Orientation Estimation Piloted by Deep Reinforcement Learning

Miaomiao Liu, Sikai Yang, Arya Rathee, Wan Du
Computer Science and Engineering
University of California, Merced, USA
{mliu71, syang126, arathee, wdu3}@ucmerced.edu

***Abstract*—Inertial Measurement Unit (IMU) sensors are commonly used for estimating device orientation. However, due to the irregular movements of devices and distortions of magnetic fields, IMU sensors may present varying data quality. Conventional data fusion approaches such as Complementary Filter (CF) and Kalman Filter struggle to adapt to these variations. Recent efforts have explored the utilization of deep learning to directly infer orientation from IMU sensor data. Nevertheless, when facing new scenarios that have different data distributions from training (e.g., different movement patterns or magnetic fields), deep learning methods cannot accurately infer orientation. In this paper, we conduct extensive experiments and identify two critical parameters for CF-based orientation estimation. We propose employing deep learning to adjust these two parameters, rather than directly inferring the final orientation outcomes. Since the relationship between sensor data and the settings of CF parameters is relatively simpler than the relationship between sensor data and orientation, a deep learning model of the same size can learn the first relationship more effectively and efficiently. We develop *DRLPilot* which leverages Deep Reinforcement Learning (DRL) to pilot CF-based orientation estimation based on the data quality of IMU sensors. Our DRL framework incorporates novel state design and reward function to accommodate the unique features of IMU sensor data and orientation estimation. Extensive experiment results demonstrate *DRLPilot* outperforms baseline systems by 27% in orientation accuracy.**

***Index Terms*—Inertial Measurement Unit, Orientation Estimation, Deep Reinforcement Learning**

I. Introduction

Orientation estimation is crucial for many mobile applications, including virtual reality [1], [2], augmented reality [3], and gaming [4]. Inertial Measurement Unit (IMU) sensors on mobile devices are commonly used for orientation estimation. Gyroscopes measure angular velocity. With an initial orientation, we can track the device's orientation by integrating angular velocity over time. However, sensor noise is also integrated over time. Accumulated noise causes orientation drift rapidly [5], [6], [7], [8], [9]. Accelerometers can measure the gravity direction of the Earth. Magnetometers measure the magnetic field direction of the Earth. Using gravity direction and magnetic field direction as references, the gyroscope-based orientation tracking can be constantly calibrated. However, when the device is accelerating, accelerometers cannot accurately measure gravity direction. Meanwhile, metal can distort magnetic fields, tilting the magnetometer away from north.

Various classical sensor fusion methods have been proposed for orientation estimation, such as Complementary Filter (CF) and Kalman Filter (KF) [10], [11], [12]. For example, a recent work, MUSE [13], combines the data from three IMU sensors by a complementary filter for orientation estimation. These classical sensor fusion methods do not rely on training, so they can be used to new scenarios easily. However, they need careful parameter tuning, which requires expert knowledge and extensive trial-and-error experiments.

Recent studies, such as IDOL [9] and RTAT [14], leverage deep learning to train a model by labeled data. These models take the raw IMU sensor data as input and outputs the device's orientation. If well trained, they demonstrate the ability to provide accurate orientation predictions. However, these methods struggle to adapt to new scenarios where the movement patterns or magnetic fields are different from the scenario where the training data is collected. For instance, for a model trained by the data collected at one place, it may fail to do prediction at a different place, since the magnetic field in the new place may be different from that in the training data. It is labor-intensive to collect training data that covers all scenarios such as diverse movement patterns or magnetic fields. Moreover, modeling the relationship between raw IMU sensor data and orientation results becomes complex if we have the data that covers all scenarios. Consequently, a small neural network model may struggle to effectively learn this intricate relationship due to its limited modeling capacity. This problem is especially severe for orientation estimation, since orientation tracking is an iterative process, i.e., the orientation estimated at one time step will be used as the basis for the subsequent time steps.

By comparing the above two kinds of methods, we find that deep learning methods are capable of utilizing ground-truth data to automatically tune parameters and learn non-linear relationships without requiring explicit modeling of the system dynamics, while classical data fusion methods can easily scale to different scenarios. In this paper, we propose to combine both advantages by integrating deep learning into classical orientation estimation (using a CF). We observe two primary parameters in CF: α and β that control the weight of accelerometer and magnetometer respectively. They also reflect the trustworthiness of these two sensors. Extensive

This publication was supported in part by NSF Grant #2008837. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

experiments show that α and β greatly impact the accuracy of CF-based orientation estimation. Their best setting also varies over time, due to the variation of sensor data quality.

We develop *DRLPilot*, a CF-based orientation estimation system that can dynamically adapt its parameter configurations using a Recurrent Neural Network (RNN). With the deep learning module piloting the classical orientation estimation, *DRLPilot* can dynamically adapt to sensors' varying quality. Instead of directly outputting the orientation result (a 3×3 matrix), the deep learning module now predicts α and β (two scalars), which is a relatively lightweight task. Additionally, the potential strategy to determine α and β is based on the observation on sensors' data quality, which is ubiquitous for all scenarios. Therefore, this framework grants *DRLPilot* stronger modeling capacity and generalizability.

Due to the absence of the ground truth α and β, supervised learning cannot be adopted for learning these two parameters. Even though ground-truth orientation can be obtained using an external device like a VR system, the labeled orientation data cannot be directly used to train our neural network. First, as orientation does not serve as the output of the neural network model in *DRLPilot*, we are unable to utilize the labeled orientation for computing a loss that could drive gradient descent and back propagation. Second, the orientation at the current time step is influenced by the orientation at all previous time steps, which makes it almost impossible to acquire the real ground truth of α and β based on the ground-truth orientation. Our experiments also show that calculating optimal α and β for each time step individually fails to obtain the ground truth of these two parameters (Section III).

To tackle the above challenges, we design a Deep Reinforcement Learning (DRL) framework to learn the best α and β setting through ground-truth orientation. The DRL agent is a neural network model that takes the features we extract from IMU sensor as inputs, and outputs the setting of α and β at each time step. The inferred setting will be used to configure CF for orientation estimation. Our DRL-based orientation estimation framework proposes two novel designs. First, to guide the learning process, our DRL module takes *some physics-aware features* extracted from IMU data as inputs, instead of raw sensor data. Second, *a customized reward* is designed by considering both orientation error and time-series orientation estimation. The reward is used to update the agent.

To facilitate DRL training, we extract a set of features from the raw implicit IMU readings that are relevant to α and β. For example, we propose a novel approach that uses raw IMU sensor data to predict the errors in accelerometer and magnetometer measurements, which are directly related to α and β determination. It utilizes the gyroscope's temporal accuracy to measure angular errors of accelerometer and magnetometer. We incorporate these two errors and many other digested information into the state design of our DRL framework (Section IV-B).

We design a customized reward function to better reflect the quality of the DRL's actions and further boost its training efficiency. Since the orientation at a time step will be used as the basis to estimate future orientations, the α and β setting at a time step will influence not only current orientation estimation but also the orientation in a short future. Therefore, the setting of these two parameters may have a delayed impact on the orientation accuracy in the future. To that end, we design an inertial reward, which allows the delayed impact of α and β to be aggregated into the reward, and thus can better reflect the quality of DRL outputs.

We utilize the dataset provided by [15] to evaluate *DRLPilot*. The authors collect this dataset from five volunteers at two distinct places, with a total data collection duration of approximately 500 minutes. The data was sampled at a frequency of 50 Hz, resulting in an aggregate of approximately 1,500,000 samples. We augment the dataset by 12x as introduced in Section V to conduct training and test. Additionally, we collect more data at three new locations following the same data collection method to evaluate the performance of *DRLPilot*. The newly collected data has approximately 215,040 data samples. From our experiments, *DRLPilot* reduces the orientation error by 27% compared to baseline systems.

In summary, this paper makes the following contributions:

- We analyze the advantages and disadvantages of classical and deep learning based orientation estimation methods.
- We develop a *DRLPilot* system to combine advantages from both methods in a DRL framework. The key design principle of combining deep learning with classical methods has the potential to be applied to other tasks beyond IMU orientation estimation.
- We boost the DRL performance by customized state and reward designs.
- Extensive experiments are conducted to validate the effectiveness of *DRLPilot*.

