LLIO: Lightweight Learned Inertial Odometer

Yan Wang, Jian Kuang, Xiaoji Niu and Jingnan Liu

***Abstract*—The 3D position estimation of pedestrians is a vital module to build the connections between persons and things. The traditional gait model-based methods cannot fulfill the various motion patterns. And the various data-driven-based inertial odometry solutions focus on the 2D trajectory estimation on the ground plane, which is not suitable for AR applications. TLIO (Tight Learned Inertial Odometry) proposed an inertial-based 3D motion estimator that achieves very low position drift by using the raw IMU measurements and the displacement prediction coming from a neural network to provide low drift pedestrian dead reckoning. However, TLIO is unsuitable for mobile devices because it is computationally expensive. In this paper, a lightweight learned inertial odometry network (LLIO-Net) is designed for mobile devices. By replacing the network in TLIO with the LLIO-Net, the proposed system shows a similar level of accuracy but remarkable efficiency improvement. Specifically, the proposed LLIO algorithm was implemented on mobile devices and compared the computational efficiency with TLIO. The inference efficiency of the proposed system is up to 12 times improved than that of TLIO. Source code can be found on github.**

***Index Terms*—Pedestrian Dead Reckoning (PDR), Inertial Navigation, Tightly-coupled Fusion, AI-Based Methods, Internet of Things (IoT)**

I. Introduction

Augmented reality (AR) exhibits tremendous potential for improving the quality of life. To support AR, a pedestrian positioning system can provide high accuracy 3D trajectories in indoor and outdoor environments and play a crucial role in connecting AR devices to the internet of things (IoT) network [**?**].

Various techniques have been adopted to achieve indoor navigation in recent years. Positioning systems based on Bluetooth low-energy (BLE) [**?**] and WIFI [**?**] [**?**] [**?**] can only achieve low accuracy positioning. Ultra-wideband (UWB) systems [**?**] can provide decimeter-level positioning accuracy in theory, but their performance significantly degrades in non-line-of-sight (NLOS) environments. Furthermore, both techniques mentioned above rely on pre-installed infrastructures.

Vision-based systems, such as the visual-inertial navigation system (VINS), have seen tremendous success today. The VINS [**?**] [**?**] can achieve high accuracy positioning over a long period through combined vision and inertial measures. Meanwhile, the hardware cost of VINS has become acceptable for consumer-level devices. These advantages have made the VINS one of the best for determining indoor positions, especially for AR. However, despite the impressive performance of state-of-the-art VINS solutions, applying these methods in product scenarios remains challenging. For example, the vision-based system relies heavily on consistent feature association. However, feature association cannot be achieved in certain challenging scenarios (such as positioning in a dark room or when the camera is blocked by obstacles). Thus, a system that can provide consistent pose estimation independent of external environments is necessary.

An inertial measurement unit (IMU) collects, i.e., linear acceleration (or specific force for strict speaking) and angular rate data, which are used in an inertial navigation system (INS) to estimate 3D motion relative to the first instance. The INS is a fully self-contained positioning system. In other words, it estimates trajectory without any dependency on the external environment. This feature indicates that INS complements the visual-based system well in AR. In fact, INS is utilized widely in mobile devices that need indoor positioning. However, the MEMS IMUs embedded in mobile devices, such as mobile phones and AR headsets, cannot provide long-term motion estimation alone. This is because the noise in low-end IMUs is strong, and the position accumulation error of strapdown INS is proportional to the square of time.

Pedestrian dead reckoning(PDR) uses sensors in mobile devices to detect gait information and form a dead-reckoning model. Some approaches [**?**] used prior knowledge of human motion to eliminate the accumulation error of velocity. One way of applying the prior knowledge is to detect gait cycles and use this information to estimate trajectories. However, this approach consists of several sub-modules: step detection, step length estimation and step orientation estimation. Each module requires several hand-designed rules or machine learning. For hand-designed rules, it is difficult to determine rules that are suitable for every scenario and all users. For machine learning, it is not easy to collect massive datasets with the ground truth for certain sub-modules, e.g., step detection.

Recent research has shown that data-driven inertial odometers can directly provide trajectories by integrating the average velocity through machine learning. Many approaches have focused on achieving 2D positioning [**?**] [**?**] [**?**] [**?**]. IONet [**?**] first proposed an LSTM-based architecture to estimate relative displacement in the ground plane. RoNIN [**?**] assumed that the global orientation is calculated by fusing linear acceleration, angular rate, and magnetic. Then, the velocity was estimated using a neural network (including ResNet, LSTM, and TCN) based on acceleration and gyroscope data represented in the gravity-aligned frame. IDOL [**?**] uses acceleration, angular

This work was supported in part by the National Key Research and Development Program of China (Grant No.2016YFB0502202) and the Special Fund of Hubei Luojia Laboratory (220100007) (Corresponding author: Jian Kuang)

X. Niu is with the GNSS Research Center, Wuhan University, Wuhan, Hubei, CO 430072 PR China. He is also with the Artificial Intelligence Institute of Wuhan University. (e-mail:xjniu@whu.edu.cn)

Jingnan Liu, J. Kuang and Y. Wang are with the GNSS Research Center, Wuhan University, Wuhan, Hubei, CO 430072 PR China (e-mails: jnliu@whu.edu.cn;kuang@whu.edu.cn; wystephen@whu.edu.cn;)

rate, and magnetic to estimate global orientation rather than use global orientation estimated conventionally. It achieves the best accuracy in terms of both orientation and position. Compared to traditional PDR, these learning-based methods exhibit higher accuracy and are more robust for various motion patterns.

However, for AR headsets in complex environments, a 3D pose estimator is necessary. Tight learned inertial odometry (TLIO) [**?**] achieved learned inertial odometry for AR headsets and can estimate 3D poses accuracy for complex scenarios. It adopts ResNet to estimate the 3D displacement in short periods and uses a Kalman Filter to fuse it with IMU measurements to achieve long-term dead reckoning. It exhibits the best performance in field testing but is computationally expensive. Compared with the visual solution, the method proposed in this paper only uses IMU observations, which is not affected by the external environment and can provide more stable positioning performance.

However, computational efficiency is a vital metric for AR applications run on mobile devices because the computation power of mobile devices is limited. In reality, the efficiency bottleneck of TLIO is the ResNet-based neural network, which is used to infer 3D displacement. More specifically, the ResNet-based architecture adopted in TLIO is computation expensive and not friendly for mobile-device code implementations. Previous researchers replaced LSTM-based with WaveNet-based architecture in IONet and significantly increased computing efficiency [**?**]. However, the size of the tested datasets is relatively simple, and their performance needs to be verified when faced with large data sets.

Recent multilayer perceptron (MLP) models [**?**] [**?**] [**?**] have shown potential to replace ResNet because their architectures can provide a better efficiency and accuracy trade off. For example, recent MLP-based models can improve efficiency while achieving similar accuracy in image classification. Moreover, the MLP architecture mainly uses matrix multiplication, which has been highly optimized in mobile devices. This fact indicates that the MLP architecture could be easily implemented on mobile devices.

