Real-Time Tracking of Smartwatch Orientation and Location by Multitask Learning

Miaomiao Liu
mliu71@ucmerced.edu
University of California, Merced
Merced, CA, USA

Sikai Yang
syang126@ucmerced.edu
University of California, Merced
Merced, CA, USA

Wyssanie Chomsin
wchomsin@ucmerced.edu
University of California, Merced
Merced, CA, USA

Wan Du
wdu3@ucmerced.edu
University of California, Merced
Merced, CA, USA

ABSTRACT

Arm posture tracking is essential for many applications, such as gesture recognition, fitness training, and motion-based controls. Smartwatches with Inertial Measurement Unit (IMU) sensors (i.e., accelerometer, gyroscope, and magnetometer) provide a convenient way to track the orientation and location of the wrist. Existing orientation estimations are based on predefined data fusion methods that do not consider the variations in the data quality of different IMU sensors. Existing location estimations rely on the estimated orientation results. A small orientation estimation error may cause high inaccuracy in location estimation. Moreover, these location estimation algorithms, e.g., Hidden Markov Model and Particle Filters, cannot provide real-time tracking on commercial mobile devices due to high computation overhead. This paper presents *RTAT*, a Real-Time Arm Tracking system that tackles the above limitations in a data-driven way. *RTAT* estimates both orientation and location simultaneously using a multitask learning neural network. It also incorporates a unique attention layer and a dedicated loss function to learn the dynamic relationship among IMU sensors. *RTAT* supports real-time tracking by performing model inference on smartphones. Finally, to train *RTAT*'s neural network, we develop an easy-to-use labeled data collection system that uses a low-cost virtual reality system to provide orientation and location labels for the smartwatch. Extensive experiments show *RTAT* significantly outperforms existing state-of-the-art solutions in both accuracy and latency.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile devices**; • **Computer systems organization** → **Real-time systems**; • **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Arm Tracking, Inertial Measurement Unit, and Multitask Learning

SenSys '22, November 6–9, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9886-2/22/11.
https://doi.org/10.1145/3560905.3568548

ACM Reference Format:
Miaomiao Liu, Sikai Yang, Wyssanie Chomsin, and Wan Du. 2022. Real-Time Tracking of Smartwatch Orientation and Location by Multitask Learning. In *The 20th ACM Conference on Embedded Networked Sensor Systems (SenSys '22), November 6–9, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3560905.3568548

1 INTRODUCTION

Real-time and accurate arm posture tracking is essential to the performance of many applications, e.g., gesture recognition [1, 2], gym exercise assessment [3], and motion-based control [4]. Once we know the a user's forearm length and orientation and location of wrist, we can estimate the user's elbow location and track the arm movements [3, 5]. As smartwatches are more pervasively adopted, they provide a more easily accessible arm posture-tracking alternative to other infrastructure-based systems, such as wireless sensing [6–8], visible light [9, 10] and customized wearable sensors [11]. The Inertial Measurement Unit (IMU) inside a smartwatch, can be used to track arm motions [3, 5, 12, 13].

Arm posture tracking requires continuous knowledge of a smartwatch's three-dimensional (3D) orientation and location in a desired reference frame, e.g., the Global Reference Frame (GRF), typically <North, East, Up>. However, all three IMU sensors report sensing readings in the Watch's Reference Frame (WRF). We must find the transformation between these two reference frames. The WRF-to-GRF rotation is the orientation of the smartwatch in GRF. This rotation is needed to calculate the watch's location in GRF.

Gyroscopes measure the angular velocity around the three axes of a device. Intuitively, with a known initial orientation, subsequent orientations can be estimated by integrating gyroscope readings over time. However, estimation error accumulates with the noise and bias of the gyroscope [14–17]. A^3 calibrates the orientation results using the direction anchors measured by the accelerometer and magnetometer when the smartphone is static or moving at a constant speed [18]. Only in these moments can gravity be accurately decomposed from accelerometer readings, since the accelerometer measures a mixture of gravity and linear acceleration. The magnetometer measures geomagnetic North, which can be leveraged to estimate the heading angle of a device. Once the directions of both gravity and magnetic North are known in GRF, the device's orientation in GRF can be uniquely determined. However, such calibrations can only be done opportunistically because a device may

pause infrequently. MUSE [12] adopts a complementary filter to do calibration continuously using magnetic North because the magnetic North is unaffected by the motion of a device. However, such fixed calibration methods prove insufficient because the magnetic readings are sensitive to nearby environments. For example, when the magnetic readings are skewed by nearby ferromagnetic materials, as is often true indoors [12, 18], orientation estimation should not strongly rely on the magnetic readings. Therefore, an adaptive orientation estimation method is necessary for incorporating the readings of three IMU sensors according to their varying quality.

State-of-the-art approaches for wrist location estimation rely on orientation estimation. ArmTrak [5] adopts a Hidden Markov Model (HMM) and MUSE [12] uses a Particle Filter. ArmTroi [3] reduces the computational latency of the HMM proposed in ArmTrak. These methods use estimated orientation to project accelerometer readings onto the desired reference frame. However, locations derived from inaccurate orientations can fall outside the possible space. Furthermore, they cannot provide real-time wrist location tracking on smartphones if the sampling frequency is higher than 10 Hz. A high sampling frequency (e.g., 50 Hz) is desired for fine-grained arm tracking applications, e.g., motion-based control.

In this paper, we develop a Real-Time Arm Tracking (*RTAT*) system for smartwatches. *RTAT* uses a multitask neural network for simultaneous prediction of both orientation and location. It leverages Bidirectional Long Short-Term Memory (BiLSTM) as its backbone, considering its effectiveness for time-series data processing [17, 19]. Our proposed multitask learning scheme offers the following benefits over conventional arm tracking systems. First, *RTAT* estimates orientation and location simultaneously [20]. Orientation and location estimations are two related tasks. Solving them together with multitask learning improves accuracy and avoids additional overhead for training two separate models. Second, as a supervised learning method, *RTAT* learns the best fusion scheme of three IMU sensor data streams from the labeled data, which is more immune to the noise of IMU sensor data [14–17]. Unlike conventional sensor fusion methods with predefined calibrations, *RTAT* adapts to complex temporal variations in the quality of the three IMU sensors' readings. Third, *RTAT* is faster than conventional location estimation methods (i.e., 0.1633 ms vs. 2337.50 ms for processing 50 samples).

