EKF-based real-time self-attitude estimation with camera DNN learning landscape regularities

Ryota Ozaki[1] and Yoji Kuroda[1]

***Abstract*— This paper presents an EKF (extended Kalman filter) based real-time self-attitude estimation method with a camera DNN (deep neural network) learning landscape regularities. The proposed DNN infers the gravity direction from a single shot image. It outputs the gravity direction as a mean vector and a covariance matrix in order to express uncertainty of the inference. It is pre-trained with datasets collected in a simulator. Fine-tuning with datasets collected with real sensors is carried out 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's outputs in an EKF. The covariance matrix output from the DNN is used as process noise of the EKF. Moreover, inferences with too large variance are filtered out before processing the integration in the EKF. Static validations are performed to show the DNN can infer the gravity direction with uncertainty expression. Dynamic validations are performed to show the DNN can be used in real-time estimation. Some conventional methods are implemented for comparison.**

I. INTRODUCTION

Estimating the attitude of a robot 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 multirotor. 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 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 update is also required. Some methods [8], [9] 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 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 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 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 for every estimation. It helps suppressing drift, noise, and accumulative error. This method is similar to our DNN. However, this method contains some problems. It cannot express uncertainty of the inference; 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, we presented a DNN inferring the gravity direction as mean and variance in order to express the uncertainty of the inference [17]. The validations in the paper shows the DNN can filter out inferences with large error by judging their variance values. However, the method can not output any estimation while the large variance is continuing, which means only static estimation is considered there. Another problem is that the DNN is tested on only simulator data in the paper. To use the DNN with real senors for real-time estimation, this paper presents an EKF-based method which integrates a gyroscope and the DNN fine-tuned with real data. By integrating a gyroscope, the method can estimate the attitude with higher frequency even when the DNN can not infer the gravity direction. The data augmentation method is also updated. It is more important in this paper because collecting real data is time-consuming. The updates from the previous study [17] are summarized here:

- Mirroring, rotation, and the homograpy transformation are applied to the image for augmenting data more,

[1]The authors are with Graduate School of Science and Technology, Meiji University, Kanagawa, 214-8571, Japan ce192021@meiji.ac.jp; ykuroda@meiji.ac.jp

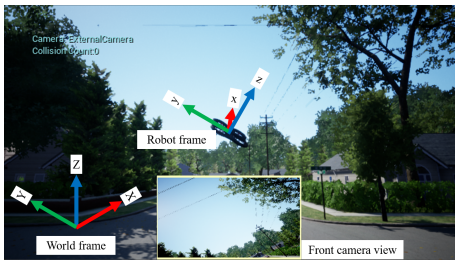


Fig. 1: Screenshot of AirSim with coordinate description. An IMU and a camera are equipped to the drone in the simulator. The purpose of this work is estimating attitude (roll and pitch) of the robot frame.

while the previous work applies only rotation.

- Both of synthetic and real data are applied to this paper's study, while the previous study was validated on only simulator data.
- Fine-tuning after pre-training is processed to applying the network to real data efficiently.
- The DNN inference is integrated in the EKF for real-time estimation, while the previous study evaluated only the DNN outputs not in real-time. The inferred covariance matrix is used for adjusting process noise of the EKF and for filtering out large variance.

The datasets and the source code used in this paper have been released in open repositories (see APPENDIX).

II. DNN ESTIMATING GRAVITY DIRECTION

The proposed method trains a DNN to learn landscape regularities for estimating a gravity vector in a robot frame. The gravity direction is expressed as mean and variance to consider the uncertainty of the inference.

A. Coordinate definition

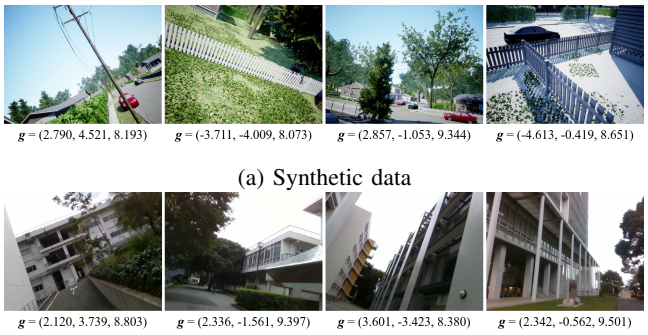
A world frame is defined as a standard right-handed coordinate system. A robot frame is defined as a right-handed coordinate system which is fixed on the robot pose. They are shown in Fig.1.

B. Dataset collection

Both of synthetic and real data are collected. The datasets consist of images and corresponded gravity vectors $\boldsymbol{g}$ in the robot frame. Fig.2 shows examples of the datasets.

The synthetic datasets are collected in AirSim [13]. AirSim is a simulator for drones, cars and more, built on Unreal Engine, which provides visually realistic graphics. An IMU and a camera are installed to a drone in the simulator in this work. The robot 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.