In this paper, we propose a lightweight learned inertial odometry for mobile devices whose primary goal is to improve computing efficiency while ensuring that the accuracy is not significantly reduced. This paper has two major contributions:

- We proposed a lightweight MLP-based network to perform regression on both the 3D displacement and the corresponding covariance. Specifically, we use this network to replace the ResNet architecture in TLIO and evaluate the system performance. The proposed networks provide similar performance and are (1.9 - 12.0)x faster than ResNet-based methods when implemented on mobile devices.
- We conduct systematic research into the relationship between the computational efficiency and the positioning performance of the neural network models on mobile devices.

The remainder of the paper is organized as follows. Section II gives a brief description of the entire system. Section III describes the whole solution in detail. Section IV uses real test datasets to prove that the proposed network achieves similar accuracy while significantly improving efficiency. Section V summarizes the entire study.

We denoted the proposed system as the LLIO and the lightweight MLP-based network as LLIO-Net for the remainder of this paper.

II. System Overview

The proposed system uses raw IMU measurements (linear acceleration and angular velocity) and performs 3D motion estimation using the first instance. As shown in Figure.1, this system consists of two components: a stochastic clone extended Kalman filter (SCEKF) [**?**] and a lightweight inertial odometry neural network (denoted as LLIO-Net).

The SCEKF estimates the 3D motion (including position, orientation, and velocity) and the biases of IMU. Block IMU mechanization is the propagation of the SCEKF. It predicts the system state through INS mechanization based on IMU raw measurements. The input of the measurement update of SCEKF is the 3D displacement provided by the LLIO-Net. In summary, the filter tightly couples the raw IMU measurement and the displacement, which is provided by LLIO-Net to estimate the 3D motion and the IMU biases.

The LLIO-Net takes a sequence of IMU measurements represented in a gravity-aligned frame to estimate the displacement between the first and last instances. In the IMU coordinate conversion block, the IMU measurements are converted from the IMU frame to the navigation frame using the rotation matrix estimated by the SCEKF. The network block estimates the displacement and the corresponding covariance based on the converted IMU measurements. The LLIO-Net is inferred every 0.1 seconds and uses the previous 1 second of IMU measurements as inputs. Thus, each measurement of IMU was used ten times for inferencing.

The IMU measurement is utilized twice in the entire system. Firstly, the raw IMU measurements are input for the IMU mechanization to estimate the prior distribution of the system state. Secondly, in the measurement update, the displacement is estimated by the IMU measurements used to mitigate the accumulation errors of SCEKF. The primary information source of the measurement update is the human motion patterns memorized in the LLIO-Net rather than the IMU measurement itself.

III. Algorithm Description

A. Coordinate Definition

In this paper, three coordinate frames are defined: the navigation frame denoted as $\mathbf{F}^N$, the t-th body frame denoted as $\mathbf{F}^{B_t}$, and the t-th local gravity-aligned coordinate denoted as $\mathbf{F}^{L_t}$. $\mathbf{F}^{B_t}$ aligned the coordinates of IMU at t moments. $\mathbf{F}^N$ is a gravity-aligned coordinate. It is aligned with the IMU center at the initial moment. $\mathbf{F}^{L_t}$ is the gravity-aligned coordinate frame at which the yaw is the same as $\mathbf{F}^{B_t}$. In this paper, the 3D motion in the SCEKF is parameterized as position (t_{nb_t}) of the t-th body frame, rotation (R_{nb_t}) from the t-th body frame to the navigation frame, and the velocity of the t-th body frame in the navigation frame. The IMU raw

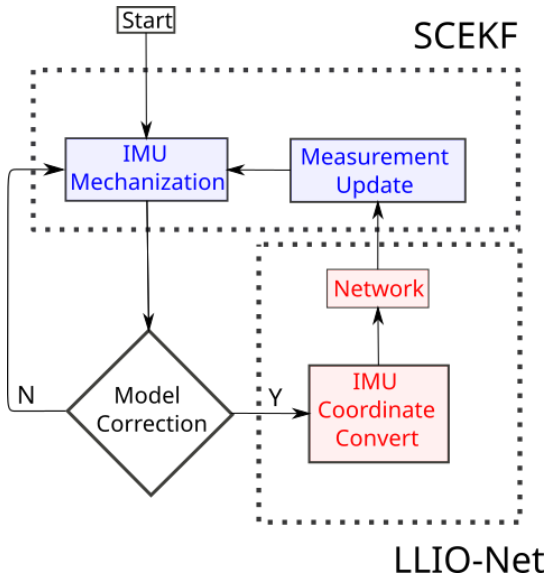

Fig. 1. System flowchart. The block named model correction selects a keyframe for inference displacement based on LLIO-Net. In this paper, the key frame is selected every 0.1 s.

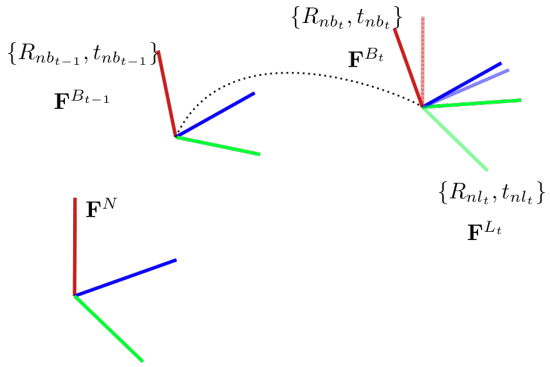

Fig. 2. Coordination Definition

measurements at the t-th moments are denoted as $a_t^{B_t}$ and $\omega_t^{B_t}$. Moreover, a_t^N and ω_t^N denote them represented in the navigation frame.

B. Lightweight Learned Inertial Odometry Network

In this section, we introduce the lightweight learned inertial odometer network (LLIO-Net). Figure 3 shows the framework of LLIO-Net.

1) Network architecture: The LLIO-Net uses a residual multi-layer perceptron (ResMLP) architecture [**?**] as a feature extractor to predict displacement and corresponding covariance. Compared with the traditional MLP, the ResMLP uses fewer parameters and can establish interactions between any two positions in the feature matrix, which the traditional MLP does. Compared with the ResNet, the ResMLP can achieve long-range interaction easier and with lower inductive bias. The proposed LLIO-Net consists of three modules. The feature conversion module rearranges the raw input as a feature matrix. The ResMLP module extracts high-level features from the input feature embeddings. The regression module regresses the displacement and corresponding covariance.

The feature conversion module rearranges the IMU measurements in the navigation frame. Using the IMU measurements between $t-L$ and t moments, the input is a $6\times L$ matrix. The input is split into N_{patch} patches, where each patch contains $L_{feature}$ measurements ($L = N_{patch} \times L_{feature}$). Then, each patch is flatted, and all features are combined. Subsequently, we obtain N_{patch} $(6 \times L_{feature})$-dimensional embeddings. The resulting set of N_{patch} embeddings is fed to a sequence of ResMLP blocks

The ResMLP module consists of a sequence of ResMLP blocks that all have the same structure. Before the ResMLP modules, a linear layer converts the N_{patch} $(6 \times L_{feature})$-dimensional embeddings into N_{patch} $L_{feature}^{inner}$-dimensional embeddings. $L_{feature}^{inner}$ is the feature dimensions in the ResMLP block. Each ResMLP block is a combination of an affine layer (`AFF`), linear layer, and gaussian error linear unit (GELU) layer (`GELU`).

The affine layer function performs a modified layer normalization. It simply rescales and shifts the input component-wise [**?**]. More specifically, the affine layer is defined as follows:

$$\texttt{AFF}_{\alpha,\beta}(x) = Diag(\alpha)x + \beta \tag{1}$$

where α and β are learnable vectors. Note that $\texttt{AFF}()$ for a matrix is applied independently to each column of the matrix.