II. Experimental Study on Existing Solutions

Classical orientation estimation methods rely on physical observations to track orientation, e.g., angular velocity integration and calibrations via gravity and magnetic field directions. As an example, the state-of-the-art solution, MUSE [13], adopts a Complementary Filter (CF) that fuses the IMU sensors to track orientation. However, MUSE requires experts' effort and experience to tune the parameters. On the other hand, deep learning is capable of utilizing the ground truth to automatically optimize its neural network parameters, via gradient descent and back propagation. When properly trained, the latest Recurrent Neural Network (RNN) based orientation estimation solution, RTAT [14], can achieve better performance than classical methods.

However, RTAT fails to maintain its high orientation estimation accuracy when the scenarios become complex. In this section, we test the performance of RTAT at different places, and with different facing directions of the user.

A. RTAT's Performance at Different Places.

Besides the regular 'room' dataset, there is also another dataset collected in a 'hallway' in [14], where the magnetic field is distorted, making its magnetometer data pattern different from the 'room' dataset. With the two different datasets,

TABLE I
RTAT ERROR ON DIFFERENT DATASETS

Train / Test	Room	Hallway	Hybrid
Room	**11.78**°	21.44°	13.87°
Hallway	84.33°	**17.47**°	30.56°
Hybrid	48.56°	19.43°	**22.32**°

we train three separate models, respectively using ’room’ data only, ’hallway’ data only, and both datasets together. For simplicity, we use room model, hallway model, and hybrid model to represent these models. We also test these models’ performance on different datasets, as shown in Table I. In our experiments, the testing data and the training data may come from the same dataset, but they never overlap.

From Table I, when a model is trained with ’room’ data and tested on ’room’ data, its error is 11.78°. When another model is trained with ’hallway’ data and tested on ’hallway’ data, the error is 17.47°. The ’hallway’ has higher magnetic distortion (31°) compared to the ’room’ according to [14]), indicating lower quality of magnetometer measurements. However, when a hybrid model is trained and test on the two combined datasets, its error is 22.32°, which is higher than the two separately trained models. This is because the learning task is becoming too complex for the model to learn from two distinct data distributions, and thus the modeling complexity exceeds its modeling capacity, causing the model to underfit the combined datasets.

In comparison, although classical methods are also impacted by magnetic distortions, they are not affected by the task complexity as much, i.e. their performance on hybrid data will simply be the average, since they do not require training.

B. RTAT’s Performance with Different Facing Directions.

The data used by RTAT [14] is for the arm tracking task, in which users wear a smartwatch and perform daily movements like drawing and exercising. We analyze the data used by RTAT and find it is collected with users roughly facing north (with difference < 29°), which implies the magnetic field is pointing forward. Additionally, users do not change their facing direction or move their shoulder much, due to the constraint of the arm tracking task. This constraint is also shared by other arm tracking studies [13], [16], [17]. However, this constraint does not apply in the task of orientation estimation, where users may perform any movements, like turning around and walking freely. To test the performance of RTAT with different facing directions of the users, we generate augmented data based on the original ’room’ data from RTAT, in which a virtual user performs the same original movement but with different facing directions: 360° from north to west, south and east, with a step length of 30°. Details of data augmentation can be found in Section V.

First, we test the original RTAT model, i.e. the model trained using the original data without augmentation, on the augmented data with different facing directions. Model1 in Figure 1 demonstrates our testing results. The radius of the outer circle stands for an error of 60°, and the distance from each dot to the center of the circle represents the orientation error at the corresponding facing direction. As we can see, on the original facing direction (0°), the model performs the best, with an error of 11.78°, since the model is trained on the same direction. As the test data’s facing direction changes, the error increases. Specifically, at 150°, 180°, and 210°, the error is 52.02°, 53.03°, 52.98° respectively.

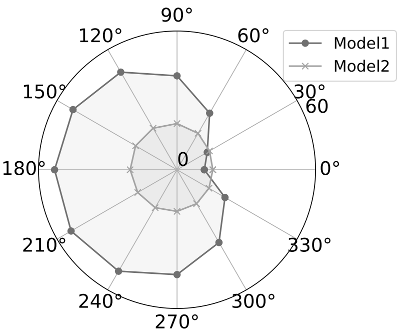


Fig. 1. RTAT Performance of a specialized model and a generalized model.

Second, we train a Model2 using all augmented data from 0° to 360°, and also test its performance on different facing directions. As shown in Figure 1, comparing to Model1, Model2 averages the performance on all directions and achieves overall error of 18.21°, 54.86% lower than the average error of Model1, since it is trained on all directions. However, this comes at a cost of the best performance, at a 0° (18.21° v.s. 11.78°). This shows that when the training data size increases by 12×, RTAT struggles to learn all data effectively. The modeling task for the neural network exceeds its modeling capacity and results in under-fitting and increased error.

In comparison, the facing direction hardly affects the performance of classical methods. This is because the movements of the users remain same across the augmented data, and the facing direction does not affect the accuracy of the physical IMU sensor fusion.

Discussion: The neural network in RTAT is responsible for modeling the entire orientation estimation process, from IMU sensor data inputs to orientation outputs. In complex scenarios, the learning task of the neural network becomes heavy, and the performance of RTAT falls behind due to its limited modeling capacity. Simply increasing the size of the neural network may not solve this problem, since a larger neural network requires more resources and needs much more training data to converge, resulting in significant increased labor cost on training data collection. In this paper, we investigate one question: can we achieve accurate orientation estimation at different places and different directions with similar model size as RTAT?

III. PRELIMINARY DESIGN

In this section, we propose to combine the advantages of both classical methods and deep learning methods into one system. The key in designing such a system is to enable a deep learning module to automatically optimize the parameters of a classical orientation estimation module, while letting the latter fuse the data from three IMU sensors.

A. Two Important CF Parameters

For the classical orientation estimation module, we use the CF, as MUSE [13] does. Comparing to another popular algorithm, Kalman Filter, CF does not require the sensor noise model to be Gaussian. It is commonly used in navigation systems, where data from gyroscopes, accelerometers, and magnetometers are fused to provide orientation information.

1) Parameter α and β in CF: CF integrates the gyroscope readings over time to track orientation. However, noises and biases in the gyroscope readings are also accumulated along with gyroscope integration, causing drifts in the gyroscope-based orientation results. CF also uses an accelerometer and a magnetometer to provide calibrations for the orientation results, since they measure the reference directions: gravity and north, respectively. However, due to the relatively high noise in the accelerometer and magnetometer readings, they are granted with weighting coefficients. This ensures that they contribute to smooth calibrations rather than overwhelming the gyroscope-based orientation results. Putting everything together, CF updates the orientation by Equation (1):

$$\Theta^t = \Theta^{t-\Delta t} \cdot R^t_{gyro} \cdot \alpha R^t_{acc} \cdot \beta R^t_{mag} \tag{1}$$

In Equation (1), R^t_{gyro} is the orientation from the integration of gyroscope readings at time step t, which relies on the orientation at the previous time step $\Theta^{t-\Delta t}$. R^t_{acc} and R_{mag} are the rotation matrices derived from the accelerometer and magnetometer calibrations at time step t. α and β are the weighting coefficients that determine how much Θ^t relies on accelerometer and magnetometer, respectively. They control the percentage of the accelerometer and magnetometer calibrations to be incorporated, and typically set between 0 and 1. A higher value of α assigns a greater weight to the accelerometer, and a higher value of β assigns a greater weight to the magnetometer.

2) Importance of Choosing the Best α and β: Apparently, α and β are directly related to the quality of the accelerometer and magnetometer, respectively. Conversely, the value of α and β should reflect the trustworthiness of the accelerometer and magnetometer in order to achieve the best accuracy. For instance, if the linear acceleration is high, the accelerometer readings contain both linear acceleration and gravity, leading to erroneous calibration. In such cases, α should be decreased. Similarly, in the presence of significant magnetic distortion, the magnetometer may not point north accurately, and its calibration may be compromised. As a result, β should be decreased to mitigate the impact of magnetometer inaccuracies on orientation estimation.

