with small variance are absorbed in the EKF (extended Kalman filter) to suppress accumulative error. Moreover, the output covariance matrices are used to adjust the process uncertainty in the EKF. The differences from the related work [14] are noted below:

- A larger dataset is collected in our work by using a simulator.
- Our network is trained with reliable ground truth while the related work collects a dataset with a hand carried phone.
- L2 normalization is applied to the output gravity vector while ReLU is applied in the related work.
- Our network outputs mean and a covariance matrix while the related work directly outputs the gravity vector.
- Our method filters out DNN inferences with large variance before processing the integration in the EKF.
- Our method uses the covariance matrix output from the DNN as update process noise in the EKF.

The dataset and the source code used in this paper for deep learning, and for the EKF have been released in open repositories.

DNN estimating gravity vector

The proposed method makes the DNN learn landscape information for estimating a gravity vector in a camera frame. The gravity is expressed as mean and covariance to consider uncertainty of the prediction.

Coordinate definition

A camera frame is defined as a right-handed coordinate system which is fixed on the camera pose. It is shown in Fig. 1.

Dataset collection

A dataset is collected in AirSim [13]. AirSim is a simulator for drones, cars and more, built on Unreal Engine, which provides visually realistic graphics. The dataset consists of images and corresponded gravity vectors $\boldsymbol{g}$ in the camera frame. The camera pose and weather parameters are randomized, and a image and a gravity vector are recorded at each pose. The range of random Z is limited as [2 m, 3 m] in this work. The ranges of random roll ϕ and pitch θ are limited as [-30 deg, 30 deg], respectively. Fig. 2 shows examples of the dataset.

Real data is also collected with the sensors in Fig. 3 for test. The stick is hand-carried, and images and linear acceleration vectors measured with the IMU (Xsens MTi-30) are recorded. They are saved only when the stick is shaking less than 0.001 m, 0.1 deg, 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.

Data transformation and augmentation

Input data and label data are transformed, and are augmented in each epoch of training.

Image (input)

The image is randomly rotated for augmenting the roll data. The rotation angle $\Delta\phi$ is limited as [− 10 deg, 10 deg]. The image is resized to 224 × 224. RGB values are normalized. In this work, this normalization is done following $\boldsymbol{mean} = (0.5, 0.5, 0.5)$ and $\boldsymbol{std} = (0.5, 0.5, 0.5)$. Fig. 4 shows an example of the data augmentation.

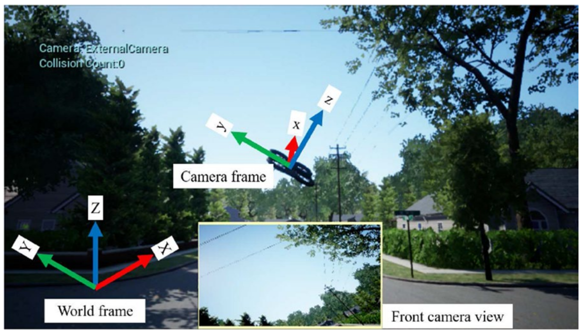


Fig. 1 Screenshot of AirSim with coordinate description. An IMU and a camera are equipped to the drone in the simulator. The purpose of the proposed neural network is estimating a gravity vector in the camera frame from a front camera image

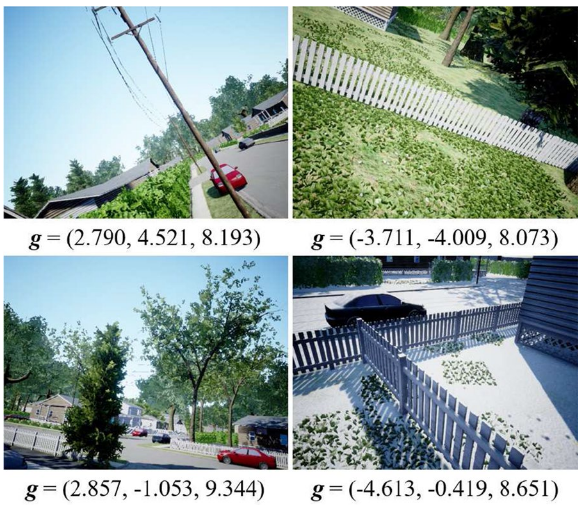

g = (2.790, 4.521, 8.193)
g = (-3.711, -4.009, 8.073)
g = (2.857, -1.053, 9.344)
g = (-4.613, -0.419, 8.651)

Fig. 2 Examples of datasets. The dataset consists of images and correspond gravity vectors $\boldsymbol{g}$ [m/s^2] in the camera frame. These examples were collected in "Neighborhood" of AirSim. The camera pose and weather parameters are randomized for creating a dataset. An example of the weather parameter is that the road in the lower right image is covered in snow

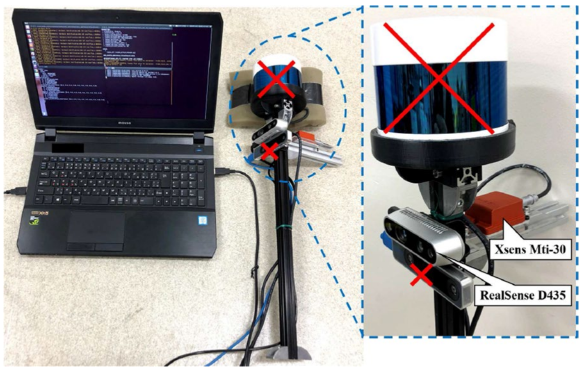


Fig. 3 Sensor stick. Images and acceleration are recorded with this stick when it is still. The judge whether it is still is processed by programing. Note that depth information is not used in this study although RealSense D435 is a RGB-D camera

Gravity vector (label)

A gravity vector in the camera frame is rotated according to $\Delta\phi$. Since the network does not need to learn norm of the gravity, L2 normalization is also applied to the vector in order to make training efficient.

$$\boldsymbol{g}_{\mathbf{trans}} = \boldsymbol{Rot}^{\mathrm{xyz}}_{(\Delta\phi)} \frac{\boldsymbol{g}}{|\boldsymbol{g}|}$$
$$\boldsymbol{Rot}^{\mathrm{xyz}}_{(\Delta\phi)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(-\Delta\phi) & -\sin(-\Delta\phi) \\ 0 & \sin(-\Delta\phi) & \cos(-\Delta\phi) \end{pmatrix} \quad (1)$$