The real datasets are collected with a stick with an IMU (Xsens MTi-30) and a camera (RealSense D435) installed on a stick (Fig.3). The stick is hand-carried, and a image and a linear acceleration vector are recorded. They are saved



(a) Synthetic data

(b) Real data

Fig. 2: Examples of datasets. The dataset consists of images and corresponded gravity vectors $\boldsymbol{g}[\mathrm{m/s^2}]$ in the robot frame. The examples in (a) were collected in 'Neighborhood' of AirSim. The robot pose and weather parameters are randomized for creating the dataset. The examples in (b) were collected in the campus of Meiji University.

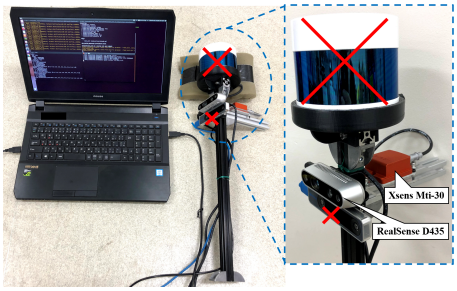


Fig. 3: Sensors installed on stick. Images and acceleration are recorded with this stick when it is still. The judge whether it is still is processed by programing. Note that depth information is not used in this study although RealSense D435 is a RGB-D camera.

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. Learning the static IMU is valuable because the DNN can reproduce it even in dynamic.

C. Data preprocessing

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

- Image (input data):
  Each image is flipped in 50% of probability for augmenting the dataset. After the flipping process, the homography transformation is randomly applied to the image for augmenting the pitch data. The virtual pitch variant $\Delta\theta$ is limited as [-10 deg, 10 deg]. The transformed height h', h'' and width w', w'' of the image are respectively computed as following. Fig.4 may help

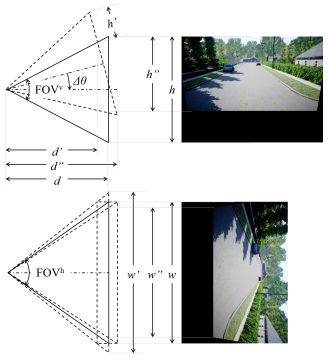


Fig. 4: Homography. The pitch data of the dataset is augmented by the homograpgy transformation.

understanding the equations.

$$
\begin{aligned}
d &= \frac{\frac{h}{2}}{\tan(\frac{\mathrm{FOV^v}}{2})} \\
d' &= \frac{d\cos(\frac{\mathrm{FOV^v}}{2})}{\cos(\frac{\mathrm{FOV^v}}{2} - |\Delta\theta|)}, \quad d'' = \frac{\frac{h}{2}\cos(\frac{\mathrm{FOV^v}}{2} - |\Delta\theta|)}{\sin(\frac{\mathrm{FOV^v}}{2})} \\
h' &= h/2 - d\tan\left(\frac{\mathrm{FOV^v}}{2} - |\Delta\theta|\right), \quad h'' = h - h' \\
w' &= \frac{d}{d'}w, \quad w'' = \frac{d}{d''}w
\end{aligned}
\tag{1}
$$

where h, w respectively denote the height and the width of the original image, and $\mathrm{FOV^v}$ ($< 2\pi$) denotes the camera's vertical field-of-view. The image is also 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. The RGB values are normalized following $\boldsymbol{mean} = (0.5, 0.5, 0.5)$ and $\boldsymbol{std} = (0.5, 0.5, 0.5)$. Fig.5 shows an example of the data augmentation. Note that the training time increases by about 4 times by adding the homography transformation according to our Python implementation. It does not influence the inferring time since the transformation is only applied to the training.

- Gravity vector (label data):
  The gravity vector is also transformed according to the image transformation. Since the network does not need to learn the norm of the gravity, L2 normalization is also applied to the vector in order to make the training efficient.

$$
\hat{\boldsymbol{g}} = \begin{cases} \boldsymbol{Rot}^{\mathrm{x}}_{(\Delta\phi)}\boldsymbol{Rot}^{\mathrm{y}}_{(\Delta\theta)}\frac{\boldsymbol{g}}{|\boldsymbol{g}|} & (\mathrm{w/o\ flip}) \\ \boldsymbol{Rot}^{\mathrm{x}}_{(\Delta\phi)}\boldsymbol{Rot}^{\mathrm{y}}_{(\Delta\theta)}\frac{(g_x, -g_y, g_z)^{\mathrm{T}}}{|\boldsymbol{g}|} & (\mathrm{w/\ flip}) \end{cases} \tag{2}
$$

where $\boldsymbol{Rot}^{\mathrm{x}}$, $\boldsymbol{Rot}^{\mathrm{y}}$ respectively denote rotation matrices along x, y axes.