Overall, the ResMLP is a combination of the cross-patch interaction block and the cross-channel interaction block. The cross-patch interaction block is defined as

$$Z = M + \texttt{AFF}((A\ \texttt{AFF}(M)^T)^T) \tag{2}$$

The cross-channel interactions block is defined as

$$Y = Z + \texttt{AFF}(C\ \texttt{GELU}(B\ \texttt{AFF}(Z))) \tag{3}$$

where A, B, and C are the main learnable parameters. The dimensions of the parameter matrix A are $N_{patch} \times N_{patch}$. Consequently, the dimensions of Z are the same as those of M, i.e., $N_{patch} \times (L_{feature}^{inner})$. The dimensions of B and C are $L_{feature}^{inner} \times (E \times L_{feature}^{inner})$ and $(E \times L_{feature}^{inner}) \times L_{feature}^{inner}$, respectively. Thus, the dimensions of Y are the same as those of Z and M. Here, E is the expansion dimension. Note that, in contrast to the original ResMLP, we added a dropout layer after `GELU`. This block is not shown in Figure 3.

The regression block uses the features extracted from ResMLP module to estimate two 3D vectors: displacement $\hat{d}_t^N$ and the diagonal of the covariance matrix $\Sigma_{\hat{d}_t^N}$. We assumed that the uncertainty of $\hat{d}_t^N$ at each axis is independent to simplify the problem. Thus, the covariance matrix $\Sigma_{\hat{d}_t^N}$ is a diagonal matrix. The network structure is shown in Figure 3 and consists of the average pooling layer, linear layer, and GELU layer.

2) Training Methodology: The LLIO-Net is trained based on two loss functions: the mean square error (MSE) loss function for $\hat{d}_t^N$ and the negative log-likelihood (NLL) loss function for $\hat{d}_t^N$ and $\Sigma_{\hat{d}_t^N}$ together. The MSE loss is defined as

$$\mathscr{L}_{MSE}(d, \hat{d}) = \frac{1}{n}\sum \|d - \hat{d}\|^2 \tag{4}$$

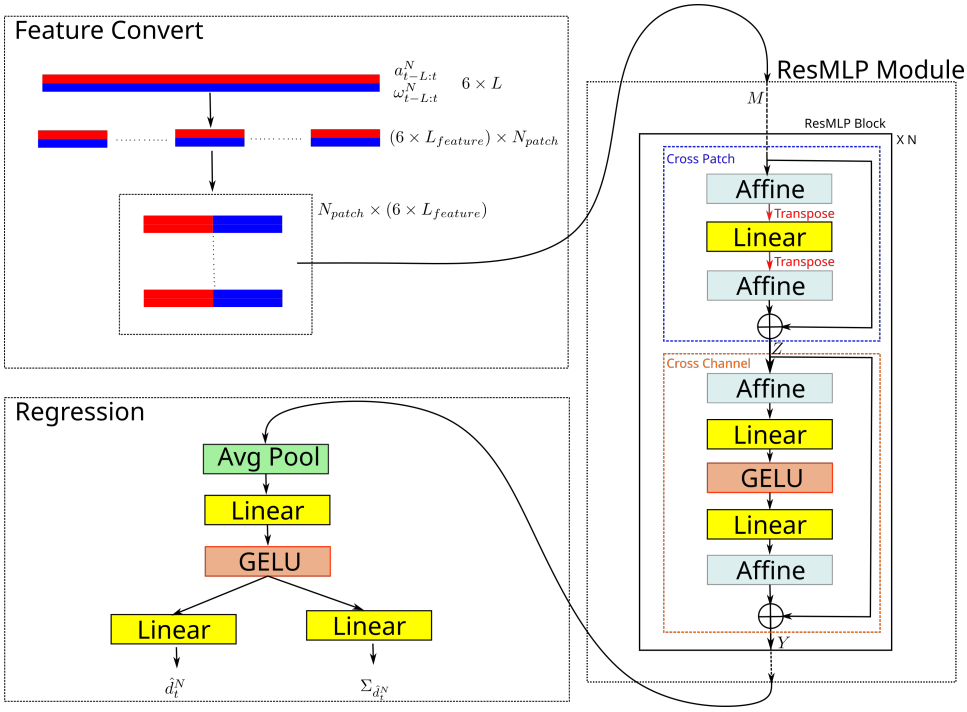

Fig. 3. Framework illustration of LLIO-Net.

where d is the ground truth displacement and $\hat{d}$ is the estimated displacement of the network. By minimizing $\mathscr{L}_{MSE}$, the network learned to estimate 3D displacement.

The NLL loss function is defined as

$$\mathscr{L}_{NLL}(d, \hat{d}, \hat{\Sigma}) = \frac{1}{n}\sum \frac{1}{2} log(det(\hat{\Sigma})) + \frac{1}{2}\|d - \hat{d}\|^2_{\hat{\Sigma}} \quad (5)$$

where $\hat{\Sigma}$ is the corresponding covariance matrix of $\hat{d}$. Additionally, $\|d - \hat{d}\|^2_{\hat{\Sigma}}$ is defined as

$$\|d - \hat{d}\|^2_{\hat{\Sigma}} = (d - \hat{d})^T \hat{\Sigma} (d - \hat{d}) \quad (6)$$

. By minimizing $\mathscr{L}_{NLL}$, the network learned to estimate the covariance corresponding to the 3D displacement.

During the training stage, we adopt the same training strategy as proposed in TLIO [**?**]. $\mathscr{L}_{MSE}$ is adopted first for training until the network converges. Then, we switch to using $\mathscr{L}_{NLL}$ only to train $\hat{\Sigma}$ and $\hat{d}$ together until the network converges again.

C. State Clone Extended Kalman Filter

1) System State Definition: The full systemstate at t moment is defined as

$$X_t = (s_t, \eta_1, ..., \eta_m) \quad (7)$$

where η are past system states, and s_t is the current system state. More specifically,

$$\eta_i = [R_{nb_i}, t_{nb_i}], \quad (8)$$

$$s_t = [t_{nb_t}, R_{nb_t}, v_{nb_t}, b_a, b_g] \quad (9)$$

We express R_{nb_t} as the rotation from $\mathbf{F}^{B_t}$ to $\mathbf{F}^N$, and t_{nb_t} and v_{nb_t} are the position and velocity of $\mathbf{F}^{B_t}$ in $\mathbf{F}^N$, respectively. b_a and b_g are the IMU accelerometer and gyroscope biases. Indeed, we use the IMU noise model defined as

$$a^{b_t}_t = \hat{a}^{b_t}_t + b_a + n_a \quad (10)$$

$$\omega^{b_t}_t = \hat{\omega}^{b_t}_t + b_g + n_g \quad (11)$$

$a^{b_t}_t$ and $\omega^{b_t}_t$ are the measured values of acceleration and angular rate, respectively. $\hat{a}^{b_t}_t$ and $\hat{\omega}^{b_t}_t$ are the true values of acceleration and angular rate, respectively. n_a and n_g are random noise variables following a zero-centered Gaussian distribution. Moreover, the evolution of b_a and b_g is modeled as discrete random walk processing.