Building the aforementioned learning system involves the following three challenges. 1) How do we teach the neural network to adapt to temporal variations of IMU sensor data quality? 2) If we utilize a loss function that minimizes the difference between inferred results and the labels, the neural network's outputs are independent at different timestamps. However, arm movements are not just sequences of independent wrist orientations and locations. How can we incorporate the temporal correlation of consecutive arm movements in our neural network? 3) A large-scale labeled dataset is necessary to develop the system. It is challenging to collect accurate orientations and locations of a smartwatch at a low cost.

We tackle the above three challenges with a set of novel techniques developed for *RTAT*. 1) We develop a BiLSTM-based multitask neural network for processing three IMU data streams. We also design an attention mechanism in our neural network architecture to dynamically learn the importance of different IMU sensor streams. 2) We incorporate smooth losses into the loss function of our neural network. These losses ensure the changing rate of orientation and location are sufficiently similar to the labels. 3) We use the Meta Quest 2 VR system to collect labeled data to train and test our model offline. Meta Quest 2 includes one headset and two touch controllers. To collect data, users wear a smartwatch and hold a VR controller simultaneously while moving their arms. Meta Quest 2 accurately measures its controller's orientation and location, but not that of the smartwatch. Thus we develop a labeled data collection system that converts the VR measurements into the orientation and location of the smartwatch. *RTAT* only relies on the VR system for training data collection. Once our model is well trained, it can estimate the orientation and location of users' wrist by smartwatch alone.

We collect data from nine volunteers (four females and five males) at two places. This research study has been approved and exempted by the IRB committee of UC Merced. We do not pre-define any gestures for the volunteer users. We ask them to move their arms freely at their natural speed. They perform random arm gestures or daily gestures, including driving, drinking, writing, exercising, push and pull, drawing and so on. Each user performs gestures in their own way. The arm motion style and motion speed of different users are different. To the best of our knowledge, this is the first dataset of fine-grained orientation and location labels for smartwatch tracking. The dataset is available at https://github.com/mmmmliu/RTAT.

We use data from five volunteers to train and evaluate a general neural network model. Extensive experiments show that *RTAT* reduces the orientation estimation error by up to 51% and 31.63% at two places, respectively, compared to state-of-the-art orientation estimation methods. *RTAT* also reduces the location error up to 45% and 46.9% these two places, respectively, compared with state-of-the-art location estimation methods. Additionally, we test *RTAT* with four new users whose data is not used for training. The experiment results show *RTAT* has no significant performance degradation on new users. Furthermore, *RTAT* can easily support real-time arm tracking with the maximum data sampling frequency on commercial smartphones.

In summary, this paper makes the following contributdions.

- To the best of our knowledge, this is the first work to leverage end-to-end deep learning for IMU-based arm tracking.
- This is the first work to track the orientation and location of a smartwatch simultaneously, rather than sequentially.
- We consider adjusting the importance of the three IMU channels according to their temporal variations.
- We develop a labeled data collection system to collect the orientation and location labels of a smartwatch.
- Extensive experiments are conducted to evaluate *RTAT* and baseline solutions.

2 RELATED WORK

Deep Learning for Device Orientation Estimation. Deep learning has been used in many applications, such as image processing [21, 22], wireless networking [23], smart buildings [24, 25], smart driving [26], smart irrigation [27], and map matching [28]. Recent literature has begun utilizing deep neural networks for IMU measurements processing [14, 29–31] and orientation estimation [15–17]. OriNet [15] uses an LSTM-based architecture to estimate the 3D orientation of flying robots from gyroscope readings. Brossard et al. [16] estimate device orientation by correcting gyroscope readings with a CNN, then integrating the corrected readings. These

methods do not explore learning from multi-modality sensors. The accelerometer and magnetometer can help estimate orientation when the gravity error or the deviation of magnetic field density is small. IDOL [17] proposes an Extended Kalman Filter (EKF) architecture to estimate device orientation. The prediction model of its EKF utilizes an LSTM-based neural network. This neural network estimates orientation using all three IMU sensors. The measurement model of its EKF is based on gyroscope readings integration. While Kalman Filter and its variants are dependent on the system noise parameters, the noise of IMU sensors is environment-dependent. IDOL uses a static diagonal propagation noise matrix for the gyroscope readings integration, which is not able to depict the system noise.

RTAT greatly differs from IDOL in the following ways. First, IDOL outputs its orientation result as a unit quaternion, which cannot be accurately predicted by a neural network without a dedicated loss design. Second, we propose a novel network architecture from IDOL. We design a multitask learning network to output orientation and location simultaneously rather than with separate neural networks. Third, we do not treat readings from the three IMU channels equally. Instead, we incorporate an attention layer to adjust the feature importance from the three IMU channels. Finally, we develop a labeled data collection system for smartwatch-based arm tracking.

Conventional Device Orientation Estimation. A^3 [18] intelligently selects the moments that gravity and magnetic North are reliable and calibrates the orientation estimation from gyroscope integration. However, the assumption that device motions have frequent pauses for resetting the orientation is inapplicable for wearables. MUSE [12] proposes that magnetic North is more trustworthy than gravity because it is unpolluted by device motion. It designs a magnetometer-centric sensor fusion algorithm based on the complementary filter for orientation tracking. However, magnetic North can only calibrate 2 of 3-DoF (Degrees of Freedom) of orientation, and magnetic fields can vary significantly within the same space due to the local ferromagnetic disturbances. The complementary filter's performance is highly dependent on the appropriate selection of its parameters. For example, when magnetic interference is high, we should adjust our confidence in the magnetometer readings. The complementary filter proposed by MUSE cannot adapt to different environments with its fixed magnetometer calibration parameters.