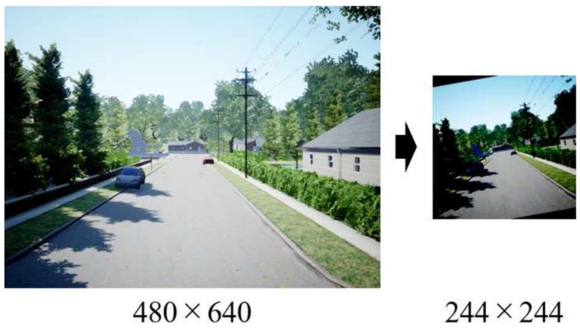


Fig. 4 Example of a transformed image. An image is randomly rotated according to $\Delta\phi$. This example shows an image when $\Delta\phi = 10$ deg. It is also resized, and is normalized

Network

The proposed DNN is shown in Fig. 5. It consists of CNN layers and FC layers. Its input is a resized image, and its outputs are a mean vector of a gravity vector and a covariance matrix. Technically, the output of the FC layers is $(\mu_{g_x}, \mu_{g_y}, \mu_{g_z}, L_0, \cdots, L_5)$, and the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ are computed as Eqs. 2, 3, respectively. Since the lower-triangular matrix $\boldsymbol{L}$ is required to have positive-valued diagonal entries, an exponential function is applied to the diagonal elements.

$$\boldsymbol{\mu} = \frac{(\mu_{g_x}, \mu_{g_y}, \mu_{g_z})^{\mathrm{T}}}{|(\mu_{g_x}, \mu_{g_y}, \mu_{g_z})^{\mathrm{T}}|} \quad (2)$$

$$\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}, \quad \boldsymbol{L} = \begin{pmatrix} \exp(L_0) & 0 & 0 \\ L_1 & \exp(L_2) & 0 \\ L_3 & L_4 & \exp(L_5) \end{pmatrix} \quad (3)$$

It is expected that the CNN layers lean extracting features such as edges, and the FC layers learn landscape information. Feature module of VGG16 [16] pre-trained on ImageNet [17] is adopted as the CNN layers of the proposed method. All layers, except the final output layer, use the ReLU function [18] as activation function. All FC layers, except the final output layer, use the 10% Dropout [19].

Loss function

By learning the distribution of the dataset and updating the weights to maximize the probability density for the outputs, the DNN can output mean and variance. Assuming that the estimation follows a multivariate normal distribution, the probability density is computed as Eq. 4.

$$P(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{\exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}))}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}, \quad k = \mathrm{rank}(\boldsymbol{\Sigma}) \quad (4)$$

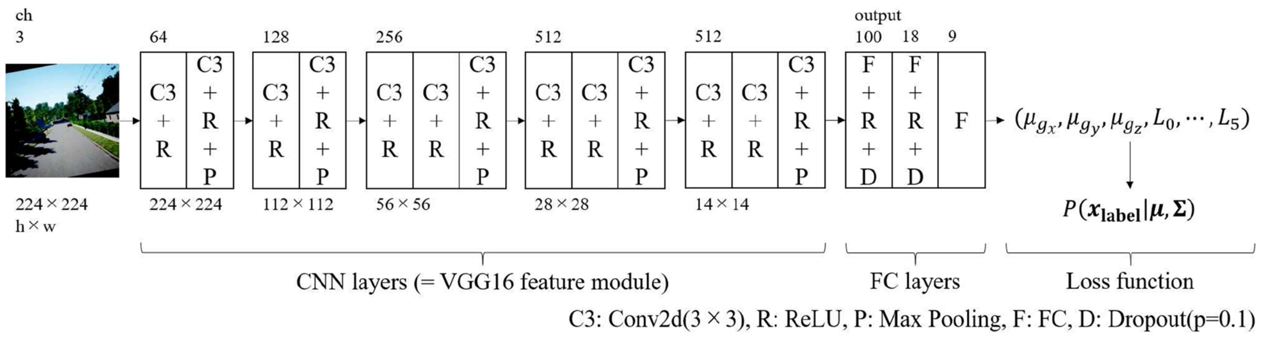

ch
3
224 × 224
h × w
64
128
256
512
512
output
100 18 9
C3 + R
C3 + R + P
F + R + D
F
224 × 224
112 × 112
56 × 56
28 × 28
14 × 14
$(\mu_{g_x}, \mu_{g_y}, \mu_{g_z}, L_0, \cdots, L_5)$
$P(\boldsymbol{x}_{\text{label}}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$
CNN layers (= VGG16 feature module)
FC layers
Loss function
C3: Conv2d(3 × 3), R: ReLU, P: Max Pooling, F: FC, D: Dropout(p=0.1)

Fig. 5 Proposed network architecture. It consists of CNN layers and FC layers. The input data is a resized image, and the output data are a mean vector and a covariance matrix. They are computed with an output from the final layer as Eqs. 2, 3, respectively. Log-probability of multivariate normal distribution is used as a loss function of this model

where k denotes the dimensions of the variables, i.e. $k = 3$ in the proposed method. To make the network learn the probabilistic model, it is trained maximizing $P(\boldsymbol{x}_{\mathbf{label}}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{x}_{\mathbf{label}}$ $(= \boldsymbol{g}_{\mathbf{trans}})$ is a label of the dataset. The total probability density P_{total} of a dataset D_{label} is computed by multiplying as Eq. 5.

$$\begin{aligned} P_{\text{total}} &= \prod_{i=0}^{n} P(\boldsymbol{x}_{\mathbf{label}_i}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \\ D_{\text{label}} &= [\boldsymbol{x}_{\mathbf{label}_0}, \cdots, \boldsymbol{x}_{\mathbf{label}_i}, \cdots, \boldsymbol{x}_{\mathbf{label}_n}] \end{aligned} \tag{5}$$

Since the natural logarithm is a monotonically increasing function, maximizing P_{total} can be simplified by taking the natural logarithm. This avoids the value becoming too small by multiplying, and decreases computational cost. Here, $P_{\log_{\text{total}}}$ denotes a total value of the log-probability.

$$P_{\log_{\text{total}}} = \sum_{i=0}^{n} \ln P(\boldsymbol{x}_{\mathbf{label}_i}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{6}$$

Moreover, maximizing $P_{\log_{\text{total}}}$ is equivalent to minimizing $P_{\log_{\text{total}}}$. Finally, the loss function of the proposed method is Eq. 7.

$$f(\boldsymbol{w}) = -P_{\log_{\text{total}}} \tag{7}$$

where $\boldsymbol{w}$ denotes parameters of the network. The network minimizes the loss by updating $\boldsymbol{w}$.