The error-state-based indirect Kalman filter is utilized in the proposed system. The error state indicates the difference between the estimated and real values, estimated in the SCEKF. It is defined as

$$\delta X_t = (\delta s_t, \delta\eta_1, ..., \delta\eta_m) \quad (12)$$

$$\delta s_t = [\delta t_{nb_t}, \phi_{nb_t}, \delta v_{nb_t}, \delta b_a, \delta b_g] \quad (13)$$

$$\delta \eta_i = [\delta t_{nb_i}, \phi_{nb_i}] \quad (14)$$

Hence, the dimension of the system is $15 + 6m$, where m is the number of cloned system states and 15 is the dimension of s_t.

Since the rotation cannot be added directly, the error of rotation ϕ_{nb_t} is defined as

$$\hat{R}_{nb_t} = R_{nb_t} exp_{SO3}(\phi_{nb_t}) \quad (15)$$

$\hat{R}_{nb_t}$ and R_{nb_t} represent the estimate and real values of rotation, respectively. $exp_{SO3}(\cdot)$ denotes the $SO(3)$ exponential map.

2) State Propagation: The filter propagates the system state using the IMU raw measurements based on IMU mechanization. Because the proposed system aims to perform pedestrian motion estimation, the trajectory length is limited. Thus, we ignored the earth's curvature. Meanwhile, the gravity g^N is assumed to be equal at every place of the navigation frame. The simplified strapdown IMU mechanization is defined as

$$\hat{R}_{nb_t} = \hat{R}_{nb_{t-1}} exp_{SO3}((\hat{\omega}_t^{B_t} - b_g)\Delta t) \tag{16}$$

$$\hat{v}_{nb_t} = \hat{v}_{nb_{t-1}} + g^N \Delta t + \hat{R}_{nb_t}(\hat{a}_t - b_a)\Delta t \tag{17}$$

$$\hat{t}_{nb_t} = \hat{t}_{nb_{t-1}} + 0.5(\hat{v}_{nb_{t-1}} + \hat{v}_{nb_t})\Delta t \tag{18}$$

The cloned states need not update in the propagation stage.

The error-state covariance propagation can be written as

$$P_t = \Phi_t P_{t-1} \Phi_t^T + G_t Q G_t^T \tag{19}$$

$$\Phi_t = \begin{bmatrix} \Phi_t^s & 0 \\ 0 & I_{6m} \end{bmatrix}, G_t = \begin{bmatrix} G_t^s \\ 0 \end{bmatrix} \tag{20}$$

where Φ_t^s and G_t^s are the linearized state propagation matrix of the previous state $\hat{s}_{t-1}$ and all noise (including sensor noise and biased random walk noise), respectively. I_{6m} is a $6m$-dimension identity matrix.

3) State Augmentation: The measurement update of the SCEKF using the relative position of the current system state and a previous system state. Thus, the previous system state should be maintained in the SCEKF through stochastic cloning. The cloned system state is a direct copy of the current system state (only t_{nb_t} and R_{nb_t}). The probability propagation of the stochastic clone step in the proposed system is defined as

$$P_t^{new} = \begin{bmatrix} I_{15} & 0 & 0 \\ 0 & I_{6m} & 0 \\ A & 0 & 0 \end{bmatrix} P_t \begin{bmatrix} I_{15} & 0 & 0 \\ 0 & I_{6m} & 0 \\ A & 0 & 0 \end{bmatrix}^T \tag{21}$$

$$A = \begin{bmatrix} I_{3\times3} & 0_{3\times3} & 0_{3\times9} \\ 0_{3\times3} & I_{3\times3} & 0_{3\times9} \end{bmatrix} \tag{22}$$

4) Measurement Update: LLIO-Net provides the pseudo measurement using the acceleration and angular rate in a gravity-aligned coordinate frame. Thus, the output of LLIO-Net is represented in these gravity-aligned frames. As described in Section II, each IMU measure uses 10 times in LLIO-Net. Converting the measurements at different moments to the same coordinate frame can avoid redundant coordinate conversion. Thus, all IMU measurements are converted to the navigation frame, and the output of LLIO-Net is represented in the navigation frame. However, the displacement in the navigation frame imposed a constraint on the absolute heading. More specifically, the displacement in the navigation frame indicates the absolute heading observable in the SCEKF. Nevertheless, the absolute heading is unobservable in theory. To mitigate this problem, the 3D displacement was converted to the local gravity-aligned frame F^{L_t}, anchored to the t-th body frame F^{B_t}.

The displacement and its covariance, which the LLIO-Net outputs, are represented in F^N and are denoted as $\hat{d}_t^N$ and $\Sigma_{\hat{d}_t^N}$, respectively. Additionally, the information used in the measurement update is defined as

$$\hat{d}_t^{L_t} = \hat{R}_{yaw_t}^T \hat{d}_t^N \tag{23}$$

$$\Sigma_{\hat{d}_t^{L_t}} = \hat{R}_{yaw_t}^T \Sigma_{\hat{d}_t^N} \hat{R}_{yaw_t} \tag{24}$$

$\hat{R}_{yaw_t}$ is the heading rotation matrix of $\hat{R}_{nb_t}$. More specifically, $\hat{R}_{nb_t}$ can be decomposed to the three-rotation matrix ($\hat{R}_{nb_t} = \hat{R}_{yaw_t}\hat{R}_{pitch_t}\hat{R}_{roll_t}$), where $\hat{R}_{yaw_t}$, $\hat{R}_{pitch_t}$, and $\hat{R}_{roll_t}$ denote yaw, pitch, and roll, respectively.

Hence, the measurement function can be written as:

$$h(X_t) = R_{yaw_t}^T(t_{nb_j} - tnb_i) = \hat{d}_t^{L_t} + n_{\hat{d}_t^{L_t}} \tag{25}$$

where t_{nb_j} represent the position of the j-th moment. In this paper, the j-th moment is 1 second before i-th moment. $n_{\hat{d}_t^{L_t}}$ follows the normal distributed $\mathcal{N}(0, \Sigma_{\hat{d}_t^{L_t}})$.

In practice, there are some abnormal pseudo observations contained in the LLIO-Net outputs. To mitigate the effect of abnormal observations, a χ^2-test is employed. In detail, the observation is checked by the following condition:

$$\|R_{yaw_t}^T(t_{nb_j} - t_{nb_i}) - \hat{d}_t^{L_t}\|_{HPH+\Sigma_{\hat{d}_t^{L_t}}} < \alpha \tag{26}$$

H is the Jacobian matrix of $h(X_t)$. α is the threshold of χ^2-test. We choose $\alpha = 11.345$ corresponding to 99% of the χ^2 distribution with 3 degree of freedom. Furthermore, to avoid continuous reject pseudo observable leads to SCEKF crashed, we directly accept the LLIO-Net output if the previous three are rejected.

IV. Experiments

In this section, we compared our proposed LLIO-Net with TLIO when used with a head-mounted AR device. We refer to the ResNet architecture in TLIO as ResNet for the remainder of this paper. Furthermore, we compared three versions of LLIO-Net, denoted as ResMLP512, ResMLP256, and ResMLP512. All metrics are compared on the test set, which is never present in the training stage. Note that all methods use the same setup, except for the hyperparameters that define and train the networks.

The remainder of this section is organized as follows. Section IV-A describes the test implementation details. Section IV-B describes the metrics to evaluate the accuracy of the estimated trajectories. Section IV-C compares the accuracies of all methods. Section IV-D analyzes the inference efficiencies of the proposed methods. Section IV-E study the impact of the different components in the proposed LLIO-Net.