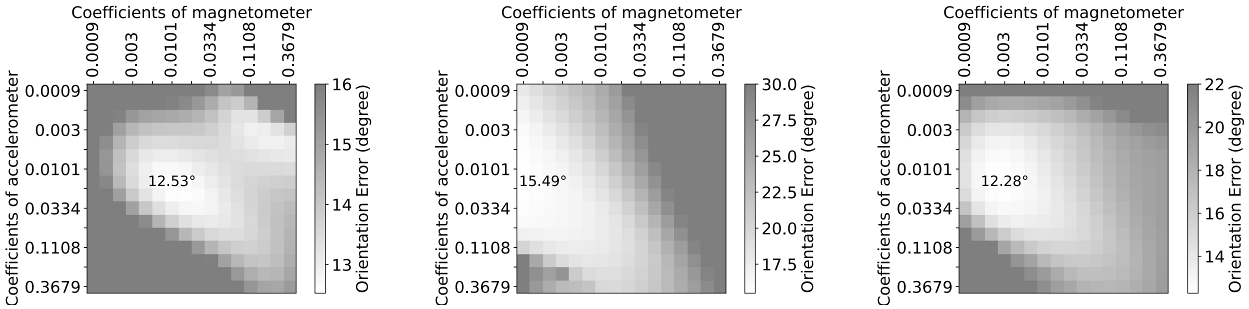
We also conduct real experiments to investigate how different settings of α and β affect the performance of CF for orientation estimation. Figure 2 presents our experimental results. We set α and β to e^x, where $-7 \leqslant x \leqslant -1$ with a step length of 0.4. Figure 2(a) shows the mean orientation error of CF on 20-minute data collected in a non-distorted magnetic field. We can see that when the magnetic field is stable, the orientation estimation performs better with higher β. On the other hand, Figure 2(b) shows the mean orientation error of a 20-minute data collected by the same user, but in a distorted magnetic field. In this case, CF should not rely too much on the magnetometer, and lower value of β is favored. Figure 2(c) presents the hybrid result of 14 data traces collected from these two environments by different users. As shown in these figures, $12.53°$, $15.49°$, and $12.28°$ are the lowest orientation error can achieve by setting appropriate parameters. These results suggest that the optimal settings of the α and β are different for different data traces, depending on the magnetic field and device motions.

3) Importance of Adaptively Choosing α and β: Statically choosing the optimal values of α and β may not suffice to achieve the highest orientation accuracy, as the quality of the accelerometer and magnetometer may fluctuate significantly over time. To validate this argument, we conduct real experiments using a short data clip spanning three seconds. We use CF to process this clip using different values of α, while setting β as its best static value we found from Figure 2. Figure 3(a) illustrates how the orientation error changes over time. It is evident that the curves take turns to reach the lowest error among themselves within only three seconds. This observation suggests that the optimal value of α frequently fluctuates and requires to be adaptively determined. Similarly, we run CF with different values of β while setting α statically. We also observe fluctuations as depicted in Figure 3(b). Therefore, the optimal value of β also needs to be adaptively calculated. Moreover, these two parameters jointly impact the orientation results and their values should be adaptively adjusted in conjunction with each other.

B. Employing Deep Learning to Determine α and β

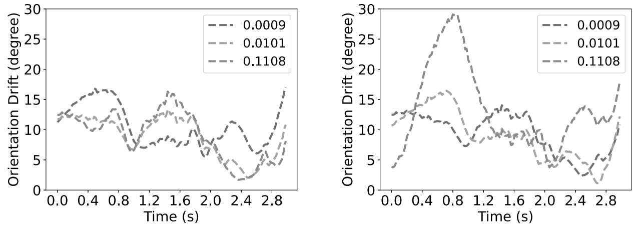
Based on the observations from Figure 3, it is challenging to adaptively and accurately adjust the values of α and β jointly. To address this challenge, we charge the deep learning with this task, leveraging its capability for automatic parameter tuning. Additionally, via changing the output target of the deep learning module from a 3×3 orientation matrix to two scalars, we simplify the output complexity of the module. Since the optimal values of α and β are closely tied to the quality of their respective sensors, the strategy to calculate them can be ubiquitously applied, unlike the scheme which directly calculates the orientation matrix, which is susceptible across different scenarios (Section II-B). Therefore, by employing deep learning to pilot the CF via dynamically providing α and β, we can enhance the potential modeling capacity of the deep learning module. As a bonus, since the task is more ubiquitous now, it may also improve the generalization ability of the deep learning module.

To deploy a deep learning module to calculate the optimal values for α and β, ground-truth values of α and β at every time step are needed for training. However, obtaining these ground-truth values is inherently difficult. As per Equation (1), the orientation result at each time step serves as basis for future



(a) Stable magnetic field (b) Distorted magnetic field (c) Hybrid magnetic field

Fig. 2. Orientation error with different value of gravity and magnet coefficients.



(a) Orientation error with different values of α, while β is 0.00136. (b) Orientation error with different values of β, while α is 0.01005.

Fig. 3. Orientation error with different value of α and β over time.

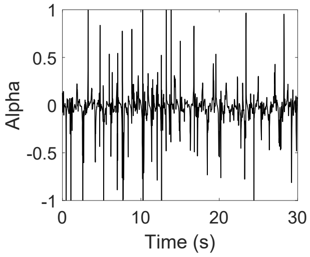


Fig. 4. The value of α_{pt}.

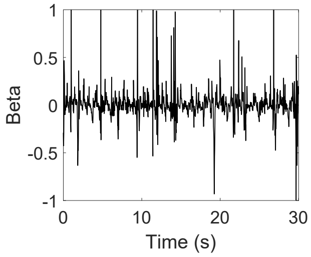


Fig. 5. The value of β_{pt}.

estimations. Conversely, the orientation result at every time step is affected by the values of α and β at all previous time steps. Therefore, it is almost impossible to globally optimize α and β at all time steps, especially considering that there are usually 50$\sim$60 time steps in one second (depending on the device's sampling rate). Consequently, the optimization task becomes exponentially large and challenging.

1) Attempt of Replacing Ground Truth: To make up for the absence of the ground truth of α and β, we try to calculate pseudo-truths for α and β, denoted as α_{pt} and β_{pt}, which are calculated via locally minimizing the orientation error at each time step separately. Therefore, given specific $\Theta^{t-\Delta t}$, R^t_{gyro}, R^t_{acc}, and R^t_{mag} in Equation (1), and additionally with the orientation ground truth Θ^t_0, the pseudo-truth α_{pt} and β_{pt} stand for the values that minimize the difference between Θ^t and Θ^t_0, denoted as $\angle(\Theta^t, \Theta^t_0)$:

$$< \alpha_{pt}, \beta_{pt} >= \underset{\alpha,\beta}{\arg\min} \angle(\Theta^t, \Theta^t_0) \qquad (2)$$

We conduct real experiments to assess the feasibility of pseudo-truth values. When we apply α_{pt} and β_{pt} to the CF, we observed a significant reduction in orientation error to less than 3°. This is because α_{pt} and β_{pt} can eliminate errors in accelerometer and magnetometer calibrations, leaving the gyroscope as the primary error source. It's important to note that this low error can only be achieved offline, as calculating α_{pt} and β_{pt} requires the orientation ground truth. Therefore, we need the deep learning module to calculate α_{pt} and β_{pt} online. We set the offline calculated α_{pt} and β_{pt} as target outputs, and IMU sensor readings as inputs to train a RNN model. Unfortunately, our results indicate that the model struggle to learn the calculation of the pseudo-truth. Since the offline calculated α_{pt} and β_{pt} can be negative sometimes, we calculate their average absolute values, $|\alpha_{pt}|$ and $|\beta_{pt}|$, which are 0.1037 and 0.0667 respectively. However, the values inferred by the RNN, denoted as $\hat{\alpha_{pt}}$ and $\hat{\beta_{pt}}$, their average absolute values, $|\hat{\alpha_{pt}}|$ and $|\hat{\beta_{pt}}|$, are 0.0256 and 0.0163. Moreover, their average absolute differences from α_{pt} and β_{pt} are 0.1041 and 0.0661, implying that the RNN learns this calculation terribly. To further explain these results, we zoom in α_{pt} and β_{pt}, as demonstrated in Figure 4 and Figure 5. They show the values of α_{pt} and β_{pt} within 30 seconds. We observed severe fluctuations in their values, making it challenging for a neural network to learn effectively.