Optimization

Adaptative Moment Estimation (Adam) [20] is used to optimize the parameters. 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.

Validation of DNN

The proposed DNN was validated by comparing to other methods. The datasets used in this validation are listed in Table 1.

Compared methods

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

MLE (ours, all)

"MLE (ours, all)" denotes the proposed method described in "DNN estimating gravity vector" section . MLE is short for Maximum Likelihood Estimation.

MLE (ours, selected)

"MLE (ours, selected)" denotes the method uses the exactly same network and the same parameters as "MLE (ours, all)" does, but only samples which output small variance are used for validation of attitude estimation. It means samples with large variance are filtered out as outliers. Assuming β in Eq. 8 expresses uncertainty of the prediction, samples with small variance are selected with a threshold TH_β. In this validation, the threshold is set as $\text{TH}_\beta = \frac{1}{n}\sum_{i=0}^{n} \beta_i$, where n is the number of samples in the testing dataset.

Table 1 Dataset list

id#	Environment	# samples	Usage
1	AirSim's "Neighborhood"	10000	Training
2	---"---	1000	Test
3	AirSim's "SoccerField"	1000	Test
4	Real world	1108	Test

$$\beta = \sqrt{\Sigma_{0,0}} \times \sqrt{\Sigma_{1,1}} \times \sqrt{\Sigma_{2,2}}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \Sigma_{0,0} & \Sigma_{0,1} & \Sigma_{0,2} \\ \Sigma_{1,0} & \Sigma_{1,1} & \Sigma_{1,2} \\ \Sigma_{2,0} & \Sigma_{2,1} & \Sigma_{2,2} \end{pmatrix} \tag{8}$$

Regression w/ L2 normalization

"Regression w/ L2 normalization" denotes the network which the final FC layer is difference from "MLE (ours)". It outputs a 3d vector without covariance. It is a similar architecture as [14]. L2 normalization is applied to the final layer while ReLU is applied in [14]. Mean square error (MSE) between labels and outputs is used as a loss function. Fig. 6 shows the network architecture.

Regression w/o L2 normalization

"Regression w/o L2 normalization" denotes the regression network without L2 normalization. It is require to learn not only the direction, but also norm of the gravity vector, i.e. approximately 9.8 m/s^2, in order to minimize the loss. This information is not required to estimate attitude ϕ, θ.

Training and validation

The network was trained with 10000 samples (#1) with a batch size of 200 samples for 200 epochs. Another 1000 samples (#2) were used for test. They were collected in "Neighborhood" of AirSim. The training dataset and the test dataset were not mixed.

Loss values during training are plotted in Fig. 7. The regression models converged much faster than the MLE model did. The regression model with L2 normalization converged a bit faster than the regression model without L2 normalization did. Tables 2, 3 and 4 respectively show the loss values after 200 epoch training. It is noted that gradient was not computed with the test datasets. It is also noted a loss function of the MLE model and one of the regression models are difference.

Another 1000 samples (#3) were also collected in "SoccerField" of AirSim as data in an unknown environment. 1108 real data samples (#4) were also collected with the sensors in Fig. 3. Note that these samples are just for test, thus the DNN was not trained with them. The loss values of these samples are larger than ones of the known environment in which the network was trained.

Attitude estimation

Roll ϕ and pitch θ of the camera pose in the gravitational coordinate are estimated by using $\boldsymbol{\mu}$.

$$\phi = \tan^{-1}\frac{\mu_{g_y}}{\mu_{g_z}}, \quad \theta = \tan^{-1}\frac{-\mu_{g_x}}{\sqrt{\mu_{g_y}^2 + \mu_{g_z}^2}} \tag{9}$$

Mean absolute error (MAE) of the estimation of the test dataset is shown in Table 5. Variance of the estimated attitude error is shown in Table 6. Both of the error and the variance of "MLE (ours, selected)" are smaller than the others. 715 samples which has $\beta < \mathrm{TH}_\beta = 0.00008814$ m^3/s^6 were selected from 1000 test samples (#2) with "MLE (ours, selected)".

Comparing "MLE (ours, all)" and "MLE (ours, selected)", filtering by TH_β is found valid, which means the network expresses the uncertainty by outputting the covariance matrix. In order to see this, the samples are sorted in Fig. 8. In Fig. 8a, top 50 samples with largest estimation error with "MLE (ours)" are shown in order. In Fig. 8b, top 50 samples with the largest β with "MLE (ours)" are shown in order. 21 samples of the top 50 samples are mutual of both groups. In Fig. 8a, most of the sample images with large error are covered by obstacles, and they are dark with much less landscape information. There is no way to detect them by the regression model. Correlation between error and β are not complete, but many samples with large error were detected by sorting samples with β. A good example with large β and one