A. Setup

1) Data Preparation: The dataset is collected using an Asus Tango phone, which is widely used in the domain of data-driven inertial odometry. The Asus Tango phone can estimate 3D motion through a fisheye global shutter camera and embedded IMU module based on the visual-inertial odometry technique. In our experiments, we close the trigger of area learning to obtain a smooth trajectory. The trajectory is output by the visual-inertial odometry function as ground truth in both

the training and evaluation stages. The full dataset contains over 40 hours of head-mounted pedestrian data, including various activities (walking, standing still, sitting down and up, and going down and up the stairs). The datasets are captured by six people with multiple different physical devices. Thus, the dataset contains various individual motion patterns and IMU systematic errors. In the AR applications, this ground truth is easy to collect by the AR device itself. We follow the data split method of TLIO and split the dataset into 80% training, 10% validation, and 10% testing subsets randomly.

2) Data Augmentation Strategy: As described before, we use L IMU samples as input for the network. We select $L = 100$ as we collect IMU at 100 Hz. Thus, we send 1 second of IMU samples to the network at each instance. We first convert all IMU measurements from the body frame to the navigation frame based on the ground truth rotation matrix in the training stage. Moreover, the following data augmentation strategies are adopted to improve the generalization of the network:

- Use a random horizontal rotation on the IMU measurements and displacement together. ($[-180, 180]^\circ$), because we assumed that the AR headset could be oriented to an arbitrary heading with respect to the walking direction.
- Add a random bias to the IMU measurements ($[-0.2, 0.2]$ m/s^2 and $[-0.5, 0.5]^\circ/s$) to make the network robust to the IMU bias.
- Add a random perturbation of gravity direction ($[0, 5]^\circ$) to make the network robust to the gravity orientation perturbation.

3) SCEKF Setting: In this study, we use measurements updated at 10 Hz, providing 1 second of IMU samples to estimate displacement, as described before. Thus, the SCEKF contained 9 cloned states in the filter. Furthermore, SCEKF removes the last cloned state immediately after the measurement update. In the evaluation stage, the SCEKF needs to be initialized. We initialize the SCEKF based on the rotation matrix at the first moment only to simulate the scenario that the IMU system should be working individually. This is because the rotation could be provided by an AHRS system easily. The bias of IMU is set to zero as the initial value and is estimated online.

4) Training Details: We implemented all models using a PyTorch 1.8 framework [**?**]. The ResNet uses the hyperparameters provided in TLIO [**?**]. Those hyperparameters exhibit the most outstanding performance in our dataset as well. The proposed ResMLP series model uses 6 ResMLP blocks with a 0.2 dropout probability. The patch length L_{patch} is 25. The total length L is 100 for 1 second IMU measurements collected at 100 Hz. The inner feature dimensional $L^{inner}_{feature}$ of ResMLP is 512, 256, and 128 for ResMLP512, ResMLP256, and ResMLP512, respectively. All models are trained via an ADAM optimizer [**?**]. The ResNet uses a learning rate of 1e-4. Moreover, the ResMLP series uses a learning rate of 5e-4. We trained each of the model configurations using an NVIDIA GTX 3090.

B. Evaluation Metric

To evaluate the positioning performance, we define the following metrics similar to TLIO:

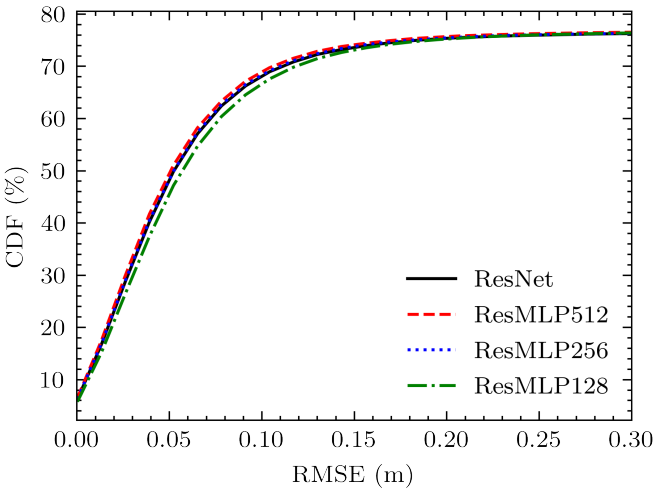

Fig. 4. Comparison of the inference error CDFs of different models in the test set. The network infers the displacement in 1 second as described before.

- ATE (m): $\sqrt{\frac{1}{n}\sum \|t_{nb_t} - \hat{t}_{nb_t}\|^2}$
  t_{nb_t} and $\hat{t}_{nb_t}$ are the ground truth position and estimated position at the t-th moment, respectively. The absolute translation error (ATE) is computed as the root-mean square error (RMSE) of the estimated trajectory and the ground true trajectory.
- RTE-Δt (m):
  $\sqrt{\frac{1}{n}\sum \|R^T_{yaw}(t_{nb_t} - t_{nb_{t-\Delta t}}) - \hat{R}^T_{yaw}(\hat{t}_{nb_t} - \hat{t}_{nb_{t-\Delta t}})\|^2}$
  R_{yaw} and $\hat{R}_{yaw}$ are the ground truth and estimated heading rotation matrix, respectively. The relative translation error (RTE) is the error of the trajectory represented in the local gravity-aligned frame at the $t - \Delta t$-th moment. Hence, the RTE is not affected by the global yaw drift. In our implementation, $\Delta t = 1\ minute$.
- RTE-ΔL (m):
  $\sqrt{\frac{1}{n}\sum \|R^T_{yaw}(t_{nb_t} - t_{nb_k}) - \hat{R}^T_{yaw}(\hat{t}_{nb_t} - \hat{t}_{nb_k})\|^2}$
  RTE-ΔL is the relative translation error (RTE) for every 10 m trajectory ($\Delta L = 10m$).
- AYE (°): $\sqrt{\frac{1}{n}\sum \|\gamma_t - \hat{\gamma}_t\|^2}$
  γ_t and $\hat{\gamma}_t$ are the heading orientation of the ground truth and estimated trajectory, respectively. The absolute yaw error (AYE) is the RMSE of the absolute heading drift. In practical, we calculation the rotation difference through a three-axis rotation matrix and decompose the yaw component to calculate the AYE.
- RYE-Δt (°): $\sqrt{\frac{1}{n}\sum \|(\gamma_{t+\Delta t} - \gamma_t) - (\hat{\gamma}_{t+\Delta t} - \hat{\gamma}_t)\|^2}$
  The relative yaw error (RYE) is calculated in the same manner as the AYE. $\Delta t = 1\ minutes$.

C. System Performance

We describe a systematic comparison of the ResNet and ResMLP series in this section. Specifically, we compared the inference accuracy of the network and the positioning accuracy of the entire system using different networks.

Table I gives an overview of the performance comparison. The distance error in Table I indicates the average error of

TABLE I
PERFORMANCE COMPARISON BETWEEN THE RESNET BASED SOLUTION AND THE PROPOSED LLIO-NET.