IMU-based Location Tracking. Prior studies [32–37] leverage multiple sensors to track the human body or upper limb movement. ArmTrak [5] proposes to recover and track the 3D arm posture using only a smartwatch. It leverages a Hidden Markov Model (HMM) to continuously estimate elbow and wrist locations. However, its computation latency is high, and it cannot support real-time performance on smartphones. ArmTroi [3] optimizes ArmTrak with HMM state reorganization and hierarchical search. It is faster than ArmTrak but still suffers from high latency with sampling rates above 10 Hz. MUSE [12] uses Particle Filters to estimate the smartwatch location. It achieves a higher location estimation accuracy than ArmTrak and ArmTroi, but with higher computational latency than ArmTroi Its real-time computation still cannot be afforded by a smartphone.

Human Skeleton Tracking. Various sensing modalities have been used for estimating the posture of the human skeleton, including vision [38], light [9, 10], wireless signals [39, 40], and mm-wave [8]. However, these systems require infrastructure support. They also have limited service coverage and performance will decrease when tracking multiple people simultaneously.

3 BACKGROUND & MOTIVATION

In this section, we introduce orientation representations and analyze the existing orientation estimation solutions.

3.1 Orientation Representation

The 3D orientation of an object can be represented in different ways, i.e., quaternion, rotation vector (axis/angle) and 3x3 rotation matrix. Each representation can be converted to another.

A unit quaternion is a 4D complex vector $q = (q_0, q_1, q_2, q_3)$.

$$q_0 = cos(\frac{\theta}{2}), q_1 = x \cdot sin(\frac{\theta}{2}), q_2 = y \cdot sin(\frac{\theta}{2}), q_3 = z \cdot sin(\frac{\theta}{2}). \quad (1)$$

It represents a rotation of degree θ along a vector (x, y, z) from the default orientation. All four items in a unit quaternion must satisfy the following constraint.

$$\sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1 \quad (2)$$

A unit quaternion can be uniquely transformed into a rotation matrix using the Rodrigues' rotation formula as follows.

$$\begin{bmatrix} 1-2(q_2^2+q_3^2) & 2q_1q_2-2q_3q_0 & 2q_1q_3+2q_2q_0 \\ 2q_1q_2+2q_3q_0 & 1-2(q_1^2+q_3^2) & 2q_2q_3-2q_1q_0 \\ 2q_1q_3-2q_2q_0 & 2q_2q_3+2q_1q_0 & 1-2(q_1^2+q_2^2) \end{bmatrix} \quad (3)$$

A unit quaternion can also be uniquely transformed into a 3D rotation vector by Equation (4).

$$\begin{aligned} &\vec{v} = (v_1, v_2, v_3) = (\theta \cdot x, \theta \cdot y, \theta \cdot z) = \theta \cdot (x, y, z) \\ &\theta = 2 \cdot arccos(q_0) \quad (\theta \neq 0) \end{aligned} \quad (4)$$

where $\|\vec{v}\| = \theta$, and its direction is the rotation axis. Uniquely, when $\theta = 0$, $\vec{v} = \mathbf{0}$. Any rotation vector can be uniquely transformed into a unit quaternion by Equation (5).

$$\begin{aligned} \mathbf{q} = (&cos(\frac{\theta}{2}), \frac{v_1}{\theta} \cdot sin(\frac{\theta}{2}), \frac{v_2}{\theta} \cdot sin(\frac{\theta}{2}), \frac{v_3}{\theta} \cdot sin(\frac{\theta}{2})) \\ &(\theta = \|\vec{v}\|, \|\vec{v}\| \neq 0) \end{aligned} \quad (5)$$

Uniquely, when $\|\vec{v}\| = 0$, $\mathbf{q} = (1, 0, 0, 0)$.

3.2 Conventional Orientation Tracking

We implement a complementary filter, representative of the conventional orientation estimation approach, and conduct a set of experiments under different scenarios to investigate its performance. The device we use is a Fossil Gen 5 smartwatch. A VR controller of Meta Quest 2 is used to provide ground truth orientations for evaluation. Details about our data processing are introduced in Section 4.5.

We find two places to do experiments: a room and a hallway. From our measurements, the magnetic field in the room is stable, whereas the magnetic field in the hallway is unstable. We ask one volunteer to wear the smartwatch and hold the VR controller by his/her left hand, then perform free gestures as introduced in Section 1. The experiments are conducted under three scenarios. For each, the user is asked to move his/her arm for ten minutes.

We implement three kinds of complementary filters, all primarily dependent on gyroscope readings integration. The first is an implementation of MUSE [12]. It fuses magnetometer and accelerometer

Table 1: Average orientation error of complementary filter at different places for ten-minute data.

	Speed(m/s)	Gravity Error(degree)	Magnet Deviation(degree)	Gravity Opportunities(%)	Sensors_to_use	Orientation Error(degree)
S1:Hallway	0.39	7.15	45.41	3.34 %	Mag. + Accl.	58.93
					Mag.	72.53
					Accl.	27.10
S2: Room	0.30	5.37	5.10	4.4%	Mag. + Accl.	19.17
					Mag.	46.93
					Accl.	29.88
S3: Room	0.98	20.47	9.0	1.94%	Mag. + Accl.	53.03
					Mag.	52.59
					Accl.	77.04

readings to calibrate orientation drift caused by gyroscope readings integration. The magnetometer readings are used continuously, while the accelerometer readings are used opportunistically (when the accelerometer roughly measures 9.8 m/s^2). The second fuses just magnetometer readings continuously, and the third fuses just accelerometer readings opportunistically to perform the calibration.

Table 1 shows the statistical analysis of the three scenarios: S1, S2 and S3. In Table 1, speed is calculated as the average moving speed of the device for the ten-minute data. Gravity error is the average angular difference between every measured gravity direction and the true gravity direction ('down'). It is highly affected by the motion of a device. Magnet deviation represents the angular deviation of the measured magnetic direction in GRF. It measures the stability of the magnetic field. The gravity opportunities refer to the percentage of data samples that are considered to be linear-acceleration-free over the ten-minutes data. They are considered as such moments when the accelerometer readings are 9.8 ± 0.3 m/s^2 during consecutive 500 ms. Sensors_to_use denotes which sensors are used to calibrate the gyroscope readings integration. S1 stands for the scenario where the magnetic field is unstable but the motions of the device are slow (North is inaccurate, gravity is accurate). S2 is the scenario where the magnetic field is stable and the motions of the device are slow (both North and gravity are accurate). S3 represents the scenario where the magnetic field is stable but the motions of the device are fast (North is accurate, gravity is inaccurate). By analyzing the results in Table 1, we get two observations.