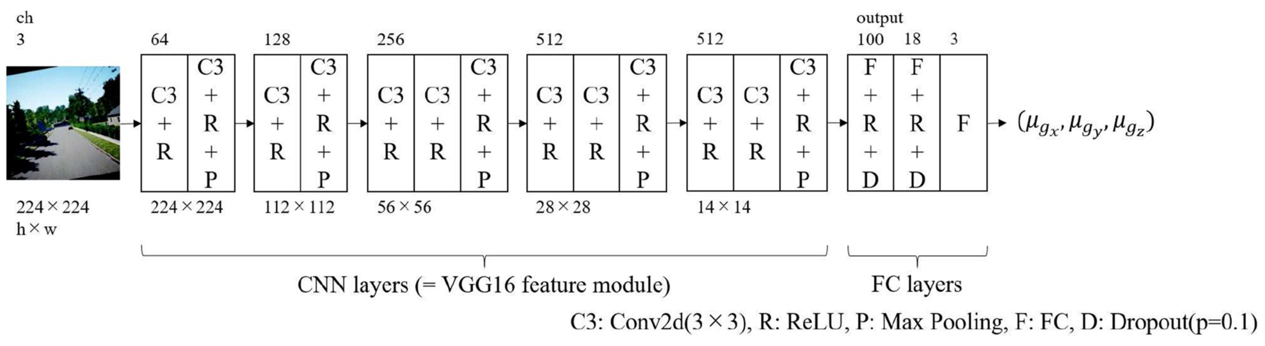

ch
3
64
128
256
512
512
output
100
18
3
C3 + R
C3 + R + P
F + R + D
F
$(\mu_{g_x}, \mu_{g_y}, \mu_{g_z})$
224×224
h×w
224×224
112×112
56×56
28×28
14×14
CNN layers (= VGG16 feature module)
FC layers
C3: Conv2d(3×3), R: ReLU, P: Max Pooling, F: FC, D: Dropout(p=0.1)

Fig. 6 Comparative network architecture. This network is almost same as [14]. Whereas the proposed method outputs the mean and variance, this one outputs only the mean

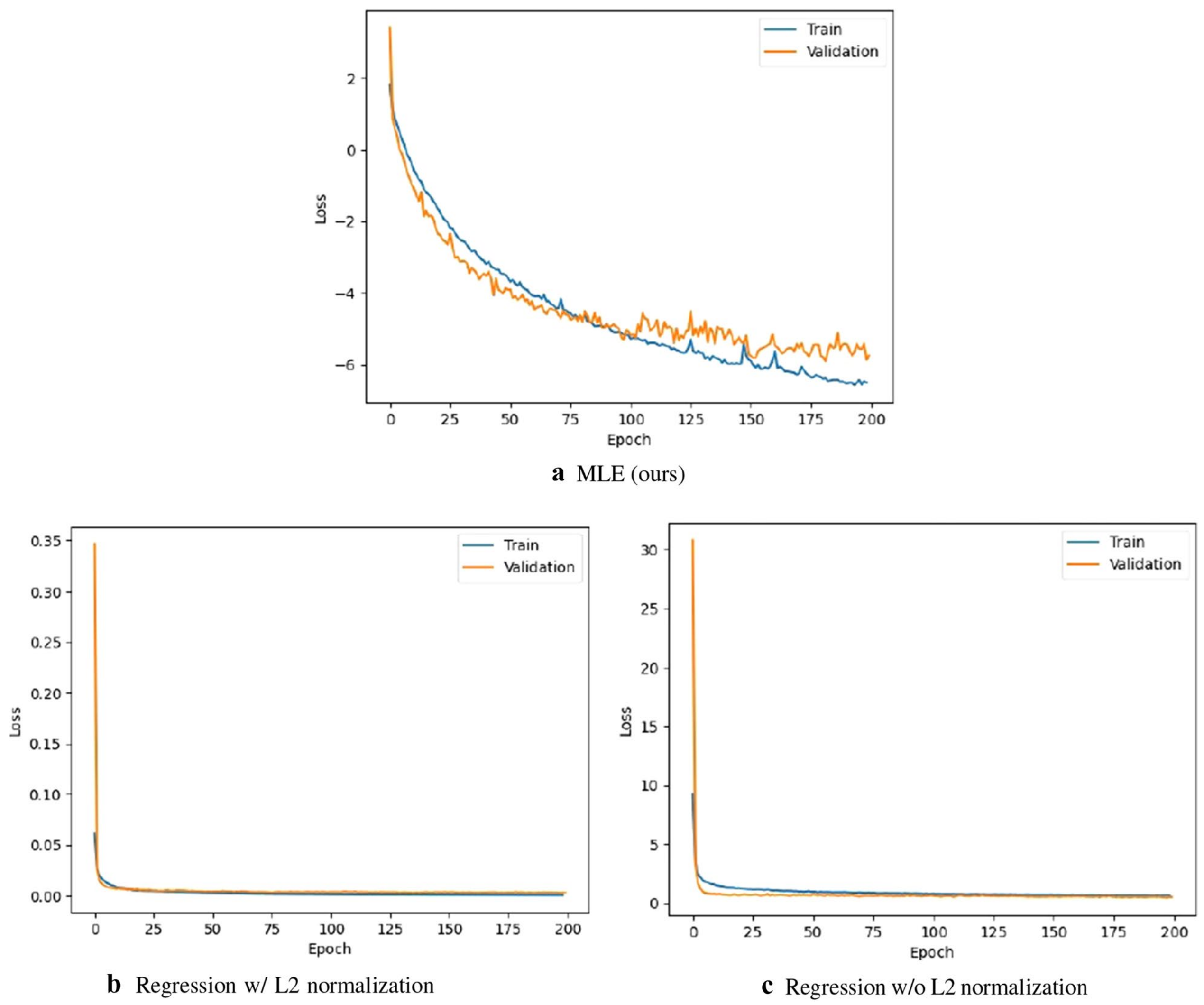


Fig. 7 Loss plotting. It is noted a loss functions of the MLE model and one of the regression models are difference. Therefore, their values can not be simply compared

Table 2 Loss of MLE (ours) after 200 epochs

	Dataset#			
	#1	#2	#3	#4
Loss [m/s^2]	– 6.4991	– 5.7436	– 2.9386	– 2.7129

Table 3 Loss of Regression w/ L2 normalization after 200 epochs

	Dataset#			
	#1	#2	#3	#4
Loss [m^2/s^4]	0.0011	0.0031	0.0084	0.0055

Table 4 Loss of Regression w/o L2 normalization after 200 epochs

	Dataset#			
	#1	#2	#3	#4
Loss [m^2/s^4]	0.6588	0.4916	1.0951	0.8284

with small β are shown in Fig. 9, respectively. Obviously, the sample in Fig. 9a does not have enough landscape information to estimate the gravity direction, and the proposed network expresses the uncertainty with large β.