2) Using Orientation Error to Guide Training: As the attempt to utilize pseudo-truth fails to compensate for the absence of ground truth, we propose to leverage orientation error information to guide the training of α and β. Given that minimizing orientation error is our primary objective, we seek to incorporate the error information into the training process. However, since orientation is not the output of the deep learning module, gradient descent and backpropagation can no longer transmit the error back to optimize the model. To that end, we convert the supervised learning into a DRL module, which uses negative orientation error as reward to pilot CF by outputting α and β via its actions.

DRL is a promising machine learning approach that combines deep learning with reinforcement learning [18]. A DRL agent learns to perform a task through trial and error by interacting with the environments. The agent receives rewards or penalties based on its actions and learns to maximize its cumulative reward over time. DRL is particularly useful in environments where traditional methods such as rule-based systems or supervised learning are difficult or impossible to apply. Examples include games [19], robotics [20], and

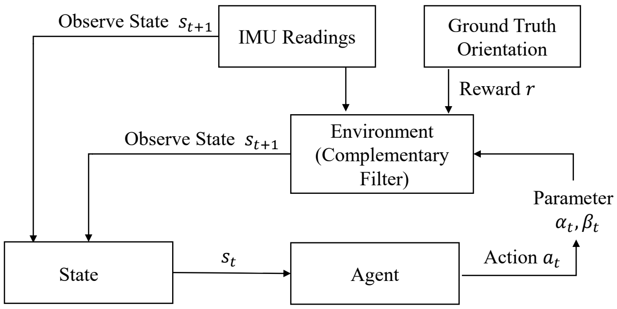


Fig. 6. System Architecture of *DRLPilot*

autonomous driving [21], where the agent must learn to adapt to the changing conditions.

IV. *DRLPilot* DESIGN

In this section, we introduce the design of *DRLPilot*, including an overview, state, action, reward, and training policy.

A. DRLPilot Overview

Figure 6 illustrates the architecture of *DRLPilot*. At each time step $t+1$, what the agent of *DRLPilot* can observe from the environment includes IMU readings and the output of CF at the previous time step (the estimated orientation Θ^t). We design the state of *DRLPilot* based on these observations. The responsibility of the agent is to adjust the parameter settings of α and β at time step $t+1$ in CF.

DRLPilot extracts system dynamic features from IMU readings, which are intricately related to the setting of α and β [22]. Section IV-B provides detailed insights into these dynamics. Among these features, some remain constant scalar values across different reference frames, while others are vectors obtained from the device's local reference. Utilizing these extracted features as states, the agent makes decisions and outputs α and β for CF to estimate the orientation at time $t+1$. IMU readings are also the inputs to CF. We leverage the ground-truth orientation to guide the training process, but it is no longer needed once the agent is well-trained.

B. State Representation in DRLPilot

A state is a feature vector that represents the current environment observed by an agent at any given time. It provides the necessary information for the agent to make decisions on what actions to take. Table II demonstrates the features of our state representation. Since *DRLPilot*'s agent is responsible for tuning parameters for the sensors, the features within a state should adequately capture the reliability of sensor readings. Higher reliability should correspond to larger weights for the associated sensor, and vice versa. The state features of *DRLPilot* includes two parts: those not controlled by the agent (1-9 in Table II) and those controlled by the agent (10-13 in Table II).

1) Features not Controlled by the Agent: These features consist of readings from the three IMU sensors and information extracted from the sensor readings, including accelerometer and magnetometer direction errors, the angle between the accelerometer and magnetometer, and the magnitude of the three IMU sensors. These features are affected by the movements of the devices and the magnetic field in the surrounding environment, but are not affected by the actions of the DRL agent.

Feature 1-3 in Table II are straightforward. They are the readings we get from the three sensors.

Feature 4&5: Accelerometer and magnetometer direction error. These two kinds of errors are paramount in determining sensor quality. To accurately calculate the two errors, we need to transform accelerometer and magnetometer directions into the global reference, and compare them with their expected directions: gravity and magnetic north. However, accurately calculating these errors requires ground-truth orientation. To that end, we propose using the gyroscope to measure the direction errors of the accelerometer and magnetometer since the gyroscope can be trusted in a short term [5]. The calculation process is shown below:

$$\hat{R} = \prod_{t=t_1}^{t_2} R^t_{gyro} \tag{3}$$

$$\hat{\overrightarrow{acc}} = \overrightarrow{acc}(t_1) \cdot \hat{R} \tag{4}$$

$$\hat{\overrightarrow{mag}} = \overrightarrow{mag}(t_1) \cdot \hat{R} \tag{5}$$

$$\hat{err}_{acc} = \angle(\hat{\overrightarrow{acc}} \cdot \overrightarrow{acc}(t_2)) \tag{6}$$

$$\hat{err}_{mag} = \angle(\hat{\overrightarrow{mag}} \cdot \overrightarrow{mag}(t_2)) \tag{7}$$

Within the local reference, for a given time step t_2, we aggregate the historical gyroscope readings R^t_{gyro} within 160 ms (determine via real experiments) to derive a gyroscope-based rotation $\hat{R}$ from time $t_1 = t_2 - 160ms$ to the current time t_2. Subsequently, based on this rotation, we rotate the directions of the accelerometer reading $\overrightarrow{acc}(t_1)$ and magnetometer reading $\overrightarrow{mag}(t_1)$ at t_1, and acquire two anticipated directions of accelerometer and magnetometer at t_2, denoted by $\hat{\overrightarrow{acc}}$ and $\hat{\overrightarrow{mag}}$. We then calculate the angle between the actual direction of accelerometer at t_2 and the anticipated direction, denoted by $\hat{err}_{acc}$. We use that angular difference as an anticipation for accelerometer direction error at t_2. The same principle is applied in calculating the magnetometer direction errors $\hat{err}_{mag}$. The whole calculation process is conducted within local reference and does not require ground-truth orientation. We evaluate the errors we derived from this process with those calculated using ground-truth orientation in Section VI-C.

Feature 6: Angle between magnetometer and accelerometer. This can be easily calculated within the local reference via simply calculating the angle between the raw readings of the accelerometer and magnetometer. This angle can provide insights into the variation of data quality, particularly by comparing it with historical values. For instance, a significant change in the angle between the magnetometer and accelerometer could indicate two potential scenarios. First, it may suggest a high degree of linear acceleration, causing a deviation of the accelerometer direction from the gravity direction. Second, it might imply severe magnetic distortions, resulting in the magnetometer pointing to various directions

TABLE II
FEATURES IN *DRLPilot*'S STATE

Not controlled by agent
1. Accelerometer direction in local reference
2. magnetometer direction in local reference
3. Angular velocity in local reference
4. Accelerometer direction error
5. Magnetometer direction error
6. Angle between accelerometer & magnetometer
7. Accelerometer magnitude
8. Angular velocity magnitude
9. Magnetometer magnitude
Controlled by agent
10. Orientation estimated by CF
11. Accelerometer direction in global reference
12. Magnetometer direction in global reference
13. Angular velocity in global reference

at different time steps. In either scenario, the data quality is impacted, and thus the weights should be adjusted.

Feature 7-9: Magnitude of the three IMU sensors. The magnitude of the three sensors can be easily calculated from the raw data. A^3 [5] and MUSE [13] propose trusting accelerometer when the system is static. To detect static moments, they observe whether the measurements of accelerometer are around $9.8m/s^2$. When accelerometer measures $9.8m/s^2$, the system is most probably static, and the accelerometer quality is guaranteed, thus the weight for accelerometer should be high. However, when the accelerometer magnitude deviates significantly from $9.8m/s^2$, indicating the system is moving violently. In such cases, the gravity will be polluted by a large proportion of linear acceleration, and according to A^3 [5], the gyroscope may also become untrustworthy, hence the weights should be decreased.