Model	Distance Error (m)	ATE (m)	RTE-Δt (m)	RTE-ΔL (m)	AYE (°)	RYE-Δt (°)
ResNet	0.111	**6.68**	0.900	0.239	11.54	2.46
ResMLP512	**0.108**	7.40	**0.884**	**0.237**	12.99	2.39
ResMLP256	0.110	6.80	0.963	0.255	**11.48**	**2.36**
ResMLP128	0.119	7.51	1.10	0.286	13.1	2.44

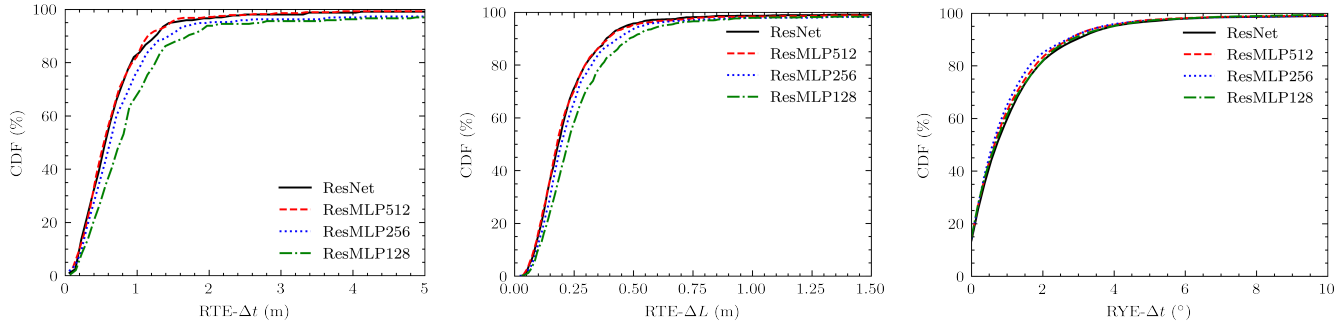

Fig. 5. CDFs of relative error metrics of different methods in test set.

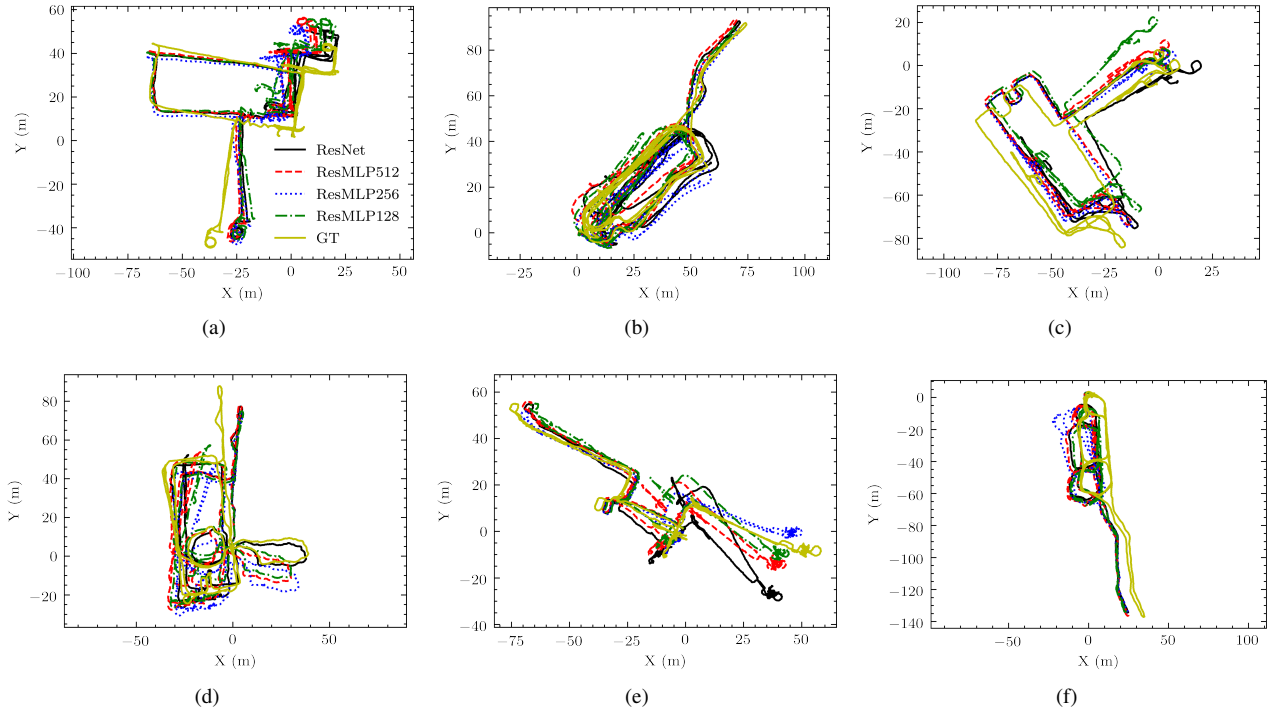

Fig. 6. Selection of trajectories with different contours. The ground truth trajectory generated from the Tango phone denoted as GT and marked in yellow. The ResNet trajectory is denoted by the black line. The ResMLP512, ResMLP256, and ResMLP128 are represented by red, blue, and green, respectively.

the inference results defined as $\frac{1}{n}\sqrt{\sum \|t_{nb_t} - \hat{t}_{nb_t}\|^2}$, which is slightly different from the RMSE. This metric is used to directly evaluate the neural network performance. Additionally, Figure 4 depicts the cumulative distribution function (CDF) of inference RMSE. The ResMLP series shows similar inference performance for both average distance error and the CDF of inference RMSE. Specifically, the ResMLP512 and ResMLP256 achieve slightly better inference accuracy than ResNet, and the ResMLP128 is slightly worse than ResNet.

Table I also provides the ATE, RTE-Δt, RTE-ΔL, AYE, and RYE-Δt of the trajectories using different networks. Note that all the relative metrics use a sliding window to calculate those metrics. The step length of the sliding windows is $\frac{1}{10}$ of the sliding window length (etc. 6s for TRTE-Δt). Even ResMLP512 achieved a slightly better RTE-Δt, RTE-ΔL, and RYE than ResNet, but the ResNet showed better ATE and AYE. Since those trajectories are collected over 15 minutes, the AYE and ATE are easily affected by random

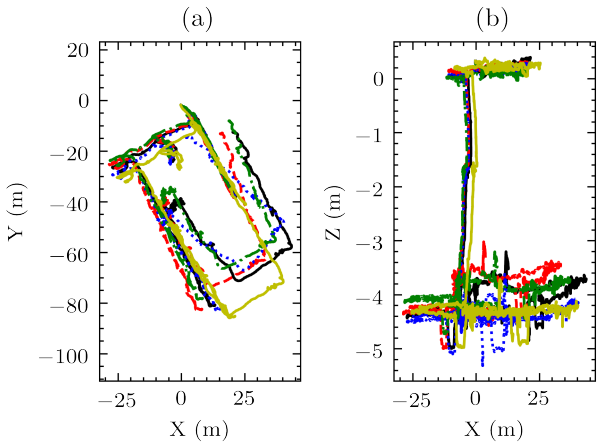

Fig. 7. Selected trajectory with height illustrated. (a) illustrated trajectory in the x-y plane. (b) illustrated trajectory in the x-z plane. The ground truth trajectory generated from the Tango phone is denoted as GT and depicted in yellow. The black line represents the ResNet trajectory. The ResMLP512, ResMLP256, and ResMLP128 are represented in red, blue, and green, respectively.

perturbation. Figure 5 shows the CDFs of RTE-Δt, RTE-ΔL, and RYE-Δt. All methods exhibit the same level of accuracy except ResMLP128. The ResMLP128 shows slightly worse performance than ResNet from the perspective of stochastic metrics.