D. Network

The proposed DNN is shown in Fig.6. It consists of CNN (convolutional neural network) layers and FC (fully connected) layers. The input to the network is the resized image, and the outputs are a mean vector of the gravity direction and a covariance matrix. Technically, the output of the final FC layer is $(\mu_x, \mu_y, \mu_z, L_0, \cdots, L_5)$, and the mean vector $\hat{\boldsymbol{\mu}}$ and the covariance matrix $\boldsymbol{\Sigma}$ are computed as Eq.3, 4, 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.

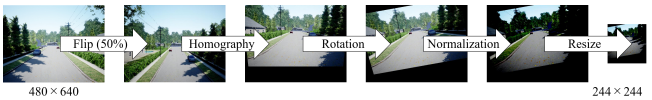


Fig. 5: Example of transformed image. Each input image is randomly flipped, transformed and rotated according to $\Delta\theta$ and $\Delta\phi$. This example shows an image when $\Delta\theta = \Delta\phi = 10$ deg. It is also resized, and is normalized.

$$
\hat{\boldsymbol{\mu}} = \frac{(\mu_x, \mu_y, \mu_z)^{\mathrm{T}}}{|(\mu_x, \mu_y, \mu_z)^{\mathrm{T}}|} \tag{3}
$$

$$
\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}, \quad \boldsymbol{L} = \begin{pmatrix} \exp(L_0) & 0 & 0 \\ L_1 & \exp(L_2) & 0 \\ L_3 & L_4 & \exp(L_5) \end{pmatrix} \tag{4}
$$

It is expected that the CNN layers lean extracting features such as edges, and the FC layers learn landscape information. Feature module of VGG16 [18] pre-trained on ImageNet [19] is adopted as the CNN layers of the proposed method. Transfer learning helps deep learning to be efficient even though the transferred network is trained on a different task [20]. All layers, except the final output layer, use the ReLU function [16] as activation function. All FC layers, except the final output layer, use the 10% Dropout [21] to avoid the over-fitting problem.

E. 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 proposed loss function l is computed as below.

$$
\begin{aligned}
l_{(\boldsymbol{\Theta})} &= -\sum_{\iota=0}^{\#D} \ln p(\hat{\boldsymbol{g}}_\iota | \hat{\boldsymbol{\mu}}_\iota, \boldsymbol{\Sigma}_\iota), \quad d = \mathrm{rank}(\boldsymbol{\Sigma}_\iota) \\
p(\hat{\boldsymbol{g}}_\iota | \hat{\boldsymbol{\mu}}_\iota, \boldsymbol{\Sigma}_\iota) &= \frac{\exp(-\frac{1}{2}(\hat{\boldsymbol{g}}_\iota - \hat{\boldsymbol{\mu}}_\iota)^{\mathrm{T}}\Sigma^{-1}(\hat{\boldsymbol{g}}_\iota - \hat{\boldsymbol{\mu}}_\iota))}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}_\iota|}}
\end{aligned}
\tag{5}
$$

where $\boldsymbol{\Theta}$ denotes the parameters of the network, $\#D$ denotes the number of samples in the dataset, and d denotes the dimensions of the variables, i.e. $d = 3$ in the proposed method. The network minimizes the loss by updating $\boldsymbol{\Theta}$.

F. Optimization

Adam (adaptive moment estimation) [22] is used to optimize the parameters. For the training with the synthetic data, the learning rates are set as $lr_{\mathrm{CNN}} = 0.00001$, $lr_{\mathrm{FC}} = 0.0001$, where lr_{CNN} is a value for the CNN layers, lr_{FC} is a value for the FC layers. For the fine-tuning with the real data, they are set smaller as $lr_{\mathrm{CNN}} = 0.000001$, $lr_{\mathrm{FC}} = 0.00001$.

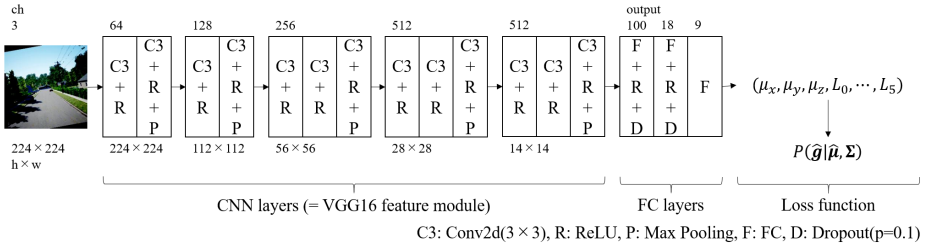


Fig. 6: Proposed network architecture. It consists of CNN layers and FC layers. The input data is a resized image, and the output data are a mean vector and a covariance matrix. They are computed with an output from the final layer as Eq.3, 4, respectively. Log-probability of multivariate normal distribution is used as a loss function of this model.