The magnitude of angular velocity indicates how fast the system rotates. According to A^3 [5], gyroscope errors increase with rising angular velocity, leading to faster error accumulation in gyroscope integration. Therefore, it might be beneficial to increase the weights of the accelerometer and magnetometer to intensify calibrations.

The magnetometer is different from these two sensors. Deviations in the magnitude of the magnetometer could reflect the intensity of magnetic distortions [23], [24]. The magnitude of the Earth's magnetic field is typically around $50\mu T$, but it changes when distorted by metal. In the presence of magnetic distortions, the directions of the magnetic field change at different locations, causing errors in the directions measured by the magnetometer. As a result, calibration via magnetometer could be wrong and pollutes orientation estimation. Therefore, it would be wise to decrease the weight of magnetometer to avoid further impact on orientation estimation.

2) Features Controlled by the Agent: These features include the orientations calculated by CF, accelerometer and magnetometer directions, and angular velocity in global reference.

Feature 10: Orientation estimated by CF. The orientation matrix at each time step is the target that the agent controls via tuning the weights for the accelerometer and magnetometer.

Feature 11-13: Accelerometer direction, magnetometer direction, and angular velocity in global reference. The features introduced before contain 24 dimensions in total. However, a DRL agent may not learn well if the agent has limited influence on the state. We observe that, among the 24-dimension designs above, only the 9 dimensions of the orientation matrix are controlled by the agent: The agent controls the weights for the accelerometer and magnetometer, and the weights controls the calibration processes, ultimately affecting the orientation. In the pursuit of expanding the dimensions that the agent can control, we transform the readings in the local reference into the global reference. Since the sensor readings are measured in the local reference, we apply the estimated orientation matrix to transform the accelerometer direction in the local reference to the global reference. Intuitively, this direction should point 'down' in the global reference. Any deviation from this alignment suggests potential sensor inaccuracies. We use the same method to transform the magnetometer direction and angular velocity we measured from the local reference to global reference.

C. Action and Reward in DRLPilot

1) Action in DRLPilot: Based on the outlined state above, *DRLPilot* aims to optimize the parameter settings for CF to accurately estimate the orientation of a device in varying environments with varying levels of sensor noise over time. The primary goal is to find the best parameter settings that enable CF adapt to changes in sensor noise, thereby providing reliable orientation estimates over time. The action space consists of two continuous values that control the weight of the accelerometer and magnetometer coefficients, respectively.

2) Reward in DRLPilot: The primary objective of orientation estimation is to minimize the overall orientation error in 3D space. The reward is defined to minimize the error. Our dataset contains the ground-truth orientation at each time step, which is used to calculate the reward. Orientation error refers to the minimum degree of rotation required to align the estimated orientation with the ground-truth orientation. Smaller errors result in a higher reward.

We use 3D rotation matrix to represent the orientation calculated by *DRLPilot*, denoted as R. Suppose the ground-truth orientation is R_G. Since a rotation matrix is orthogonal, its inverse is its transpose. Then the rotation difference between two rotation matrices is:

$$\Delta R = R \cdot R_G^{-1} = R_G \cdot R^T \tag{8}$$

where ΔR can be transformed to rotation angle in degrees, denoted as err_{ori}. We define our reward function as follows:

$$Reward = 180^\circ - err_{ori} \tag{9}$$

3) Refined Reward Design: We further improve the reward design of *DRLPilot* based on the original one, so that it learns more accurately. Since orientation estimation is an iterative process, the orientation result at every time step will also serve as the basis to infer future results. Therefore, the choices of

α and β will not only affect the current orientation result, but also all results within a short period. To fully reflect the quality of the current choice of α and β, we design a simple but effective inertia reward that aggregates the orientation errors with diminishing weights.

$$Reward(t) = Reward(t-\Delta t)*k_I+(180^{\circ}-err_{ori})*(1-k_I) \quad (10)$$

As Equation (10) shows, the inertia reward is calculated based on the reward of the last time step and the current orientation error. Specifically, it is a linear combination of both, as the inertial coefficient k_I between 0 and 1, which controls the ratio of the combination. If $k_I = 0$, it means the inertia is zero and the reward degrades to the ordinary design that only reflects the quality of the choice of α and β at current time step. If $k_I = 1$, it means the inertia is infinite, and the reward never updates. Therefore it is crucial to set the best value for k_I. To that end, we conduct real experiments on the setting of k_I and find $k_I = 0.2$ achieves the best performance.

D. DRLPilot Training

Policy gradient algorithms are a class of RL methods that aim to optimize the policy parameters of an agent to maximize its expected cumulative reward. These algorithms rely on computing an estimate of the policy gradient for the expected cumulative reward, which is then used to update the policy parameters using a stochastic gradient ascent algorithm. Policy gradient algorithms can handle continuous action spaces.

$$\Theta \leftarrow \Theta + \alpha \bigtriangledown_\theta \mathbb{E}_{\pi_\Theta}[r] \quad (11)$$

Various policy gradient algorithms can be used to train the agent. In this study, we utilize proximal policy optimization (PPO) [25]. PPO has gained popularity due to its utilization of the clipped surrogate objective function (Equation (12)) to improve its stability and robustness to hyper-parameters.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \quad (12)$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \quad (13)$$

where $\hat{\mathbb{E}}_t$ is the empirical average over a batch of trajectories, $\hat{A}_t$ is an estimator of the advantage function, and $r_t(\theta)$ is the probability ratio between the new policy and the old policy. The hyperparameter ϵ controls the extent to which the policy update is clipped, and its value can be tuned to balance the trade-off between stability and speed of convergence. By using the clipped surrogate objective function, PPO is able to improve the stability and convergence of the policy updates, making it a promising choice for training RL agents in a variety of environments [26], [27].

V. Implementation

Data Augmentation. Suppose users collect the training data while facing north and then uses a model trained on that data to infer actions performed while facing west. Unless there is no directional difference in the user's motion (e.g. only rotating in the north/west direction), the inference will be affected. To magnify this problem, we desire data that has different directionalities. However, humans cannot control their motions to ensure only directional differences in collected data. Therefore, we augment existing data by simulating various directions, where virtual users replicate identical motions along these simulated paths.

To achieve that, we modify the raw data to simulate different directionality. Apparently, from the perspective of a virtual user, the motions remain unchanged. The only difference is the direction of magnetic north. Consequently, it appears as the north is rotating relative to the user, meaning we only need to rotate the magnetometer readings to augment the data.

To implement this, we first transform the magnetometer readings into the global reference using ground-truth orientations. Then in the global reference, we rotate the magnetometer directions along the vertical axis with different angles, such as 30°. We then transform the data back to the local reference using the ground-truth orientations. This process involves setting 30° as the step length and rotating the data through 360°, thus generating 12x data with different directionality.

***DRLPilot* Implementation.** *DRLPilot* is implemented using the CleanRL framework, which is a flexible and extensible reinforcement learning framework, allowing us to easily experiment with different algorithms, environments, and configurations. The agent in *DRLPilot* is based on the actor-critic architecture and utilizes two LSTM layers, each with 128 neurons, the output of actor network is a fully-connected layer with 2 neurons (action space is 2), the output of critic network is one fully-connect layer with 1 neuron. To train *DRLPilot*, a computer that is equipped with an 13th Gen Intel Core i9-13900HX CPU and a NVIDIA GeForce RTX 4090 GPU is used. It runs on the Ubuntu 22.04 operating system.

Hyper-parameter Tuning. The performance of a DRL agent can be significantly affected by the choice of hyperparameter values. However, there is no straightforward method to determine the optimal hyperparameter combination that would result in the best possible total reward. We set the hyperparameters through a set of experiments. The optimizer used for gradient-based optimization is Adam with a learning rate of 7e-5. The discount factor used to weight future rewards is set as 0.99. The lambda parameter for GAE (Generalized Advantage Estimation) is 0.95, which is used to balance bias and variance when estimating the advantage function. The clipping parameter used to bound the change in the policy distribution during each update is 0.2.

VI. Evaluation

In this section, we test the performance of *DRLPilot*, including overall performance, performance decomposition analysis, and system overhead.