Observation 1: *Incorporating magnetometer readings will hurt the orientation estimation when the magnetic field is distorted; and vice versa.* S1 and S2 are conducted at different places with similar motion speeds. They have similar gravity errors. S1 has 45.41° magnetic direction deviation, whereas S2 has 5.10°This indicates the magnetic field in the hallway is unstable, but stable in the room. The results of S1 in Table 1 show a 58.93° error when using two sensors to calibrate. The error when using either magnetometer or accelerometer is 72.53° and 27.10° respectively. Calibrating with two sensors results in a larger error than calibrating with solely accelerometer, but a smaller error than calibrating just with the magnetometer. This indicates incorporating magnetometer hurts the performance. In comparison with S2, calibrating with two sensors produces a smaller error than calibrating with one of them, which indicates the magnetometer improves the orientation estimation under this scenario.

Observation 2: *Incorporating gravity does not improve the orientation estimation when the device moves fast.* We further investigate the effect of gravity on orientation estimation with S3 in Table 1. In this scenario, the motions of the smartwatch are fast. The average moving speed is 0.98 m/s and gravity calibration opportunities occur only 1.94% of the time. The gravity error is larger than in S1 and S2. In this scenario, calibrating with two sensors gets almost the same error as calibrating just with the magnetometer, demonstrating that gravity does not improve the orientation estimation.

Summary: *A better sensor fusion scheme is needed to tolerate the noise from IMU sensors for accurate orientation tracking.* Each IMU sensor has its own limitations. Integration of gyroscope readings drifts over time. Accelerometer readings are highly motion-dependent. Magnetometer readings are highly environment-dependent. Due to these inherent hardware limitations, we need a more flexible data fusion method that can adapt to different scenarios automatically. Data-driven methods [14–17] have shown great potential for handling data noises and adapting to the variation of data distribution in many computer vision and natural language processing applications [41, 42].

3.3 Conventional Location Tracking

Conventional location tracking approaches of smartwatches rely on the orientation of the smartwatches. Orientation is used to transform the accelerometer readings into a desired reference frame, e.g., GRF, to infer the location of the device in GRF. As introduced in Section 1 and Section 2, three solutions have been proposed for smartwatch location tracking recently, ArmTrak [5], ArmTroi [3], and MUSE [12]. These solutions require pre-existing knowledge of user-specific information, including shoulder width and the lengths of the lower arm, upper arm and torso. They use user-specific information to generate 3D point clouds for each user and search the possible locations of the smartwatches from these point clouds. There are three main limitations to those solutions. First, the point cloud generation process is time-consuming. The more point clouds generated, the more time is needed. Second, point clouds are generated according to the user-specific information, which require each user to generate his/her personal point clouds and to run HMM or Particle Filters to estimate the possible locations of the smartwatch. Point clouds are not generalizable to different users. Third, the computation latency of such searching solutions is long and cannot be afforded by commodity mobile devices in real-time if the sampling rate is higher than 10 Hz.

Summary: *A new method that supports accurate real-time smartwatch location tracking on mobile devices is needed.* In this paper,

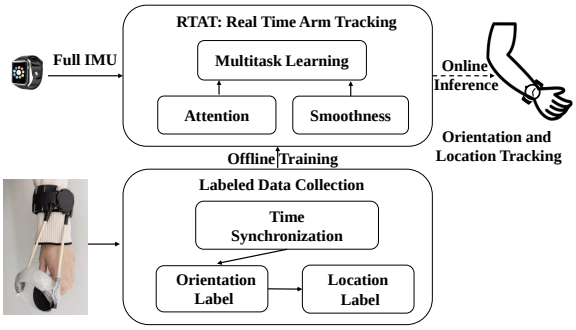


Figure 1: System Overview of *RTAT*

we exploit the benefits of deep learning to track the device's orientation and location.

4 THE DESIGN OF *RTAT*

In this section, we first introduce the overview of our system. We then show our design of multitask learning neural network, attention-based feature adjustment, and labeled data collection.

4.1 Overview

Figure 1 shows the two major parts of our system, i.e., real-time arm tracking and labeled data collection. During the offline training phase, we feed the collected IMU readings of the smartwatch and labels to *RTAT* to train a multitask model. During the online inference phase, a user can use the well-trained model to do inference by just wearing a smartwatch. The model can be executed on a smartphone. The smartwatch transmits the IMU sensor data stream to the smartphone via Bluetooth. The smartphone receives and then forwards the IMU readings to the model deployed on the smartphone. The labeled data collection system is no longer needed during the inference phase.

***RTAT*'s Arm Tracking System.** We design a multitask neural network to jointly predict the orientation and location of the wrist (Section 4.2). The inputs of the neural network are the IMU sensor data stream from a smartwatch, and the outputs are the corresponding orientation and location series. Since the importance of different IMU sensors in orientation estimation is varying over time, we design an attention mechanism on top of our multitask neural network (Section 4.3). The attention mechanism adjusts the input focuses of the network automatically according to the varying sensor data quality. Furthermore, to guarantee the smoothness of the inferred arm motions, we employ smooth losses for both orientation and location tracking (Section 4.4).

***RTAT*'s Labeled Data Collection.** To train a model with supervised learning, we need to build a training dataset. The dataset should consist of a large amount of time-series IMU readings from the smartwatch, and the orientation and location labels of the smartwatch. However, acquiring accurate labels for the smartwatch is a non-trivial task. Thus we develop a labeled data collection part (Section 4.5) to derive training labels. As depicted in Figure 1, the training data collection process requires volunteers to wear a smartwatch and hold a VR controller simultaneously. The VR system provides labels for the smartwatch. However, the readings we collect from the VR system describe the orientations and locations of the VR controller, not the smartwatch. Since the center of the VR controller and smartwatch are not aligned, training the model on unprocessed VR controller data would constitute training a model to predict the orientations and locations of the VR controller given IMU readings from the smartwatch. To fill in the gap between the acquired orientations and locations of the VR controller and the required ones of the smartwatch, we design a labeled data collection system. The system is built in three steps, i.e., time synchronization and alignment (Section 4.5.3 and Section 4.5.4), orientation label derivation (Section 4.5.1) and location label derivation (Section 4.5.2).