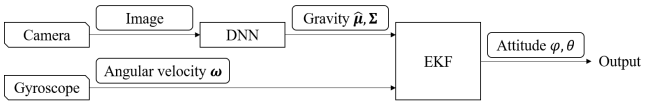


Fig. 7: Proposed EKF architecture. Gyroscopic angular rates are integrated in the prediction process in EKF. The DNN outputs are integrated in the update process in EKF.

G. Uncertainty expression

In this study, η in Eq.6 is assumed to express uncertainty of the inference. This value η is used to filter out inferences with large variance.

$$\eta = \sqrt{\sigma_x^2} \times \sqrt{\sigma_y^2} \times \sqrt{\sigma_z^2}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_z^2 \end{pmatrix} \quad (6)$$

III. EKF-BASED REAL-TIME ESTIMAION

The outputs form the DNN are integrated with gyroscopic angular rate in EKF. The proposed EKF architecture is shown in Fig.7. The state vector $\boldsymbol{x}$ of the proposed Kalman filter consists of roll ϕ and pitch θ of the robot. 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.

$$\boldsymbol{x} = \begin{pmatrix} \phi & \theta \end{pmatrix}^{\mathrm{T}} \quad (7)$$

Here, t denotes the time step in the following sections.

A. Prediction process

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

$$\begin{aligned} \bar{\boldsymbol{x}}_t &= f_{(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})} = \boldsymbol{x}_{t-1} + \boldsymbol{Rot}^{\mathrm{rpy}}_{(\boldsymbol{x}_{t-1})} \boldsymbol{u}_{t-1} \\ \boldsymbol{u}_{t-1} &= \boldsymbol{\omega}_{t-1} \Delta t = \begin{pmatrix} \omega_{\mathrm{x}_{t-1}} \Delta t \\ \omega_{\mathrm{y}_{t-1}} \Delta t \\ \omega_{\mathrm{z}_{t-1}} \Delta t \end{pmatrix} \end{aligned} \quad (8)$$

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.

$$\bar{\boldsymbol{P}}_t = \boldsymbol{J}_{\boldsymbol{f}t-1} \boldsymbol{P}_{t-1} \boldsymbol{J}_{\boldsymbol{f}t-1}^{\mathrm{T}} + \boldsymbol{Q}_{t-1}, \quad \boldsymbol{J}_{\boldsymbol{f}t-1} = \left. \frac{\partial f}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}} \quad (9)$$

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

B. Update process

Outputs of the DNN with large variance value η are rejected. A threshold TH_η is set for judging η, and only outputs with $\eta < \mathrm{TH}_\eta$ are observed in this EKF. The observation vector is $\boldsymbol{z}$ as below.

$$\boldsymbol{z} = \hat{\boldsymbol{\mu}} \quad (10)$$

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

$$h_{(\boldsymbol{x}_t)} = \boldsymbol{Rot}^{\mathrm{xyz}}_{(\boldsymbol{x}_t)} \frac{\boldsymbol{g}_{\mathbf{world}}}{|\boldsymbol{g}_{\mathbf{world}}|}, \quad \boldsymbol{g}_{\mathbf{world}} = \begin{pmatrix} 0 \\ 0 \\ g_{\mathrm{world}} \end{pmatrix} \quad (11)$$

where $\boldsymbol{g}_{\mathbf{world}}$ denotes a gravity vector in the world frame i.e. $g_{\mathrm{world}} \fallingdotseq 9.8\ \mathrm{m/s^2}$, and $\boldsymbol{Rot}^{\mathrm{xyz}}$ denotes a rotation matrix for vectors. The covariance matrix of the process noise is $\boldsymbol{R}$.

$$\boldsymbol{R} = \begin{pmatrix} \xi\sigma_x^2 & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \xi\sigma_y^2 & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \xi\sigma_z^2 \end{pmatrix} \quad (12)$$

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

$$\begin{aligned} \check{\boldsymbol{x}}_t &= \boldsymbol{x}_t + \boldsymbol{K}_t(\boldsymbol{z}_t - h_{(\boldsymbol{x}_t)}), \quad \check{\boldsymbol{P}}_t = (\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{J}_{\boldsymbol{h}t}) \boldsymbol{P}_t \\ \boldsymbol{J}_{\boldsymbol{h}t} &= \left. \frac{\partial h}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}_t}, \quad \boldsymbol{K}_t = \boldsymbol{P}_t \boldsymbol{J}_{\boldsymbol{h}t}^{\mathrm{T}} (\boldsymbol{J}_{\boldsymbol{h}t} \boldsymbol{P}_t \boldsymbol{J}_{\boldsymbol{h}t}^{\mathrm{T}} + \boldsymbol{R}_t)^{-1} \end{aligned} \quad (13)$$

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

IV. VALIDATION

A. Static validation of DNN

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

TABLE I: Dataset list.

id#	Environment		#samples	Usage
1	AirSim	Neighborhood	10000	Training
2			1000	Test
3		SoccerField	1000	Test
4	Real	Area I	1108	Fine-tuning
5		Area II	539	Test

TABLE II: Loss after 300 epochs of training.