A. Experimental Settings

Dataset. We use the dataset introduced in [15] to do experiments. This dataset contains about 1,500,000 data samples. To ensure a comprehensive evaluation of *DRLPilot*, we collect new data at three additional testing places using the same data

TABLE III
ORIENTATION ERROR ESTIMATED FROM DIFFERENT METHODS ON ORIGINAL DATASET.

Methods	$RTAT$	CF_{so}	$RULE$	$DRLPilot$
Hybrid data	16.55°	16.76°	15.80°	**11.54**°
Room data	11.67°	13.14°	12.28°	**10.42**°
Hallway data	21.31°	20.29°	19.30°	**16.23**°

TABLE IV
ORIENTATION ERROR ESTIMATED FROM DIFFERENT METHODS ON ROTATION DATASET.

Methods	$RTAT$	CF_{so}	$RULE$	$DRLPilot$
Hybrid data	58.45°	17.58°	16.53°	**14.41**°
Room data	35.42°	14.98°	14.18°	**12.74**°
Hallway data	80.87°	19.73°	18.78°	**17.78**°

collection method. The new data we collect includes 215,040 data samples. By using a larger and more diverse dataset, we can better assess the performance of *DRLPilot* under a broader range of conditions.

Evaluation Metrics. We evaluate the orientation estimation of *DRLPilot* and all the baselines via 3D orientation error. It is the minimal degree of rotation needed to align the estimated orientation to the ground truth orientation.

Baselines for Orientation Estimation. We compare the orientation estimation error of *DRLPilot* with three baselines.

- $RTAT$ [14]: The latest data-driven method to estimate device orientation based on BiLSTM-based neural network. The neural network takes the data from all the three IMU channels as input and outputs the orientation of a device. The original network design in this paper is simple. For fairness, we add an additional 128-unit BiLSTM layer to the original network, resulting in more neurons and a similar model size as *DRLPilot*.
- CF_{so}: Like MUSE [13], we use a complementary filter with static setting of parameters. We use both north and gravity as anchors to calibrate the orientation estimation by gyroscope integration. We use the ground truth data from the hybrid training dataset to determine the static optimal values of α and β (i.e., 0.015 and 0.003 respectively), as introduced in Section III.
- $RULE$: A custom rule-based weight-tuning complementary filter. It combines the strategies of tuning the weights of the accelerometer and magnetometer from two methods respectively. For α, $RULE$ adopts the strategy of MUSE [13], which uses the accelerometer only when it roughly measures the magnitude of gravity ($9.8 \pm 1.3 m/s^2$, where 1.3 is determined via real experiments on MUSE), thus $\alpha = 1$, otherwise $\alpha = 0$. For the magnetometer, $RULE$ adopts the strategy from another work [24]. It mainly uses the measured magnetic field magnitude to detect magnetic distortion. As β floats between 0 and 1, when the magnetic distortion is severe, it could contain large errors, β is set to be lower. $RULE$ combines both rules to determine α and β. By comparing $RULE$ and *DRLPilot*, we demonstrate the advantage of using DRL to pilot CF-based orientation estimation.

B. DRLPilot Performance

We compare the performance of *DRLPilot* to the baseline methods. Our evaluation considers the overall performance, the generalization ability and the performance over time.

1) Overall Performance: We show the overall performance of all methods on both original dataset and our augmented rotation dataset. The original dataset refers to the data provided by [15], we combine the data from the two places (room and hallway) to train a general model. The rotation dataset refers to the rotated data of the original dataset by 360°, with a step 30°, as introduced in Section V.

Table III depicts the performance of different methods on the original dataset. The hybrid data combines room data and hallway data. We train $RTAT$ and *DRLPilot* using hybrid training data and test the models using hybrid data, room data, and hallway data, respectively. The test data has never been seen by the models during the training phase. We apply the optimal parameter settings for CF_{so} found offline to the same test data as $RTAT$ and *DRLPilot*. Table III shows the testing result. The average orientation error of $RTAT$, CF_{so}, $RULE$, and *DRLPilot* on hybrid data is 16.55°, 16.76°, 15.80°, and 11.54°, respectively. *DRLPilot* reduces the orientation estimation error by 30.3%, 31.1%, and 27% compared to $RTAT$, CF_{so} and $RULE$, respectively. It also achieves the lowest error on room data and hallway data.

To further test, Table IV depicts the overall orientation error training and testing on the augmented rotation dataset. The rotation dataset is 12x compared to the original one since we have rotated the data to 12 different directions. We train $RTAT$ and *DRLPilot* using the hybrid rotation data, and test the models using hybrid rotation data, room rotation data, and hallway rotation data, respectively. We still use the same parameters we found above for CF_{so} because rotation does not influence the performance of CF too much. The average orientation error on hybrid data of $RTAT$, CF_{so}, $RULE$, and *DRLPilot* is 58.45°, 17.58°, 16.53° and 14.41°, respectively. *DRLPilot* reduces the orientation error by 75.3%, 18.0%, and 12.83% compared to $RTAT$, CF_{so}, and $RULE$ respectively. It still achieves the lowest error on room data and hallway data. The results indicate *DRLPilot* has better performance than baselines. The significant increase in errors of $RTAT$ is attributed to the lightweight design of $RTAT$, which underfits the augmented dataset.

Compared to CF_{so}, *DRLPilot* dynamically adjusts the parameters, which is more adaptive than the fixed optimal setting. Compared to $RTAT$, the main reason for the superior performance of *DRLPilot* is that the outputs of its neural network are scalars, which are irrelevant to facing directions. On the other hand, the output of $RTAT$ are 3D orientation vectors affected by the facing direction, making the learning task more complex. Compared to $RULE$, *DRLPilot* uses a neural network to make finer parameter adjustments, since it is trained using orientation error as a reward.

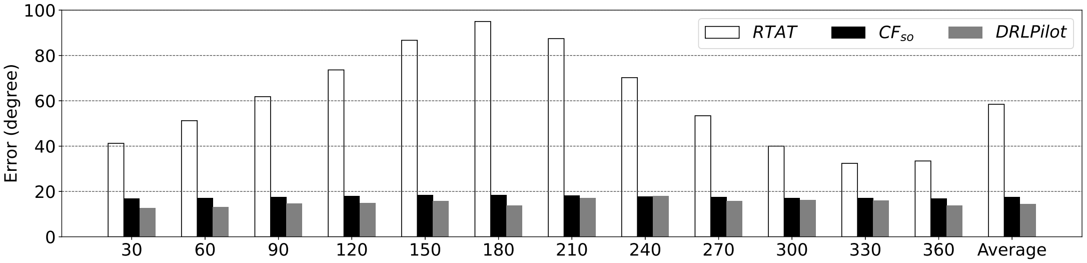


Fig. 7. Orientation error on rotation dataset (hybrid data)

CDF on Rotation Dataset. Figure 8 illustrates the CDF of *DRLPilot*'s orientation error in comparison to $RTAT$ and CF_{so} when tested on the hybrid test data. As depicted in Figure 8, the 80th percentile orientation error of *DRLPilot*, $RTAT$, and CF_{so} measures approximately 18°, 25°, and 100° respectively. The 60th percentile orientation error of the three systems is approximately 13°, 18°, and 50° respectively. Consistently, *DRLPilot* outperforms the baseline methods.

2) Performance of different rotations: As we introduced before, the rotation dataset is created by rotating the original dataset by 360 °, with a step of 30 °. Figure 7 depicts the overall performance of all methods on rotation dataset. They are trained by combining all directions of the rotated data. We show the orientation error on each rotation angle and the average orientation error on all rotation angles.

As shown in Figure 7, the estimation error of $RTAT$ on all rotation angles fluctuates from 32° to 95°. While, the error of CF_{so} and *DRLPilot* on all rotation angles is stable. CF_{so} and *DRLPilot* are based on Complementary Filter to output orientation. They exhibit relatively consistent orientation error across different rotation angles, with only small variations. The observed variations in orientation error across different rotation angles may be attributed to the fact that the rotational motion decomposition along the North is different for the same physical motions. Specifically, when the decomposition is around the North, the calibration from magnetometer readings may fail, leading to higher orientation error.