4.2 Multitask Learning Neural Network

Previous solutions usually estimate the orientation of devices first, and then the location. Multitask learning provides us the opportunity to solve multiple tasks simultaneously [20, 43]. We thus design a multitask neural network to jointly estimate the device orientation and location from the IMU readings. Our multitask neural network has two outputs, i.e., orientation and location.

Formally, we denote a posture P_w of a wrist as the union of an orientation ori_w and a location loc_w, where $P_w = \langle ori_w, loc_w \rangle$. Our multitask neural network is designed to learn a data-driven mapping from IMU measurements to orientations and locations as follows.

$$ori^t, loc^t = f(a^t, \omega^t, m^t) \tag{6}$$

where a^t, ω^t and m^t are the readings from accelerometer, gyroscope and magnetometer at timestamp t. Each of them is a 3D vector as depicted in Figure 2. f is the weight to be learned by the multitask network. ori^t and loc^t are the 3D orientation and 3D location predicted by the neural network at at timestamp t.

Figure 2 demonstrates our network architecture. The network consists of three BiLSTM layers [44] and two fully-connected (FC) layers. BiLSTM is one kind of Recurrent Neural Network (RNN). RNN is designed for processing sequential data. We set the length of a sequence as 32 time steps (32 data samples) in our implementation. A BiLSTM consists of two LSTMs, a forward LSTM and a backward LSTM. The forward one takes an input sequence in a forward direction, and the backward one in a backward direction. The BiLSTM layer's output is a combination of the two LSTM layers' outputs. We set the sampling rate of IMU sensors to 50 HZ. Data for an input sequence (32 samples) can be collected in less than one second. From Section 6.5.2, our model takes on average 2~3 ms to process one-second of data (50 samples) on smartphones.

Each BiLSTM layer takes the 3D vector sequences from one of the IMU sensors. The outputs of BiLSTM layers are the hidden states of the temporally dependent data. The hidden states from the three BiLSTMs are concatenated in the concatenation layer. The concatenated vectors are forwarded to each of the FC layers to predict the orientations and locations. A loss function plays an important role in the fast and accurate training of a neural network. The rest of this subsection focuses on our loss function design.

Orientation Output. A rotation is a process of 3 degrees-of-freedom (DoF). A quaternion we introduced in Section 3.1 is a 4D vector used to represent a 3D rotation. If we define the orientation output as a quaternion, we should normalize the square of the 4D vector as 1 to meet Equation (2). From our experiments, the neural network cannot learn quaternions well following this design. Thus,

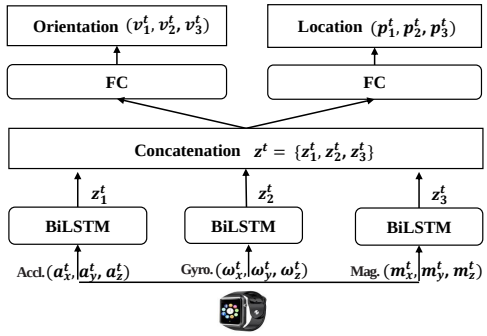


Figure 2: Multitask Network Structure.

we use a 3D rotation vector instead of a quaternion as the orientation output of our multitask neural network.

An orientation label we collect from the VR system at each timestamp is a unit quaternion. Therefore, we transform the 3D vector output of our neural network into a unit quaternion using Equation (5) when we calculate the loss. By comparing the inferred quaternions to the quaternion labels, the neural network is trained to learn from the labeled data. Since the quaternion label meets the constraint in Equation (2), we implicitly implant the constraint into our learning process to predict the 3D rotation vectors.

Loss Function for Orientation. Assume a quaternion label is q_t and the predicted quaternion is $\hat{q}_t$. The loss for orientation is:

$$L_{ori} = \frac{1}{T}\sum_{t=1}^{T} deg(\hat{q}_t \cdot q_t^{-1}) \tag{7}$$

$$deg(q) = 2 \cdot arccos((q_0', q_1', q_2', q_3')) = 2 \cdot arccos(q_0') = \theta \tag{8}$$

where q_t^{-1} is the inverse of q_t, and $\hat{q}_t \cdot q_t^{-1}$ returns a quaternion, which is the rotation from q_t to $\hat{q}_t$. The operator $\cdot$ is the Hamilton product, which represents the quaternion product. The function $deg(q)$ returns the quaternion difference of q_t and $\hat{q}_t$. The constant parameter T is the number of data samples in a training batch.

Loss Function for Location. Location can be represented as a 3D vector, with the form $p = (p_1, p_2, p_3)$. Mean squared error (MSE) between the predicted locations and labels is used for location loss.

$$L_{loc} = \frac{1}{T}\sum_{t=1}^{T}(p_t - \hat{p}_t)^2 \tag{9}$$

where the p_t and $\hat{p}_t$ are the position label and the predicted position at timestamp t, respectively.

Loss Function for Multitask Learning. With the definition of the orientation loss and the location loss, the overall loss of our multitask network is given below.

$$L_1 = \alpha L_{ori} + \beta L_{loc} \tag{10}$$

twhere α and β are hyper-parameters to balance the two losses.

4.3 Attention-based Feature Adjustment

As shown in the motivation experiments, three IMU sensors play varied roles in orientation estimation in the conventional filtering algorithms. We further investigate the importance of each sensor for orientation estimation in deep learning.

We build and train five models using data from the same user, each with different inputs. The inputs are different combinations

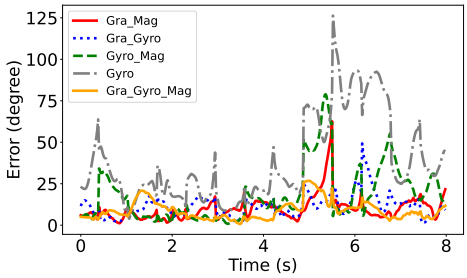


Figure 3: Orientation error over time of models with different combinations of sensor inputs.