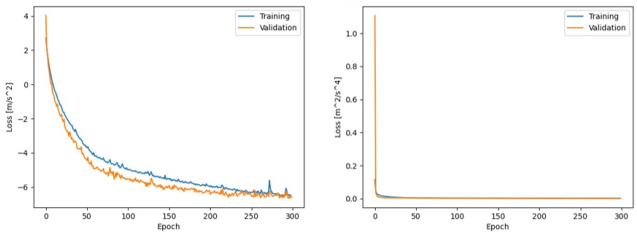
	Train (#1)	Test (#2)
MLE (ours) $[\mathrm{m/s^2}]$	-6.5002	-6.5961
Regression $[\mathrm{m^2/s^4}]$	0.0014	0.0031

1) Method list: Definitions of methods which were used in this validation are summarized here.

- 'MLE (ours, all)' denotes the proposed method described in Section II. MLE is short for Maximum Likelihood Estimation.
- 'MLE (ours, selected)' denotes a 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 the validation of attitude estimation. It means samples with large variance are filtered out as outliers. Assuming η in Eq.6 expresses uncertainty of the inference, samples with small variance are selected with a threshold TH_η. In this validation, the threshold is set as $\mathrm{TH}_\eta = \frac{1}{\#D}\sum_{\iota=0}^{\#D}\eta_\iota$, where $\#D$ is the number of samples in the testing dataset.
- 'Regression' denotes a network which the final FC layer is difference from 'MLE (ours)'. It outputs a 3D gravity vector without covariance. It is implemented base on the related work [14]. L2 normalization is applied to the final layer while ReLU is applied in [14], since ReLU outputs only positive values. MSE (mean square error) between the labels and outputs is used as a loss function.
- '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 baseline in this study.

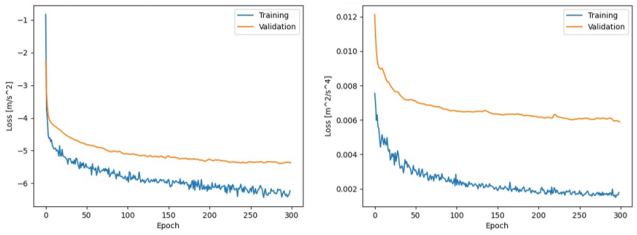
2) Training: The datasets used in this validation are listed in Table I. The network was trained with 10000 synthetic samples (dataset#1) with a batch size of 200 samples for 300 epochs. Another 1000 samples (dataset#2) were used for test. They were collected in 'Neighborhood' of AirSim. The training 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 around 43 hours with the computer.

The loss values during the training are plotted in Fig.8. The regression model converged much faster than the MLE model did. Table II shows the loss values after 300 epochs of training. Note that the loss function of the MLE model and one of the regression model are difference.



(a) MLE (ours) (b) Regression

Fig. 8: Loss plotting of training. Note that the loss function of the MLE model and one of the regression models are difference. Therefore, their values can not be simply compared.



(a) MLE (ours) (b) Regression

Fig. 9: Loss plotting of fine-tuning. The fine-tuning made the loss values on the real data smaller.

3) Fine-tuning: Fine-tuning with the real data was done after the training with the synthetic data. The network was tuned with 1108 real data samples (dataset#4) with a batch size of 200 samples for 300 epochs. Another 539 samples (dataset#5) were used for test. They were collected in the campus of Meiji University. The training dataset and the test dataset were collected in the same campus, but not in the same area.

The loss values during the fine-tuning are plotted in Fig.9. Table III shows the loss values after 300 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, wider variety of datasets are needed for training.

4) Attitude estimation: The roll ϕ and pitch θ of the camera pose in the gravitational coordinate are estimated by using $\hat{\boldsymbol{\mu}}$.

$$\phi = \tan^{-1}\frac{\hat{\mu}_y}{\hat{\mu}_z}, \quad \theta = \tan^{-1}\frac{-\hat{\mu}_x}{\sqrt{\hat{\mu}_y^2+\hat{\mu}_z^2}} \tag{14}$$

The MAE (mean absolute error) of the estimation on the synthetic datasets is shown in Table IV. With 'MLE (ours, selected)', 795 samples which has $\eta < \mathrm{TH}_\eta = \frac{1}{\#D}\sum_{\iota=0}^{\#D}\eta_\iota = 0.000120\ \mathrm{m^3/s^6}$ were selected from 1000 test samples in dataset#2. This threshold was also used for the other datasets. The number of samples selected by the threshold are shown in Table V. The MAE of the estimation on the real datasets is shown in Table VI. With the test datasets, the error of 'MLE (ours, selected)' is smaller than the others.

TABLE III: Loss after 300 epochs of fine-tuning.