3) Performance Along with Time: Figure 9 illustrates the orientation error of *DRLPilot* and the baseline methods over time for a 100-second data trace. The figure clearly shows that *DRLPilot* consistently outperforms $RTAT$ and CF_{so}, while also demonstrating greater stability over time. To further quantify it stability, we compute the standard deviation of the orientation error for the three methods. They are 9.30 for *DRLPilot*, 11.42 for CF_{so}, and 12.92 for $RTAT$. This suggests that *DRLPilot* has small variations on orientation estimation compared to the baseline methods.

4) Model generalization: We conduct further evaluation to assess the generalization of *DRLPilot* and the baseline methods to the new places that were not seen in the training dataset. We collect some new data in three places, P1, P2, and P3. P1 is an outdoor residential area, P2 is a large classroom in a University, and P3 is an indoor living area. Figure 10 presents the orientation error of *DRLPilot* and the baseline methods in the three new places. In each of them, *DRLPilot* consistently outperforms $RTAT$ and CF_{so}. It is worth noting that $RTAT$ shows degraded performance in new places, mainly because of the different magnetic fields present in those places. As a supervised learning method, $RTAT$ has limited generalization ability to new places, and exhibits large variations in performance in such scenarios.

C. Performance Decomposition of DRLPilot

We test the performance improvements from different components within *DRLPilot*, encompassing different state, action, and reward designs. Each unique design necessitates an additional training cycle. Given that training a model on the augmented rotation dataset typically consumes approximately 20 hours, we opt to conduct experiments using the considerably smaller original dataset to explore and determine the design for each component.

State Design. Table V presents the performance of different state designs. “Local data only” represents the design using nine-dimensional IMU readings as a state at each time step, including three-dimensional accelerometer readings, three-dimensional gyroscope readings, and three-dimensional magnetometer readings. The orientation error for this state design is 18.98°, which is 39.2% higher than *DRLPilot*. "Without global data" employs feature 1-10 from Table II but excludes the feature 11-13. The orientation error is 15.31° for this state design. It achieves 24.62% higher error than *DRLPilot*. "Without gravity and magnet errors" includes everything but feature 4 and 5 in Table II. The resulting orientation error is 14.04°, which is 21.67% higher than *DRLPilot*. This result reveals the importance of gravity and magnet error we extracted from gyroscope readings. In order to evaluate the accuracy of our extracted gravity and magnet error from gyroscope readings, we conduct experiments where we replace our extracted values with the ground truth values in the state, which is "Ground-truth gravity and magnet errors" in Table V. This state design resulted in an orientation error of 11.48°, demonstrating that our extracted gravity and magnet error values do not degrade the agent's performance. These experimental results indicate the features we extract to constitute a state are important to guide the agent make better decisions.

Action Design. In our design, the agent of *DRLPilot* performs actions in a continuous range within [0, 0.2], the average orientation error 11.54° from our experiments. To investigate the effectiveness of action with value 0, we conduct experiments by setting the action range to [0.001, 0.2]. The results show that the average error in this case is 13.31°, which

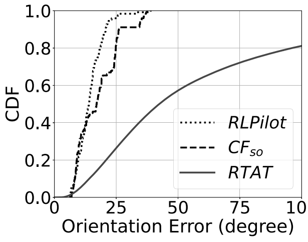


Fig. 8. Original dataset.

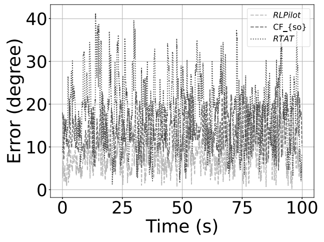


Fig. 9. Orientation error over time.

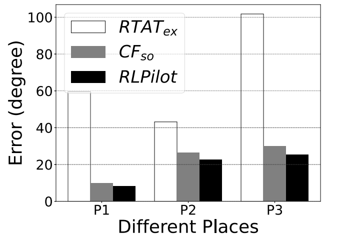


Fig. 10. Different places.

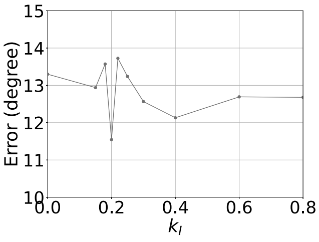


Fig. 11. Different setting of k_I

TABLE V
ORIENTATION ERROR OF DIFFERENT STATE DESIGNS.

States	Error
Local data only	18.98°
Without global data	15.31°
Without gravity and magnet errors	14.04°
Ground-truth gravity and magnet errors	11.48°
DRLPilot	**11.54°**

TABLE VI
ORIENTATION ERROR OF DIFFERENT ACTION DESIGNS.

Actions	Error
$\alpha, \beta \in$ [0.001, 0.2]	13.31°
$\alpha, \beta \in$ [0, 0.25]	13.25°
α, $\beta \in$ **[0, 0.2]**	**11.54°**
$\alpha \in$ [0, 0.2], β = 0.003	17.16°
α = 0.015, $\beta \in$ [0, 0.2]	11.74°

is 13.3% higher than the setting of [0, 0.2]. This indicates that the action with value 0 is important for the system, as it allows the agent to take actions that do not rely on accelerometers and magnetometers when their senor noise is severe. To determine the upper bound of the action value, we conduct experiments by setting the action range to [0, 0.25], the average error under this configuration is 13.25°, marking 12.9% increase compared to the [0, 0.2] setting. Following these experiments, we set the action range to [0, 0.2].

Disable One of the Two Actions. To further validate the necessity of both actions, we conduct experiments by assigning one action to the optimal value as CF_{so}, allowing the agent to control the other one. As presented in Table VI, the experimental results indicate that when the agent exclusively controls the accelerometer coefficient α while the magnetometer coefficient β is set to a static value of 0.003, the average orientation error is 17.16°. This value was 32.75% higher than when both parameters were jointly controlled. Similarly, when we set the accelerometer coefficient α to the default value of 0.015 and enable the agent to control only the magnetometer coefficient β, the average orientation error is 11.74°, marginally higher than when both parameters were jointly controlled.

Reward Design. In demonstrating the efficacy of our enhanced reward design, Figure 11 illustrates the orientation error under various settings of k_I. Specifically, when $k_I = 0$, it signifies the original reward design, resulting in an orientation error of 13.29°. From the same figure, it's observed that at $k_I = 0.2$, the system achieves optimal performance, leading to an orientation error of 11.54°. This refined reward design showcases a notable reduction in error by 13.17% compared to the original design.

D. System Overhead

The *DRLPilot* model we develop is lightweight. It is only 1.7 MB in size. It takes *DRLPilot* about 0.15 ms to perform an action. In comparison, the RTAT model, with an identical size, achieves an average inference time of 0.003 ms. The results are tested on a system powered by a 13th Gen Intel Core i9-13900HX CPU and a NVIDIA GeForce RTX 4090 GPU, operating within the PyCharm IDE using the PyTorch framework on Ubuntu 22.04 operating system.

The latency of *DRLPilot* is higher, approximately 50 times that of RTAT. Considering the inference latency of RTAT on mobile devices like Samsung S9 and Google Pixel 3, which stand at 2.98 ms and 1.82 ms for processing 1-second sampling data, respectively, the estimated latency for *DRLPilot* on these devices would be approximately 149 ms and 91 ms for processing 1-second sampling data. Taking into account that these two devices were released in 2018 and the computing capabilities of newly published devices are substantially stronger, Hence, it can be inferred that *DRLPilot* is capable of achieving real-time performance on modern mobile devices.

VII. RELATED WORK

Traditional Methods for Orientation Estimation. Various sensor fusion methods have been proposed to estimate device orientation in the literature, such as the Kalman filter [10], extended Kalman filter [11], and complementary filter [12]. The Kalman-filter-based methods relies on a mathematical model of the system being estimated. If the model is incorrect or inaccurate, the filter's estimates can be unreliable [5], [13]. Complementary Filter provides a simple and low-cost way for orientation estimation. It does not require a mathematical model, but it requires to set its parameters carefully for accurate sensor calibration. A^3 [5] proposes gyroscope integration with opportunistic replacements for device orientation estimation, utilizing both gravity and magnetic north for calibration. MUSE [13] relies only on magnetic north via a CF, limiting its orientation calibration to 2 Degrees of Freedom (DoF).