Table 2: The orientation error (degree) and Be-The-Best (BTW, %) for ten-minutes data from different sensor combinations.

Model	Gra_Mag	Gra_Gyro	Gyro_Mag	Gra_Gyro_Mag	Gyro
Error	9.38	17.14	16.92	8.12	33.98
BTW	30.16	11.22	12.82	44.18	1.62

of data from the three IMU sensors. Gra, Mag and Gyro stand for the readings of the accelerometer, magnetometer and gyroscope respectively. For example, the inputs for "Gra_Mag" are the data combinations from accelerometer and magnetometer.

Figure 3 demonstrates how the error of different models varies over time. None of them consistently outperform the other sensor combinations. Each of them achieves the lowest error at different points. The model that utilized all three sensors had the most stably low error. Table 2 shows a statistical analysis over ten minutes of data. The third row, "Be-The-Best," presents the percentage of time the corresponding data combination can achieve the lowest prediction error. "Gra_Gyro_Mag" performs the best the majority of the time, but not always. Other models sometimes take the place.

From Figure 3 and Table 2, we observe that we still need to use the data from all three sensors as input for our neural network. To fully exploit the neural network's capabilities and achieve higher accuracy, we must dynamically identify the importance of each sensor.

We thus introduce an attention-based design on top of the multitask neural network to learn how important a sensor is at each timestamp. As shown in Figure 4, the network inputs include three parts; They are the data from accelerometer, gyroscope and magnetometer, respectively. The attention scheme is designed to automatically adapt the network to different parts of the inputs. Attention is an emerging technique. It is used for automatically adjusting the focuses of a DNN by multiplying a weighting vector, the value of each element in the vector can vary [3, 41]. In our case, the network should treat readings from different sensors differently. It intends to use the most effective portion (context) to derive outputs. We can exploit this ability to dynamically increase the weight of important sensor inputs and reduce the weight of unimportant sensor inputs. Hence, the attention scheme is suitable for learning the importance of different input channels.

As shown in Figure 4, we add an attention layer into the network, which learns to assign weights for different IMU input channels. Originally, as shown in Figure (2), z^t is the concatenation of the BiLSTMs' outputs, where $z^t = \{z_1^t, z_2^t, z_3^t\}$, they are from the accelerometer, gyroscope and magnetometer, respectively. This feature vector z^t serves as the input of the last two FC layers. Instead of equally fusing them into the last two layers, an adaptive context

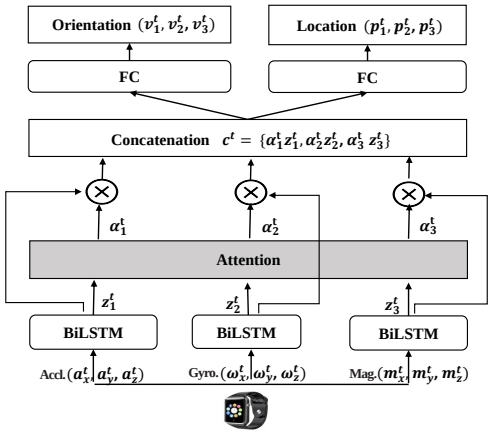


Figure 4: Attention Network Structure.

vector c_i^t is designed to weight z_1^t, z_2^t, z_3^t and then concentrate the weighted value as shown in the Equation (11) below.

$$c_i^t = concat(\alpha_1^t z_1^t,\ \alpha_2^t z_2^t,\ \alpha_3^t z_3^t) \tag{11}$$

where z_r^t and α_r^t are the features from different inputs and their corresponding weights. α_r^t measures the importance of the feature from each part of the input data. It differentiates the contributions of each input part to the orientation prediction. The weight α_r^t for a highly contributed z_r^t will be greatly increased in Equation (11) by the attention function. Otherwise, α_r^t will be gradually decreased. The attention function is typically realized as a single-layer multiplayer perceptron, such as $tanh(\cdot)$ and $ReLu(\cdot)$. The calculation of α_r^t is:

$$\begin{aligned} att^t &= tanh(W \cdot z^t + b) \\ \alpha_r^t &= f_{softmax}(att^t) \end{aligned} \tag{12}$$

where att^t is an intermediate variable; $f_{softmax}(\cdot)$ scales the weights α_r^t to the range $[0, 1]$. W and b are the trainable parameters to be determined in the training phase.

4.4 Smooth Losses

We train our multitask learning neural network by minimizing the distance between the predicted postures ($P_w = \langle ori_w, loc_w \rangle$) and the posture labels. However, this loss design treats the posture at each time point independently. Sometimes, the inferred movements of the human wrist may not be continuous and smooth over time, resulting in an unrealistic posture estimation. To fully leverage the temporal correlation between the wrist postures at two adjacent timestamps, we add a smoothness item in our loss function for both orientation and location predictions. The new smooth loss makes the difference between consecutive postures close to that of the labels:

$$L_{ori_s} = \frac{1}{T-1} \sum_{t=2}^{T} deg((q_t \cdot q_{t-1}^{-1}) \cdot (\hat{q}_t \cdot \hat{q}_{t-1}^{-1})) \tag{13}$$

$$L_{loc_s} = \frac{1}{T-1} \sum_{t=2}^{T} ((p_t - p_{t-1}) - (\hat{p}_t - \hat{p}_t))^2 \tag{14}$$

The difference between consecutive orientations measures the changing rate of orientation, which is the angular velocity. The difference of consecutive locations measures the changing rate of location, which is the velocity. With these two smooth losses, we extend the multitask learning loss function in Equation (10) as follows:

$$L_2 = \alpha L_{ori} + \beta L_{loc} + \gamma L_{ori_s} + \eta L_{loc_s} \tag{15}$$

where α, β, γ, and η are the hyper-parameters to balance these four losses. In our current implementation, we set all of them to 1. We optimize the above loss through RMSprop optimizer. By this design, we minimize both the loss of the posture (including orientation and location) and the loss of the posture's changing rate.