Meanwhile, we select a group of trajectories to compare the positioning performance. Figure 6 shows a selection of trajectories to illustrate their performances with different contours. All methods work well when the pedestrian walks straight and worsen when the pedestrian stands still or walks around a small area for a long time. Figure 7 depicts a 3D trajectory. All methods can correctly estimate the trajectory when a person is going up and down stairs.

This section fully compares the performances of the proposed networks and ResNet with respect to inference accuracy, stochastic absolute and relative metrics, and illustration of trajectories in 2D and 3D. In summary, ResMLP512 and ResMLP256 show similar performance to the ResNet. ResMLP128 is slightly worse than ResNet in all metrics but still shows the same level of performance in positioning. As mentioned in TLIO [**?**], its performance has significant advantages over other algorithms. This article has similar accuracy to TLIO, so we can consider that it has an advantage in positioning accuracy compared to other data-driven inertial odometry algorithms.

Furthermore, because the ground truth is used in visual-inertial odometry without loop-closure, the ground truth trajectory exhibits some cumulative positioning error, as shown in Figure 6. The following section compares the network performance based on inference accuracy and relative positioning accuracy to avoid the cumulative error.

D. Inference Efficiency on mobile devices

In the proposed system, the main computation cost is generated from two modules, namely the SCEKF (including propagation, state clone, and measurement update) and the network. We implement a C++ version of the SCEKF to test the computational efficiency. The C++ version of SCEKF can process data 190X faster than in real-time (2.9 seconds processing time for a 561 seconds dataset). Meanwhile, the inferencing speed of networks is significantly lower than that of SCEKF. Specifically, if ResNet is used for inference, it will cost 33 seconds that inference on this data set. Thus, the efficiency bottleneck of this 3D inertial odometry is the network. With an aim to achieve implementation on mobile devices, the efficiency of the network is systematically analyzed in this section.

At the same time, we compared the proposed method with the article whose aim is to achieve a lightweight version of IONet. The paper replaced the LSTM architecture network in IONet with a WaveNet-based network architecture, which reached a significant performance improvement. However, the inputs and outputs of IONet are different from the method discussed in this article. Thus, we made some changes based on the network structure used in the paper to achieve the same function for a fair comparison. In detail, we add two two-layer MLP to output displacement and corresponding covariance based on the output of LSTM and WaveNet. The name of compared methods we use is built from these aspects. For example, LSTM-2 is a model that takes 2 layers of bi-directional long short-term memory (Bi-LSTM) with 128 hidden states. And WaveNet-32 is a WaveNet-based model that takes 8 layers and uses 32 channels as described in [**?**].

To illustrate the computational efficiency, we compared the inference time in the following devices:

- Google Pixel 3 (announced 2018)
  Equipped with an octa-core CPU (Snapdragon 845 4x2.5 GHz & 4x1.6 GHz).
- Huawei Mate 30 5G (announced 2019)
  Equipped with an octa-core CPU (Kirin 990 2x2.86 GHz & 2x2.36 GHz & 4x1.95 GHz).
- Huawei Mate 40 (announced 2020)
  Equipped with an octa-core CPU(Kirin 9000 1.x3.13 GHz & 3x2.54 GHz & 4x2.05 GHz).

The mobile devices use PyTorch for Android 1.8.0 as a deep learning inference framework. Every model performs 1000 inferences to calculate the average inference time. We tested two versions of the just-in-time (JIT) compilation model on mobile devices. The JIT model conversion uses the JIT compiler to convert the model based on JIT alone. The Mobile model conversion uses the JIT compiler and PyTorch built-in mobile optimizer. Moreover, the inference result of the JIT model and the Mobile model are compared. The difference between the results is smaller than 1e-14. Thus, the JIT model and the Mobile model can provide the same positioning performance.

Table II shows the computation efficiency comparison. The floating-point operations (FLOPs) of each model are provided. The inference time ratio compared to ResNet at each setup was also provided. The FLOPs of ResMLP256 and ResMLP128 are significantly lower than those of the ResNet and show efficiency improvement when either the JIT model or the Mobile model is used. The ResMLP512 has more FLOPs than ResNet but exhibits better inference efficiency in testing

TABLE II
COMPARISON OF COMPUTATION EFFICIENCY

Model			ResNet	ResMLP512	ResMLP256	ResMLP128	LSTM-2	LSTM-1	WaveNet-64	WaveNet-32
Error (m)			0.111	**0.108**	0.110	0.119	0.113	0.128	0.116	0.127
FLOPs (M)			21.15	25.78	6.54	**2.08**	53.9	27.7	28.47	7.13
Time (ms)	Pixel 3	JIT	9.9 (1x)	14.5 (0.68x)	4.8 (2.1x)	1.7 (**5.7x**)	43.3 (0.23x)	19.9 (0.50x)	5.6 (1.8x)	2.7 (3.7x)
		Mobile	7.6 (1x)	3.9 (1.9x)	1.6 (4.7x)	0.8 (**9.2x**)	149.4 (0.05x)	62.6 (0.12x)	5.3 (1.4x)	2.5 (3.0x)
	Mate 30	JIT	7.4 (1x)	8.1 (0.92x)	2.4 (3.1x)	1.0 (**7.5x**)	29.9 (0.25x)	13.8 (0.54x)	4.1 (1.8x)	2.3 (3.2x)
		Mobile	6.1 (1x)	2.1 (2.9x)	0.9 (7.2x)	0.5 (**12x**)	79.5 (0.08x)	38.6 (0.16x)	3.9 (1.6x)	2.1 (2.9x)
	Mate 40	JIT	6.2 (1x)	6.0 (1.03x)	1.9 (3.3x)	0.7 (**8.3x**)	23.3 (0.27x)	10.8 (0.57x)	3.1 (2.0x)	1.7 (3.6x)
		Mobile	5.1 (1x)	1.8 (2.8x)	0.7 (7.1x)	0.4 (**12x**)	63.7 (0.08x)	32.5 (0.16x)	2.9 (1.8x)	1.5 (3.4x)

the Mobile model. This may benefit from the optimization strategy in the mobile optimizer of PyTorch. In detail, the ResMLP series shows better efficiency improvement on the Mobile model. The inference time of ResMLP256 is 4.7-7.2 times faster than the ResNet. Moreover, the ResMLP128 shows a 9.2-12 times faster inference time but with slightly worse accuracy.

Compared with other algorithms, LSTM has obvious disadvantages in inference efficiency. WaveNet-64 can achieve inference accuracy similar to that of ResNet, ResMLP512 and ResMLP256. However, its reasoning efficiency has a significant disadvantage compared with the ResMLP series algorithms proposed in this paper. On the other hand, WaveNet-32 shows similar inference accuracy to LSTM-1, consistent with the experimental results in [**?**]. But this inference accuracy is worse than ResNet. Figure. 8 shows the inference performance and accuracy of ResNet, WaveNet and ResMLP. Since the inference efficiency of LSTM is significantly worth than other algorithms, it is not shown in this figure.