Table 5 MAE of estimated attitude in known environment (#2)

	Roll [deg]	Pitch [deg]
MLE (ours, all)	2.620	2.277
MLE (ours, selected)	*1.836*	*1.467*
Regression w/ L2 normalization	2.727	2.525
Regression w/o L2 normalization	2.766	2.366

Table 6 Variance of estimated attitude error in known environment (#2)

	Roll [deg^2]	Pitch [deg^2]
MLE (ours, all)	23.139	18.348
MLE (ours, selected)	*9.657*	*4.553*
Regression w/ L2 normalization	25.161	21.996
Regression w/o L2 normalization	21.668	18.213

Comparing "Regression w/ L2 normalization" and "Regression w/o L2 normalization", L2 normalization does not contribute to the accuracy, although the one with L2 normalization converged a bit faster than the one without L2 normalization did.

A dataset of the unknown environment "SoccerField" (#3) and one of real world (#4) were also used for validation. Note that the neural networks were not trained in these environments. MAE and variance of the error are shown in Tables 7, 8, 9 and 10, respectively. Filtering by the threshold TH_β made error smaller even in the unknown environments. However, the errors and the variances of errors are larger than the known environment (Tables 5, 6). To reduce difference of results between in known environments and in unknown ones, wider variety of datasets are needed for training.

EKF-based self-attitude estimstion with DNN

The proposed method integrates angular velocities from a gyroscope and the DNN outputs in the EKF. The proposed EKF architecture is shown in Fig. 10. The state vector $\boldsymbol{x}$ of the proposed Kalman filter consists of roll and pitch of the robot. Both of the vector $\boldsymbol{x}$ and the covariance matrix $\boldsymbol{P}$ are computed in a prediction process and in an update process. The prediction process is computed by integrating angular velocity from a gyroscope. The update process is computed by observing the output of the DNN.

$$\boldsymbol{x} = \begin{pmatrix} \phi & \theta \end{pmatrix}^{\mathrm{T}} \tag{10}$$

Here, k denotes time step, S_ϕ, C_ϕ, T_ϕ are short for $\sin\phi$, $\cos\phi$, $\tan\phi$, respectively in the following sections.

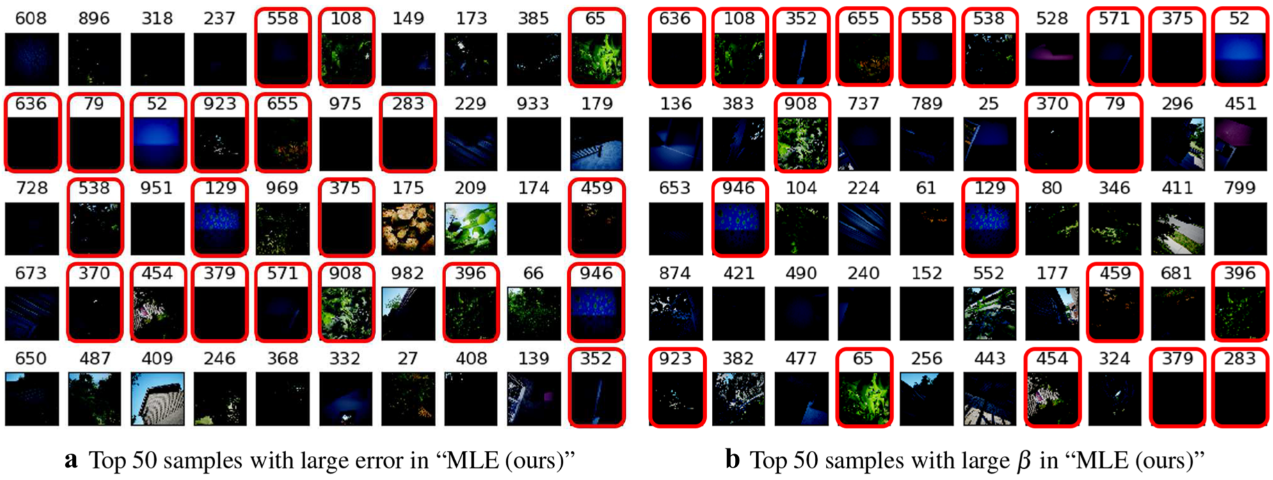

a Top 50 samples with large error in "MLE (ours)"
b Top 50 samples with large β in "MLE (ours)"

Fig. 8 Sorted samples. A number above each image is a index of a sample. In (**a**), top 50 samples are sorted in descending order of the error in "MLE (ours)". Error of sample#608 in the regression model is $\phi_{\text{error}} = -35.34$ deg, $\theta_{\text{error}} = -25.74$ deg. Error of sample#352 is $\phi_{\text{error}} = 2.80$ deg, $\theta_{\text{error}} = -12.93$ deg. In (**b**), top 50 samples are sorted in descending order of β in "MLE (ours)". Mutual samples of the both groups are marked with red rectangles

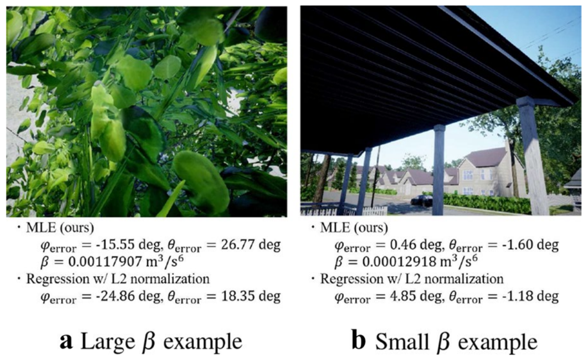

· MLE (ours)
φ_{error} = -15.55 deg, θ_{error} = 26.77 deg
β = 0.00117907 m^3/s^6
· Regression w/ L2 normalization
φ_{error} = -24.86 deg, θ_{error} = 18.35 deg
a Large β example
· MLE (ours)
φ_{error} = 0.46 deg, θ_{error} = -1.60 deg
β = 0.00012918 m^3/s^6
· Regression w/ L2 normalization
φ_{error} = 4.85 deg, θ_{error} = -1.18 deg
b Small β example
Fig. 9 Examples with large β and small β. Obviously, it is hard to estimate the gravity direction from the sample (**a**). The proposed network expresses uncertainty of the prediction of the sample (**a**) by outputting large covariance

Table 7 MAE of estimated attitude in unknown environment (#3)

	Roll [deg]	Pitch [deg]
MLE (ours, all)	3.120	4.062
MLE (ours, selected)	*2.069*	*2.549*
Regression w/ L2 normalization	3.796	4.386
Regression w/o L2 normalization	3.744	4.695