4.5 Labeled Data Collection

To train *RTAT*'s model, we need the orientation and location labels of the smartwatch for the IMU readings.

Why Meta Quest 2 is chosen for labeled data collection? To collect the labeled data, we may use Azure Kinect [45], VICON motion capture system [46], or a VR system.

Azure Kinect is not able to capture the pronation/supination (inward/outward rotation) of the forearm, which means it can only capture 2 DoF orientation of the wrist [47].

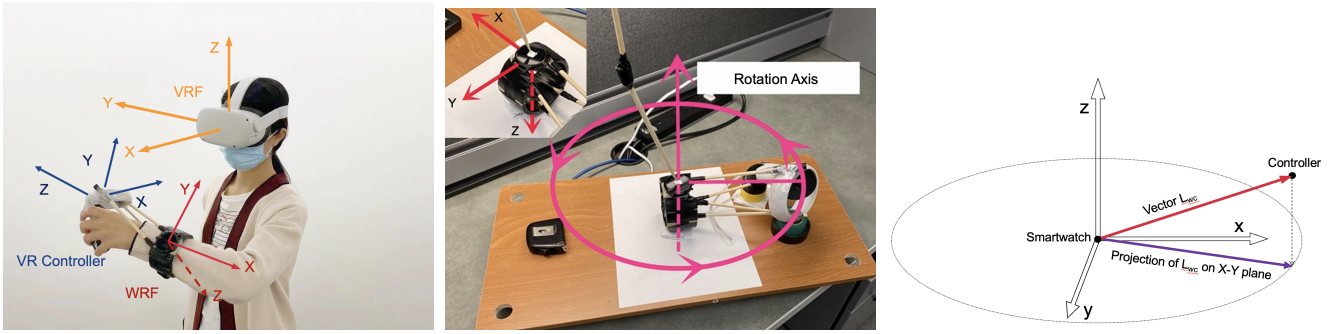
VICON motion capture system can provide accurate arm posture tracking. However, VICON cameras are expensive (more than $3000 each). Multiple cameras from different points of view in a room are needed to provide high-precision tracking. For instance, WiPose [40] uses 21 VICON cameras to provide labeled locations for WiFi-based joint tracking. Moreover, once the VICON system is installed in one room, it is costly to move it to another room.

Meta Quest 2 is the most advanced all-in-one VR system that provides accurate tracking of its controllers at a low cost ($299 for 64GB). It can be used in any indoor environment with ignorable setup efforts. Thus, we use a Meta Quest 2 to collect the labels.

Is Meta Quest 2 accurate enough for tracking? As shown in Figure 5 (a), the headset of Meta Quest 2 is embedded with four cameras on its four corners. It adopts Oculus Insight, the cutting-edge VR technology that leverages computer vision algorithms and visual-inertial simultaneous localization and mapping (SLAM) to compute 6-DoF postures (3-DoF orientation and 3-DoF location) [48]. Specifically, Oculus Insight combines information from multiple IMUs in its headset and controllers, i.e. ultra-wide-angle cameras in the headset and infrared LEDs in the controllers, to jointly track their orientation and location. Controlled experiments [49–51] have demonstrated that the orientation and location tracking errors of Meta Quest 2 are smaller than 0.85° and 0.7 cm respectively.

Labeled data in the VR Reference Frame (VRF). The orientation and location of the VR headset and controllers are tracked in VRF. Figure 5(a) shows the coordinates of WRF and VRF. The origin of VRF is the midpoint of the headset. The VRF can be established every time the VR system is used based on the initial position of the headset. We collect the orientation and location data of one VR touch controller in VRF. Our goal is to use the measurements of the VR controller to calculate the orientation and location of the smartwatch. *RTAT* uses the smartwatch's IMU sensor readings measured in WRF to infer its orientation and location in VRF.

Once the VRF is established, it will not change until the next setup, even if the headset moves. As long as the user wears the headset properly, VRF can represent the body's reference frame. Therefore, *RTAT* tracks the arm movements relative to the user's body. VRF-based labeled data also offers two advantages. 1) We



(a) Different reference frames (b) Rotation experiment (c) Projection of the vector

Figure 5: Labeled Data Collection System.

do not require users to stand in the exact same position when they collect labeled data or use *RTAT* for arm tracking since *RTAT* is focused on arm movements relative to the user's body. 2) We can combine the labeled data collected from the same user at different times. 3) We can also combine labeled data from multiple users to train a general *RTAT* model. Experiments in Section 6.2.3 show that our general model provides high performance across different users, including the users whose data has not been used for training.

Two challenges to collect labeled data with VR controllers.

- *Using the orientation and location of the VR controller to derive those of the smartwatch.* If we simply wear a smartwatch and hold one VR controller in the same hand to collect data, the relative orientation and location between these two devices may vary when a user moves her/his wrist. As shown in Figure 1, we bind the smartwatch and the VR controller onto a rigid frame. This is constructed with four sticks to ensure the two devices are static relative to each other. The VR system provides the orientations and locations of the controller, while we need those of the smartwatch.
- *Time Synchronization of VR system and smartwatch.* The smartwatch and VR system are based on two different time clocks. They need to be well synchronized to provide data at the same timestamps.

4.5.1 Orientation Label. We use T_{wv}, a 3×3 orthogonal rotation matrix introduced in Section 3.1 to denote the smartwatch's orientation in VRF. T_{wv} will be used as the orientation label. Similarly, T_{cv} represents the orientation of the VR controller in VRF. T_{cv} is measured by the VR system. Our goal is to calculate T_{wv} from T_{cv}. To do so, we need the watch's orientation relative to the controller T_{wc}, i.e., $T_{wv} = T_{wc} \cdot T_{cv}$. T_{wc} is a constant matrix, as the controller and the watch are fixed to each other. However, it is impossible to physically measure T_{wc}.