It is worth noting that we monitored the CPU usage during the test. In the inference process of all models, the CPU runs at full capacity. Therefore, the length of inference time can reflect the computational load required by the model.

To illustrate the relation between accuracy and efficiency of the whole method, Figure 9 shows the relationship between the inference time and RTE-Δt of ResMLP and ResNet models on a Huawei Mate 30.

In summary, the ResMLP series shows a higher accuracy–efficiency ratio than the ResNet, LSTM and WaveNet. Although all the models can run in real-time on current mobile devices, the computational cost is still a key metric for evaluating the suitability of executing the algorithm on mobile devices. The positioning algorithm usually functions as a fundamental component of other applications and runs during the entire workflow. Thus, the improvement in the efficiency of ResMLP is vital in this scenario.

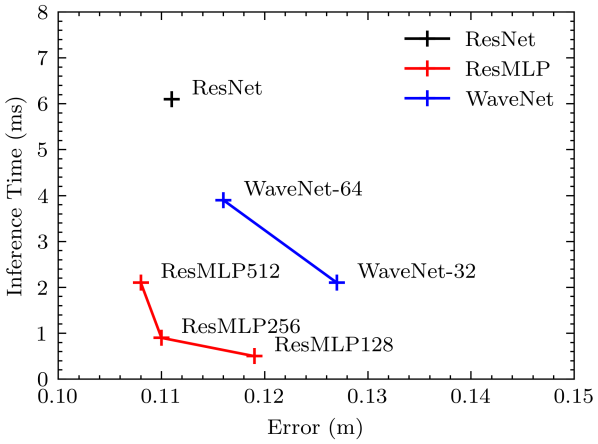

Fig. 8. Inference time and inference error of mobile models on Mate 30.

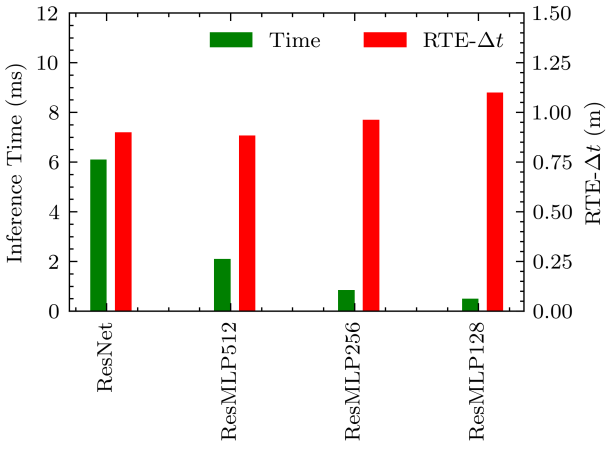

Fig. 9. Inference time and RTE-Δt of mobile models on Mate 30.

E. Ablation Study

This section provides and analyzes the effect of several parameters of the proposed LLIO-Net. As illustrated in Table III, all ablation experiments are conducted based on

TABLE III
THE RESULT OF ABLATION STUDY

Model	Feature Dimension	Expansion Dimension	Layer Number	Patch Size	Distance Error (m)	FLOPs(M)
ResMLP512	512	2	6	25	0.108	25.78
ResMLP256	**256**	2	6	25	0.110	6.54
ResMLP128	**128**	2	6	25	0.119	2.08
A	512	**1**	6	25	0.109	13.20
B	512	**4**	6	25	0.110	50.95
C	512	2	**8**	25	0.106	34.19
D	512	2	**4**	25	0.112	17.38
E	512	2	6	**10**	0.110	63.79
F	512	2	6	**50**	0.109	13.16

the ResMLP512, presented in Section IV-A. Each parameter different from the ResMLP512 are marked using bold. The meaning of conducted parameters can be found in Section III-B. Furthermore, in Table III, we provide the distance error and FLOPs in each experiment to compare both accuracy and efficiency simultaneously.

The feature dimension (ResMLP512, ResMLP256 & ResMLP128) and the layer number (ResMLP512, C & D) show significantly contribution to model performance and FLOPs. The expansion dimension (ResMLP512, A & B) and the patch size (ResMLP512, E & F) can influence the FLOPs but do not significant affect the prediction accuracy compared to the feature dimension and the layer number. However, an appropriate value is necessary to achieve high efficiency. Moreover, the feature dimension and the layer number should be priorities considered in order to obtain trade-off between accuracy and efficiency.

V. CONCLUSION

In this paper, we proposed LLIO, a lightweight learned inertial odometry, which introduces the LLIO-Net to replace the ResNet-based architecture. The LLIO-Net module estimates the 3D displacement and the corresponding covariance, which is essentially based on human motion patterns, to mitigate the accumulation error of the INS mechanization. The experiments proved that the proposed LLIO-Net could achieve the same level of accuracy as TLIO while significantly improving computational efficiency (2x to 12x faster). The inference efficiency test on mobile devices reveals that the proposed inertial odometry can be implemented on mobile devices and functions as a low-drift 3D pedestrian motion estimator. Because of its low computational load and low drift, the LLIO can be adopted as a backup for visual-inertial odometry in AR applications. Alternatively, it can function as an independent dead reckon module for fusing other sources of information.

Further work would focus on the generalization of the proposed model. For example, the performance for estimating the 3D trajectories of pedestrians without or with a small scale of labeled IMU sequences could be improved.

ACKNOWLEDGMENT

Tianyi Liu is thanked for sharing his opinion in the discussion of the method implementation. Thanks to RoNIN to provide the data collection software for Tango devices. The numerical calculations in this paper have been done on the supercomputing system in the Supercomputing Center of Wuhan University.

Yan Wang received the B.Eng. degree in Chemical Engineering and Technology from China University of Mining and Technology, Xuzhou, China, in 2016. And he received an M.S. degree in Computer Applied Technology from China University of Mining and Technology, Xuzhou, China, in 2019. He is currently pursuing a Ph.D. degree in GNSS Research Center, Wuhan University, Wuhan, China. His research interests focus on indoor navigation, sensor fusion algorithm, and computer vision.

Jian Kuang received the B.Eng. degree and Ph.D. degree in Geodesy and Survey Engineering from Wuhan University, Wuhan, China, in 2013 and 2019, respectively. He is currently a Postdoctoral Fellow with the GNSS Research Center in Wuhan University, Wuhan, China. His research interests focus on inertial navigation, pedestrian navigation, and indoor positioning.

Xiaoji Niu received the B.Eng. degree (with honors) in Mechanical and Electrical Engineering and the Ph.D. from Tsinghua University, Beijing, China. He was a Postdoctoral Fellow with the Mobile MultiSensor Systems (MFMSS) Research Group, Department of Geomatics Engineering, and the University of Calgary. He was a Senior Scientist with SiRF Technology, Inc. At present, he is a Professor of the GNSS Research Center and the Collaborative Innovation Center of Geospatial Technology at Wuhan University, Wuhan, China. His research interests focus on INS, GNSS/INS integration for land vehicle navigation, and pedestrian navigation.

Jingnan Liu is a member of the Chinese Academy of Engineering. He was born in 1943 and graduated from the Wuhan Institute of Surveying and Mapping with a bachelor's degree in astronomical geodesy. He obtained his master's degree in engineering in 1982. Professor Liu has long been engaged in research and teaching in geodetic surveying. He is considered a pioneer in the application of GNSS technology. Professor Liu has taken part in a number of research programs to promote the application of satellite positioning systems in China.