Table 8 Variance of estimated attitude error in unknown environment (#3)

	Roll [deg^2]	Pitch [deg^2]
MLE (ours, all)	31.170	36.967
MLE (ours, selected)	*9.134*	*12.932*
Regression w/ L2 normalization	35.358	45.428
Regression w/o L2 normalization	36.460	55.347

Table 9 MAE of estimated attitude in real world (#4)

	Roll [deg]	Pitch [deg]
MLE (ours, all)	2.648	4.361
MLE (ours, selected)	*2.199*	*3.755*
Regression w/ L2 normalization	2.812	4.665
Regression w/o L2 normalization	3.092	5.371

Table 10 Variance of estimated attitude error in real world (#4)

	Roll [deg^2]	Pitch [deg^2]
MLE (ours, all)	20.548	20.585
MLE (ours, selected)	*9.356*	*13.971*
Regression w/ L2 normalization	20.976	20.982
Regression w/o L2 normalization	25.996	24.160

Prediction process

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

$$\begin{aligned}
\overline{\boldsymbol{x}_k} &= f_{(\boldsymbol{x}_{k-1},\boldsymbol{u}_{k-1})} = \boldsymbol{x}_{k-1} + \boldsymbol{Rot}^{\text{rpy}}_{(\boldsymbol{x}_{k-1})}\boldsymbol{u}_{k-1} \\
\boldsymbol{u}_{k-1} &= \boldsymbol{\omega}_{k-1}\Delta t = \begin{pmatrix} \omega_{\text{x}_{k-1}}\Delta t \\ \omega_{\text{y}_{k-1}}\Delta t \\ \omega_{\text{z}_{k-1}}\Delta t \end{pmatrix} \\
\boldsymbol{Rot}^{\text{rpy}}_{(\boldsymbol{x}_{k-1})} &= \begin{pmatrix} 1 & S_{\phi_{k-1}}T_{\theta_{k-1}} & C_{\phi_{k-1}}T_{\theta_{k-1}} \\ 0 & C_{\phi_{k-1}} & -S_{\phi_{k-1}} \end{pmatrix}
\end{aligned} \tag{11}$$

where f is a state transition model, $\boldsymbol{u}$ denotes a control vector, and $\boldsymbol{Rot}^{\text{rpy}}$ denotes a rotation matrix for angular velocities.

$$\overline{\boldsymbol{P}_k} = \boldsymbol{J_f}_{k-1}\boldsymbol{P}_{k-1}\boldsymbol{J_f}^T_{k-1} + \boldsymbol{Q}_{k-1}, \boldsymbol{J_f}_{k-1} \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{k-1},\boldsymbol{u}_{k-1}} \tag{12}$$

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

Update process

Outputs of the DNN with a large variance value β are rejected. A threshold TH_β is set for judging β, and only outputs with $\beta < \text{TH}_\beta$ are observed in the EKF. The observation vector is $\boldsymbol{z}$.

$$\boldsymbol{z} = \boldsymbol{\mu} \tag{13}$$

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

$$h_{(\boldsymbol{x}_k)} = \boldsymbol{Rot}^{\text{xyz}}_{(\boldsymbol{x}_k)}\boldsymbol{g}_{\textbf{world}}, \quad \boldsymbol{g}_{\textbf{world}} = \begin{pmatrix} 0 \\ 0 \\ g_{\text{world}} \end{pmatrix} \tag{14}$$

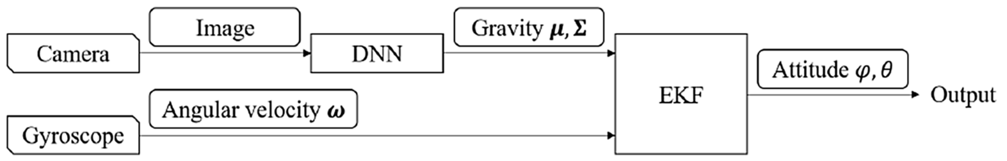

Camera
Image
DNN
Gravity $\boldsymbol{\mu}, \boldsymbol{\Sigma}$
EKF
Attitude φ, θ
Output
Gyroscope
Angular velocity $\boldsymbol{\omega}$

Fig. 10 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

$$\boldsymbol{Rot}^{\mathrm{xyz}}_{(\boldsymbol{x}_k)} = \begin{pmatrix} C_{\theta_k} C_{\psi_k} C_{\theta_k} S_{\psi_k} - S_{\theta_k} \\ S_{\phi_k} S_{\theta_k} C_{\psi_k} - C_{\phi_k} S_{\psi_k} S_{\phi_k} S_{\theta_k} S_{\psi_k} + C_{\phi_k} C_{\psi_k} S_{\phi_k} C_{\theta_k} \\ C_{\phi_k} S_{\theta_k} C_{\psi_k} + S_{\phi_k} S_{\psi_k} C_{\phi_k} S_{\theta_k} S_{\psi_k} - S_{\phi_k} C_{\psi_k} C_{\phi_k} C_{\theta_k} \end{pmatrix} \quad (15)$$

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 a vector. The covariance matrix of the process noise is $\boldsymbol{R}$.

$$\boldsymbol{R} = \begin{pmatrix} \gamma\,\Sigma_{0,0} & \Sigma_{0,1} & \Sigma_{0,2} \\ \Sigma_{1,0} & \gamma\,\Sigma_{1,1} & \Sigma_{1,2} \\ \Sigma_{2,0} & \Sigma_{2,1} & \gamma\,\Sigma_{2,2} \end{pmatrix} \quad (16)$$

where γ denotes a hyperparameter for adjusting variance. The state vector $\boldsymbol{x}$ and the covariance matrix $\boldsymbol{P}$ are respectively computed as following.

$$\begin{aligned} \hat{\boldsymbol{x}}_k &= \boldsymbol{x}_k + \boldsymbol{K}_k(\boldsymbol{z}_k - h_{(\boldsymbol{x}_k)}),\ \hat{\boldsymbol{P}}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{J}_{\boldsymbol{h}k})\boldsymbol{P}_k \\ \boldsymbol{J}_{\boldsymbol{h}k} &= \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{x_k},\ \boldsymbol{K}_k = \boldsymbol{P}_k \boldsymbol{J}_{\boldsymbol{h}k}^T (\boldsymbol{J}_{\boldsymbol{h}k} \boldsymbol{P}_k \boldsymbol{J}_{\boldsymbol{h}k}^T + \boldsymbol{R}_k)^{-1} \end{aligned} \quad (17)$$

where $\boldsymbol{J}_{\boldsymbol{h}}$ denotes h Jacobean, $\boldsymbol{K}$ denotes a gain matrix, and $\boldsymbol{I}$ denotes an identity matrix.