	Train (#4)	Test (#5)
MLE (ours) $[\mathrm{m/s^2}]$	-6.2295	-5.3756
Regression $[\mathrm{m^2/s^4}]$	0.0018	0.0059

TABLE IV: MAE of static estimation on synthetic data.

Method	Angle [deg]	Dataset# 1	2	3
MLE (ours, all)	Roll	1.206	1.982	3.535
	Pitch	1.022	1.992	6.919
MLE (ours, selected)	Roll	1.014	**1.368**	**1.800**
	Pitch	**0.824**	**1.121**	**2.478**
Regression	Roll	**1.012**	1.948	2.673
	Pitch	0.852	1.902	3.693
Statistics	Roll	15.080	15.344	15.352
	Pitch	14.998	14.783	14.408

Comparing 'MLE (ours, all)' and 'MLE (ours, selected)', filtering by TH_η is found valid, which means the network expresses the uncertainty by outputting a covariance matrix. A good example with large η and one with small η are shown in Fig.10. Obviously, the sample in Fig.10a has much less landscape information to estimate the gravity direction, and the proposed network expresses the uncertainty with large η. There is no way to detect it by the conventional regression model.

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.

Comparing with the previous study [17], the result in this paper is better. It implies the improvement of the data augmentation contributes to the accuracy. Data augmentation is especially important for real data because collecting real data is time-consuming.

B. 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.

1) Method list: Definitions of methods which were used in this validation are summarized here.

- 'Gyro' denotes an estimation method integrating angular velocity from a gyroscope.
- 'Gyro+Acc' denotes an EKF-based estimation method integrating angular velocity and linear acceleration from an IMU.
- '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.
- 'Gyro+Regression' denotes an EKF-based estimation method integrating angular velocity from a gyroscope and gravity vectors inferred by the regression network.
- 'MLE w/o rejection' denotes a method using the proposed DNN directly without EKF. It does not filter out any inferences in order to estimate the attitude continuously.
- 'Gyro+MLE (ours)' denotes the proposed method described in Section III. The hyperparameters were set as $\mathrm{TH}_\eta = 1.2 \times 10^{-4}\ \mathrm{m^3/s^6}$ and $\xi = 5 \times 10^3$. They were empirically determined based on the result in Section IV-A.

TABLE V: Number of selected samples by MAE (ours, selected).

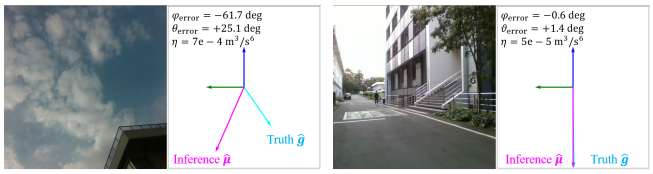
#Selected samples (percentage [%])	Dataset# 1	2	3	4	5
Before fine-tuning	8388 (83.9)	795 (79.5)	368 (36.8)	672 (60.6)	225 (41.7)
After fine-tuning	- -	- -	- -	883 (79.7)	304 (56.4)

TABLE VI: MAE of static estimation on real data.

Method		Angle [deg]	Dataset# 4	5
Before fine-tuning	MLE (ours, all)	Roll	2.272	3.484
		Pitch	4.816	5.828
	MLE (ours, selected)	Roll	**1.762**	**1.951**
		Pitch	**4.043**	**4.551**
	Regression	Roll	2.217	3.140
		Pitch	4.292	5.612
After fine-tuning	MLE (ours, all)	Roll	1.505	2.992
		Pitch	1.349	3.273
	MLE (ours, selected)	Roll	**1.253**	**1.656**
		Pitch	**1.114**	**2.364**
	Regression	Roll	1.264	2.567
		Pitch	1.296	3.121
Statistics		Roll	15.803	14.125
		Pitch	10.277	13.222

2) Experimental conditions: Flight data of a drone was recorded in 'Neighborhood' and 'SoccerField' of AirSim. The sampling frequency of the IMU and the camera are approximately 100 Hz, 12 Hz, respectively. Virtual noise was add to the IMU's 6-axis data. It was randomly added following a normal distribution with a mean of 0 $\mathrm{rad/s}$, 0 $\mathrm{m/s^2}$ and a standard deviation of 0.1 $\mathrm{rad/s}$, 0.1 $\mathrm{m/s^2}$, respectively. The flight courses are shown in Fig.11a, 11b. 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 0.01 - 0.02 seconds with the computer.

3) Experimental results: The estimated attitudes in 'Neighborhood' are plotted in Fig.12. Table VII shows the MAE of the estimated attitude. The 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 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+Regression' and 'Gyro+MLE (ours)' corrected the accumulative error by ob-



(a) Large η example (b) Small η example

Fig. 10: Examples with η values.