Data-Driven Methods for Orientation Estimation. Recent studies have explored the use of data-driven methods for processing IMU measurements and estimating orientation [6], [28], [29], [30], [7], [8], [9], [14], [31]. Two representative solutions for orientation estimation are IDOL [9] and RTAT [14].

However, both methods rely on supervised learning, which requires a large amount of labeled data. Additionally, supervised learning may suffer from performance degradation when sensing characteristics (e.g., gravity error or magnetic field) change. Self-supervised learning has recently been used to extract contextual features from unlabeled IMU data for different application tasks, aiming at building "foundation" models, e.g., TPN [32] and LIMU-BERT [30]; however, they still require labeled data for new scenarios.

Deep Reinforcement Learning. DRL has been used in a wide range of applications, such as navigation [33], smart buildings [34], [35], and irrigation control [36]. In this paper, we integrate CF into a DRL framework for IMU sensor data processing. We design the DRL to guide the agent in learning adaptive parameters for the CF. Our solution provides better generalization compared to supervised learning and better adaptability compared to traditional complementary filter-based orientation estimation. It is orthogonal to the sensor data denoising methods [37].

VIII. Conclusion

This paper proposes a novel orientation estimation approach combining deep reinforcement learning and complementary filter. We develop *DRLPilot*, a DRL framework that incorporates complementary filter for orientation estimation. With our customized data of the DRL states and reward function, *DRLPilot* can adjust the parameters of the complementary filter based on the data quality of three IMU sensors. Extensive experiments demonstrate that *DRLPilot* exhibits better modeling capability than supervised learning methods and superior adaptability compared to the classical complementary filter-based orientation estimation method.

References

[1] Manikanta Kotaru and Sachin Katti. Position tracking for virtual reality using commodity wifi. In *IEEE CVPR*, 2017.

[2] Miaomiao Liu, Xianzhong Ding, and Wan Du. Continuous, real-time object detection on mobile devices without offloading. In *IEEE ICDCS*, 2020.

[3] Swadhin Pradhan, Ghufran Baig, Wenguang Mao, Lili Qiu, Guohai Chen, and Bo Yang. Smartphone-based acoustic indoor space mapping. *ACM IMWUT*, 2018.

[4] Dushyant Mehta, Srinath Sridhar, Oleksandr Sotnychenko, Helge Rhodin, Shafiei, et al. Vnect: Real-time 3d human pose estimation with a single rgb camera. *ACM Transactions on Graphics*, 2017.

[5] Pengfei Zhou, Mo Li, and Guobin Shen. Use it free: Instantly knowing your phone attitude. In *ACM MobiCom*, 2014.

[6] Changhao Chen, Xiaoxuan Lu, Andrew Markham, and Niki Trigoni. Ionet: Learning to cure the curse of drift in inertial odometry. In *AAAI*, 2018.

[7] Mahdi Abolfazli Esfahani, Han Wang, Keyu Wu, and Shenghai Yuan. Orinet: Robust 3-d orientation estimation with a single particular imu. *IEEE Robotics and Automation Letters*, 2019.

[8] Martin Brossard, Silvere Bonnabel, and Axel Barrau. Denoising imu gyroscopes with deep learning for open-loop attitude estimation. *IEEE Robotics and Automation Letters*, 2020.

[9] Scott Sun, Dennis Melamed, and Kris Kitani. Idol: Inertial deep orientation-estimation and localization. In *AAAI*, 2021.

[10] Roberto G Valenti, Ivan Dryanovski, and Jizhong Xiao. A linear kalman filter for marg orientation estimation using the algebraic quaternion algorithm. *IEEE Transactions on Instrumentation and Measurement*, 2015.

[11] Angelo M Sabatini. Quaternion-based extended kalman filter for determining orientation by inertial and magnetic sensing. *IEEE Transactions on Biomedical Engineering*, 2006.

[12] Mark Euston, Paul Coote, Robert Mahony, Jonghyuk Kim, and Tarek Hamel. A complementary filter for attitude estimation of a fixed-wing uav. In *IEEE IROS*, 2008.

[13] Sheng Shen, Mahanth Gowda, and Romit Roy Choudhury. Closing the gaps in inertial motion tracking. In *ACM MobiCom*, 2018.

[14] Miaomiao Liu, Sikai Yang, Wyssanie Chomsin, and Wan Du. Real-time tracking of smartwatch orientation and location by multitask learning. In *ACM SenSys*, 2022.

[15] Arm tracking dataset, 2022. https://github.com/mmmmliu/RTAT/.

[16] Yang Liu, Zhenjiang Li, Zhidan Liu, and Kaishun Wu. Real-time arm skeleton tracking and gesture inference tolerant to missing wearable sensors. In *ACM MobiSys*, 2019.

[17] Sheng Shen, He Wang, and Romit Roy Choudhury. I am a smartwatch and i can track my user's arm. In *ACM Mobisys*, 2016.

[18] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 2017.

[19] Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, et al. Towards playing full moba games with deep reinforcement learning. *NeurIPS*, 2020.

[20] Hai Nguyen and Hung La. Review of deep reinforcement learning for robot manipulation. In *IEEE IRC*, 2019.

[21] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[22] Miaomiao Liu, Kang Yang, Yanjie Fu, Dapeng Wu, and Wan Du. Driving maneuver anomaly detection based on deep auto-encoder and geographical partitioning. *ACM Transactions on Sensor Networks*, 2023.

[23] Nagesh Yadav and Chris Bleakley. Accurate orientation estimation using ahrs under conditions of magnetic distortion. *MDPI Sensors*, 2014.

[24] Bingfei Fan, Qingguo Li, Chao Wang, and Tao Liu. An adaptive orientation estimation method for magnetic and inertial sensors in the presence of magnetic disturbances. *MDPI Sensors*, 2017.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017.

[26] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A Preiss, Nora Ayanian, and Gaurav S Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. In *IEEE IROS*, 2019.

[27] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, et al. Dota 2 with large scale deep reinforcement learning. *arXiv*, 2019.

[28] Hang Yan, Qi Shan, and Yasutaka Furukawa. Ridi: Robust imu double integration. In *ECCV*, 2018.

[29] Sachini Herath, Hang Yan, and Yasutaka Furukawa. Ronin: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods. In *IEEE ICRA*, 2020.

[30] Huatao Xu, Pengfei Zhou, Rui Tan, Mo Li, and Guobin Shen. Limu-bert: Unleashing the potential of unlabeled data for imu sensing applications. In *ACM SenSys*, 2021.

[31] Huatao Xu, Pengfei Zhou, Rui Tan, and Mo Li. Practically adopting human activity recognition. In *ACM MobiCom*, 2023.

[32] Aaqib Saeed, Tanir Ozcelebi, and Johan Lukkien. Multi-task self-supervised learning for human activity detection. *ACM IMWUT*, 2019.

[33] Linh Kastner, Johannes Cox, Teham Buiyan, and Jens Lambrecht. All-in-one: A drl-based control switch combining state-of-the-art navigation planners. In *IEEE ICRA*, 2022.

[34] Xianzhong Ding, Wan Du, and Alberto E Cerpa. MB2C: Model-based deep reinforcement learning for multi-zone building control. In *ACM BuildSys*, 2020.

[35] Xianzhong Ding, Wan Du, and Alberto Cerpa. OCTOPUS: Deep reinforcement learning for holistic smart building control. In *ACM BuildSys*, 2019.

[36] Xianzhong Ding and Wan Du. DRLIC: Deep reinforcement learning for irrigation control. In *ACM/IEEE IPSN*, 2022.

[37] Xiyuan Zhang, Xiaohan Fu, Diyan Teng, Chengyu Dong, Keerthivasan Vijayakumar, Jiayun Zhang, Ranak Roy Chowdhury, Junsheng Han, Dezhi Hong, Rashmi Kulkarni, et al. Physics-informed data denoising for real-life sensing systems. In *ACM SenSys*, 2023.