Validation of EKF

The proposed system was validated in real-time. Since ground truth is available in the simulator, we used synthetic flight data of a drone.

Compared methods

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 IMU.

Gyro+NDT

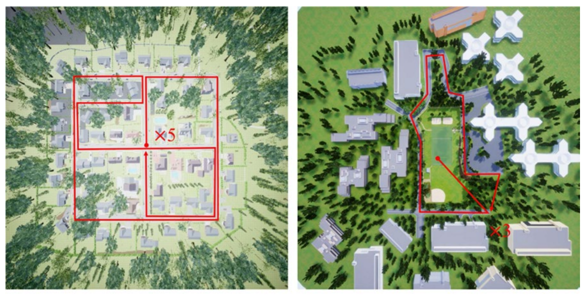
"Gyro+NDT" denotes NDT SLAM [4] using 32 layers of LiDAR. Angular velocity from a gyroscope, linear velocity of ground truth and NDT output are integrated in the EKF. Note that linear velocity of ground truth is available because the environment is a simulator.

Gyro+Regression

"Gyro+Regression" denotes an EKF-based estimation method integrating angular velocity from a gyroscope and gravity vectors from the regression network (Fig. 6). All outputs from the network are integrated in the EKF.

Gyro+MLE (ours)

"Gyro+MLE (ours)" denotes the proposed method described in "EKF-based self-attitude estimstion with DNN" section. The hyperparameters were set as $\mathrm{TH}_\beta = 8 \times 10^{-5}$ and $\gamma = 1 \times 10^4$. They were empirically determined based on the result in "Validation of DNN" section.



a Known environment **b** Unknown environment

Fig. 11 Flight courses. In (**a**), a course in "Neighborhood" which is a known environment for the DNN is shown. The drone flew for 5 rounds. It took about 22 minutes. In (**b**), a course in "SoccerField" which is an unknown environment for the DNN is shown. The drone flew for 3 rounds. It took about 6 minutes

Experimental conditions

Flight data of a drone ware recorded in "Neighborhood" and "SoccerField" of AirSim. "Neighborhood" is the same environment where the DNN was trained. An IMU and a camera are installed on a drone in the simulator. A LiDAR with 32 layers is also used for "Gyro+NDT". The sampling frequency of the IMU, the camera and LiDAR are approximately 100 Hz, 12 Hz and 20 Hz, respectively. The camera's horizontal FOV is 70 deg. Virtual noise was add 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 0.1 rad/s, 0.1 m/s^2, respectively. The flight courses of "Neighborhood" and "SoccerField" are shown in Fig. 11. A computer which has i7-6700 CPU

Table 11 MAE of estimated attitude during flight in known environment

	Roll [deg]	Pitch [deg]
Gyro	14.524	12.562
Gyro+Acc	2.920	3.380
Gyro+NDT	7.456	6.516
Gyro+Regression	3.187	1.876
Gyro+MLE (ours)	*2.869*	*1.821*