(a) Known environment (b) Unknown environment (c) Real world

Fig. 11: Driving courses. The AirSim's drone flew in 'Neighborhood' (a) for 5 rounds for about 22 minutes, and 'SoccerField' (b) for 3 rounds for about 6 minutes, respectively. The real sensors were carried in the campus (c) for around 5 minutes.

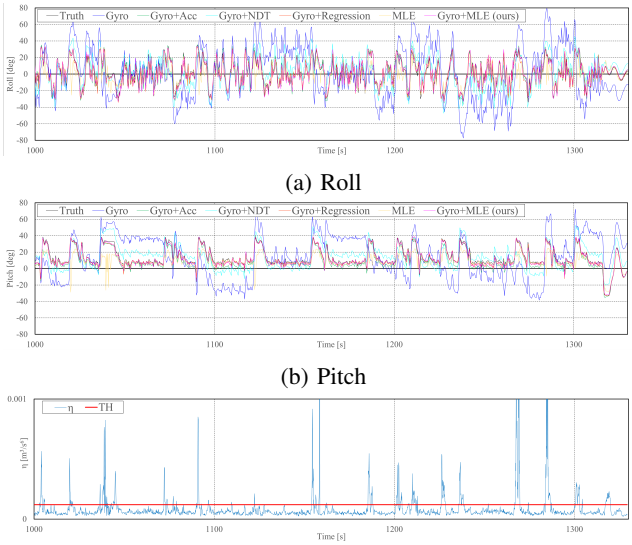
serving the estimated gravity. Comparing 'Gyro+Regression' and 'Gyro+MLE (ours)', filtering out the DNN outputs with large η is found valid. The error difference between them in the training environment ('Neighborhood') is quite small. It is considered to be because the DNNs fit the training dataset well, which led to the environment having less uncertainty to be filtered out. In the unknown environment ('SoccerField'), 'Gyro+MLE (ours)' showed stronger improvement over 'Gyro+Regression'. With 'Gyro+MLE (ours)', about 13 % and 43 % of the inferences were rejected by the threshold TH_η during the flight in 'Neighborhood' and 'SoccerField', respectively. This can avoid observing outputs with high uncertainty. Especially, 'MLE w/o rejection' tended to output large error and large variance when the attitude angles exceeded the trained range i.e. [-30 deg, 30 deg].

C. Validation of real-time estimation in real world

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

1) Indoor experiment with motion capture: The sensors were hand-carried in an indoor environment of 4.5 m × 6 m for about 23 minutes. 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 VIII shows the MAE of the estimated attitude. The proposed method suppressed accumulation of error also in real world. In the flat indoor environment, the MAE given by the proposed method was 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



(a) Roll

(b) Pitch

(c) η

Fig. 12: Real-time plotting in 'Neighborhood'. The graphs show the last 330 seconds of the synthetic flight. 'MLE' tended to output large variance η when the error is large.

TABLE VII: MAE of dynamic estimation in simulator.

Method	Neighborhood		SoccerField	
	Roll [deg]	Pitch [deg]	Roll [deg]	Pitch [deg]
Gyro	14.524	12.562	6.424	4.679
Gyro+Acc	2.920	3.380	3.155	3.091
Gyro+NDT	7.456	6.516	4.067	2.646
Gyro+Regression	2.793	1.645	3.293	2.049
MLE w/o rejection	5.016	5.711	4.098	4.926
Gyro+MLE (ours)	**2.703**	**1.598**	**2.949**	**1.777**

it would be more stable estimation. For reference, the error given by 'Gyro+Acc+MLE' was ϕ_{error} = 2.290 deg, θ_{error} = 1.618 deg.

2) 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 in real world.

The sensors were hand-carried for around 5 minutes in Area II (Fig.11c) 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 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.13, and the ground truth was assumed as $\phi_{\mathrm{gt}} = 0$ deg, $\theta_{\mathrm{gt}} = 0$ deg. This evaluation method is based on the related study [14].

Table IX shows the error of the estimation at the last pose. The proposed method suppressed accumulation of error during the driving outdoor.

TABLE VIII: MAE of dynamic estimation in mocap area.

	Roll [deg]	Pitch [deg]
Gyro	6.012	5.100
Gyro+Acc	2.509	**1.648**
Gyro+Regression	2.840	2.764
MLE w/o rejection	4.808	7.858
Gyro+MLE (ours)	**2.407**	1.902

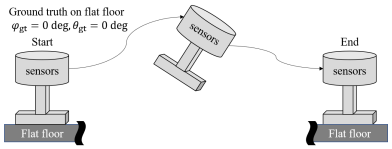


Fig. 13: Dynamic experiment. The ground truth on the flat floor is assumed to be $\phi_{\mathrm{gt}} = 0$ deg, $\theta_{\mathrm{gt}} = 0$ deg.

V. 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 a single shot image. The network outputs not only the gravity vector, but also a covariance matrix. 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 with the uncertainty. For dynamic estimation, angular rates from a gyroscope and the DNN's outputs are integrated in the EKF. The inferred covariance matrices are used for adjusting the process noise and for filtering out infers with large variance. The dynamic experiments showed the proposed method can be used for real-time estimation.