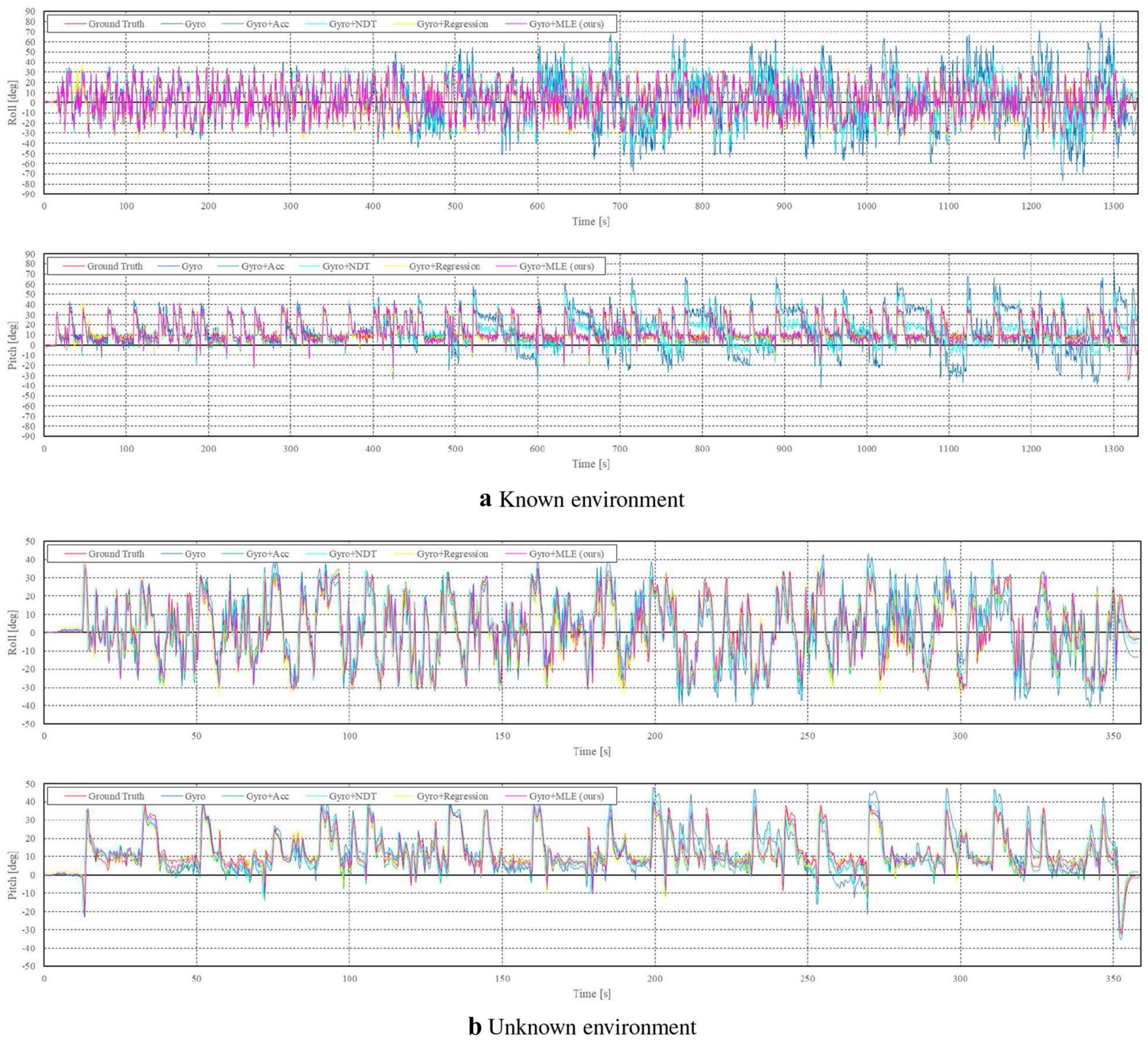


Fig. 12 Real-time attitude plotting. In (**a**), the result of the flight in "Neighborhood" is shown. In (**b**), the result of the flight in "SoccerField" is shown.

and GTX1080 GPU with 16 GB memory was used for the estimation. The DNN inference computation takes around 0.01–0.02 seconds with the computer. The proposed method is not only for UAVs, but it needs to be noted that this computer may be over-performance for real UAVs. Therefore, testing with a general UAV's computer specification should be necessary when the method is used on real UAVs, and it is not the focus of this study.

Experimental results

The estimated attitudes are plotted in Fig. 12a. Table 11 shows MAE of the estimated attitudes in "Neighborhood". MAE of "Gyro+MLE (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" does not have accumulative error. However it has error constantly, since the acceleration values of the sensor contain own acceleration of the robot. "Gyro+NDT" accumulated error slower than "Gyro" did by using LiDAR, but it could not remove the accumulation. "Gyro+Regression" and "Gyro+MLE (ours)" corrected the accumulative error by observing the estimated gravity. Comparing "Gyro+Regression" and "Gyro+MLE (ours)", filtering out the DNN outputs with large β is found valid. With "Gyro+MLE (ours)", 31 % of the outputs is rejected by the threshold in the flight.

MAE of the estimation in "SoccerField" is shown in Table 12. "Gyro+Regression" and "Gyro+MLE (ours)" suppressed accumulative error even in the environment where the DNN was not trained. MAE of "Gyro+MLE (ours)" is smaller than ones of the other methods. With "Gyro+MLE (ours)", more outputs (58 %) from the DNN are filtered out by the threshold TH_β in the unknown environment ("SoccerField") than in the known environment ("Neighborhood"). This can avoid observing outputs with high uncertainty. However, it leads chances of correcting error less. In other words, the proposed method can not correct error when the large variance is continuing. To compensate this decrease of the chances, the proposed method actually can integrates other observations such as IMU's acceleration, SLAM, and so on in a future work. Whereas, in this experiment, the proposed method integrates only angular rate and the DNN outputs in order to make the validation simple.

Table 12 MAE of estimated attitude during flight in unknown environment

	Roll [deg]	Pitch [deg]
Gyro	6.424	4.679
Gyro+Acc	3.155	3.091
Gyro+NDT	4.067	2.646
Gyro+Regression	3.248	1.984
Gyro+MLE (ours)	*2.869*	*1.785*

Conclusions and future work

EKF-based self-attitude estimation with DNN learning landscape information was proposed. The DNN estimates direction of gravity from a single shot image. The network outputs not only the gravity vector, but also a covariance matrix. Training was done with synthetic data of AirSim, and validation was done with both of the synthetic data and real sensors' data. In the validation, many samples with large error are filtered out by judging variance values. It means the proposed network expresses uncertainty of the prediction by outputting covariance matrices. The outputs from the DNN and angular velocity from a gyroscope are integrated in the EKF. The covariance matrix is used adjusting process noise. Moreover, outputs with too large variance are filtered out by a threshold. This EKF-based proposed method was validated with flight data of a drone in AirSim environments. In the experiments, the proposed method suppressed accumulative error by using the DNN.

In a future work, wider variety of datasets including real data are needed for the DNN to close the gap between the results in known environments and in unknown environments. Simulator data can be used for pre-training of the DNN before fine tuning with the real data. Only the roll augmentation is applied in this paper because it is easy and simple, but pitch augmentation by the homography transformation should also be valid for fine-tuning with the less real samples. Using other sensors or combining other methods is another future work. The experiments in this paper were done only with the daytime condition, but real robots should work in day and night. Therefor the future work needs to compensate the camera's weak point for night operations.

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 all the experiments, and drafted the manuscript. YK provided the inspiration for this study, provided advice, and checked and corrected the manuscript. Both 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
Dataset consisting of images and their corresponding gravity vectors. https://github.com/ozakiryota/dataset_image_to_gravity. .https://github.com/ozakiryota/image_to_gravity. https://github.com/ozakiryota/dnn_attitude_estimation.

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

Received: 8 September 2020 Accepted: 11 February 2021
Published online: 06 March 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) In: Probabilistic Robotics, pp. 309–336. The MIT Press
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 Transact Robotic 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. Transportat Res Part C Emerg Technol 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. do Monte Lima JPS, Uchiyama H, Taniguchi RI (2019) End-to-end learning framework for imu-based 6-dof odometry. Sensors 19(17):3777
11. 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 Transact Instrument Measure 69(1):24–34
12. 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
13. Shah S, Dey D, Lovett C, Kapoor A (2017) Airsim: High-fidelity visual and physical simulation for autonomous vehicles. Field Serv Robotics 5:621–635
14. 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)
15. Kalman RE (1960) A new approach to linear fi ltering and prediction problems. J Basic Eng 82:35–45
16. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. In: arXiv Preprint, pp. 1409–1556
17. Deng J, Dong W, Socher R, L. Li KL, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: Proceedings of 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 248–255
18. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. Proceed ICML 2010:807–814
19. 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
20. Kingma DP, Ba J (2015) Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference for Learning Representations (ICLR)

Publisher's Note