Using multiple cameras or other sensors for estimating the attitude is our future work.

APPENDIX

- Source code and dataset. The code is implemented using Python, C++, PyTorch API and ROS API.

  `https://github.com/ozakiryota/dnn_attitude_estimation`

ACKNOWLEDGMENT

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.

TABLE IX: Error of estimated attitude at last pose in outdoor.

	Roll [deg]	Pitch [deg]
Gyro+MLE (ours)	**+0.643**	**+1.851**
Gyro	+5.268	-5.047

REFERENCES

[1] J. Vaganay, M. J. Aldon and A. Fournier, Mobile robot attitude estimation by fusion of inertial data, Proceedings of 1993 IEEE International Conference on Robotics and Automation (ICRA), pp.277–282, 1993.

[2] S. Thrun, W. Burgard and D. Fox, Probabilistic Robotics, The MIT Press, pp.309–336, 2005.

[3] S. Rusinkiewicz and M. Levoy, Efficient Variants of the ICP Algorithm, Proceedings of Third International Conference on 3-D Digital Imaging and Modeling, pp.145–152, 2001.

[4] P. Biber and W. Straßer, The Normal Distributions Transform: A New Approach to Laser Scan Matching, Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp.2743–2748, 2003.

[5] J. Engel, J. Stueckler and D. Cremers, LSD-SLAM: Large-Scale Direct Monocular SLAM, Proceedings of European Conference on Computer Vision (ECCV), pp.834–849, 2014.

[6] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, ORB-SLAM: A Versatile and Accurate Monocular SLAM System, IEEE Transactions on Robotics, Vol.31, No.5, pp.1147–1163, 2015.

[7] M. A. Quddus, W. Y. Ochieng and R. B. Noland, Current map-matching algorithms for transport applications: State-of-the art and future research directions, Transportation Research Part C: Emerging Technologies, Vol.15, No.5, pp.312–328, 2007

[8] P. Kim, B. Coltin and H. J. Kim, Linear RGB-D SLAM for Planar Environments, Proceedings of European Conference on Computer Vision (ECCV), pp.333–348, 2018.

[9] M. Hwangbo and T. Kanade, Visual-inertial UAV attitude estimation using urban scene regularities, Proceedings of 2011 IEEE International Conference on Robotics and Automation (ICRA), pp.2451–2458, 2011.

[10] J. P. Silva do Monte Lima, H. Uchiyama and R. I. Taniguchi, End-to-End Learning Framework for IMU-Based 6-DOF Odometry, Sensors 2019, Vol.19, No.17, 3777, 2019.

[11] M. K. Al-Sharman, Y. Zweiri, M. A. K. Jaradat, R. Al-Husari, D. Gan and L. D. Seneviratne, Deep-Learning-Based Neural Network Training for State Estimation Enhancement: Application to Attitude Estimation, IEEE Transactions on Instrumentation and Measurement, Vol.69, No.1, pp.24–34, 2020.

[12] M. Mérida-Floriano, F. Caballero, D. Acedo, D. García-Morales, F. Casares and L. Merino, Bioinspired Direct Visual Estimation of Attitude Rates with Very Low Resolution Images using Deep Networks, Proceedings of 2019 IEEE International Conference on Robotics and Automation (ICRA), pp.5672–5678, 2019.

[13] S. Shah, D. Dey, C. Lovett and A. Kapoor, AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles, Field and Service Robotics, pp.621–635, 2018.

[14] G. Ellingson, D. Wingate and T. McLain, Deep visual gravity vector detection for unmanned aircraft attitude estimation, Proceedings of 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp.5557–5563, 2017.

[15] R. E. Kalman, A new approach to linear fi ltering and prediction problems, Journal of Basic Engineering, Vol.82, pp.35–45, 1960.

[16] V. Nair and G. E. Hinton, Rectified linear units improve restricted boltzmann machines, Proceedings of ICML 2010, pp.807–814, 2010.

[17] R. Ozaki, and Y. Kuroda, DNN-based self-attitude estimation by learning landscape information, Proceedings of 2021 IEEE/SICE International Symposium on System Integration (SII2021), pp.733–738, 2021.

[18] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556, 2014.

[19] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, ImageNet: A large-scale hierarchical image database, Proceedings of 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.248–255, 2009.

[20] S. J. Pan and Q. Yang, A Survey on Transfer Learning, IEEE Transactions on Knowledge and Data Engineering, Vol.22, No.10, pp.1345–1359, 2010.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research, Vol.15, No.1, pp.1929–1958, 2014.

[22] Diederik P. Kingma and Jimmy Ba, Adam: A method for stochastic optimization, Proceedings of the 3rd International Conference for Learning Representations (ICLR), 2015.