Fortunately, the orientation of a smartwatch in GRF, T_{wg}, can be accurately calculated when the smartwatch is static using gravity and magnetic North as direction anchors. If the transformation from VRF to GRF T_{vg} is known, we can transform the orientation of the smartwatch in VRF T_{wv} to the orientation of the smartwatch in GRF T_{wg}, and then get T_{wc}. Leveraging this resource, we update the strategy into $T_{wg} = T_{wv} \cdot T_{vg} = T_{wc} \cdot T_{cv} \cdot T_{vg}$. All four matrices are 3×3 orthogonal matrices. Since VRF does not change each time it is established, the transformation from VRF to GRF T_{vg} is a constant but unknown matrix. We perform a set of static gestures to acquire multiple T_{wg}, and with the known T_{cv}, we jointly determine T_{wc} and T_{vg}. Finally, we apply T_{wc} to T_{cv} to get T_{wv}.

T_{wg} can be acquired via its inverse, $T_{wg} = T_{gw}^{-1}$. For T_{gw}, the three axes of GRF represented in WRF, $\overrightarrow{x_{gw}}$, $\overrightarrow{y_{gw}}$, $\overrightarrow{z_{gw}}$, can all be determined when the watch is static. $\overrightarrow{z_{gw}}$ is the opposite direction of gravity $\overrightarrow{Gr}$. $\vec{x}$ is the horizontal direction of Geo-North, which is perpendicular to $\overrightarrow{Gr}$. Thus $\overrightarrow{x_{gw}} = \frac{\vec{x'}}{\|\vec{x'}\|}$, and $\vec{x'} = \overrightarrow{Mr} - (\overrightarrow{Mr} \cdot \overrightarrow{z_{gw}})\overrightarrow{z_{gw}}$, where $\overrightarrow{Mr}$ is the magnetometer reading. The smartwatch we used applies the left-hand-rule, thus we also apply left-hand-rule to our reference frame, so that $\overrightarrow{y_{gw}} = \overrightarrow{x_{gw}} \times \overrightarrow{z_{gw}}$. The inverse of an orthogonal matrix is its transpose. Then we have

$$T_{gw} = \begin{bmatrix} \overrightarrow{x_{gw}} \\ \overrightarrow{y_{gw}} \\ \overrightarrow{z_{gw}} \end{bmatrix}, \quad T_{wg} = T_{gw}^{-1} = \begin{bmatrix} \overrightarrow{x_{gw}} \\ \overrightarrow{y_{gw}} \\ \overrightarrow{z_{gw}} \end{bmatrix}^{-1} = \begin{bmatrix} \overrightarrow{x_{gw}} \\ \overrightarrow{y_{gw}} \\ \overrightarrow{z_{gw}} \end{bmatrix}^{T} \tag{16}$$

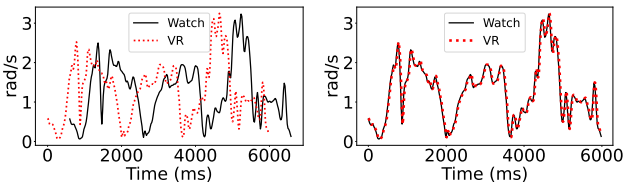
Recall that in $T_{wg} = T_{wc} \cdot T_{cv} \cdot T_{vg}$, both T_{wc} and T_{vg} do not change, and T_{wg} and T_{cv} are dynamic but can be acquired. Then we must determine the two constant matrices T_{wc} and T_{vg}. If we collect N static periods, we have N pairs of $(T_{wg}(i), T_{cv}(i))$. We define the calculation loss as:

$$Loss = \frac{\sum_{i=1}^{N} diff(T_{wg}(i), T_{wc} \cdot T_{cv}(i) \cdot T_{vg})}{N} \tag{17}$$

where $diff(T_x, T_y)$ returns the minimum degree of rotation between T_x and T_y.

A rotation of reference transformation is a process of 3-DoF. We have two unknown rotations, meaning there are six variables that need to be determined. Fortunately, we know the VRF and the GRF share the same **'up'** direction, which reduces the DoF of T_{vg} to 1. This leaves 3+1=4 variables to be determined. We jointly determine T_{wc} and T_{vg} by searching for the best pair of them that returns the minimum calculation loss in Equation (17). The minimum loss is 0.15°. We then apply T_{wc} we get to every T_{cv} to calculate T_{wv}.

4.5.2 Location Label. We can obtain the location of the VR controller represented in VRF L_{c-VRF} from the VR system. To derive the location of the smartwatch L_{w-VRF} from L_{c-VRF} in VRF, we need to obtain the vector points from the center of controller to the center of smartwatch in VRF L_{wc-VRF}. Then $L_{w-VRF} = L_{c-VRF} + L_{wc-VRF}$. Since the relative positions of smartwatch and the VR controller to each other are static, the vector L_{wc-VRF} is constant. However, it is impossible to manually measure this vector.



(a) Before synchronization. (b) After synchronization.

Figure 6: Time synchronization of smartwatch and VR.

As shown in Figure 5(b), we know that the center of the smartwatch is the center of its body, but we cannot exactly pinpoint the center of the VR controller.

Fortunately, we derived the transformation from WRF to VRF T_{wv} in Section 4.5.1, if we can acquire the vector in WRF, L_{wc-WRF}, then $L_{wc-VRF} = L_{wc-VRF} \cdot T_{wv}$. We thus design an experiment to derive the vector in WRF. If the locations of the smartwatch and the VR controller in WRF are known, the vector in WRF can be derived. The locations of the VR controller in WRF can be obtained by transforming the locations of VR controller in the VRF, because the transformation between the two references are known in Section 4.5.1. However, the smartwatch does not provide any information about its own location in WRF. To eliminate the uncertainty of the location of the smartwatch, we do not move but rotate the center of the smartwatch along its axes, so that its location does not change. The location trace of the VR controller in VRF is a circle.

As shown in Figure 5(b), we set up a spinning platform, whose rotation axis is vertical. The platform rotates automatically with electric power. We carefully adjust the smartwatch to a certain posture so that the Z-axis of the smartwatch is exactly on the rotation axis of the spinning platform. In this situation, when we rotate the platform, the location of the smartwatch does not change, from the point of the watch (WRF), the location of the controller does not change, while the location trace of the controller in VRF is a circle. The center of the circle is on the Z-axis of the watch. As shown in Figure 5(c), the radius of the circle is the length of the projection of the vector L_{wc-WRF} on the X-Y plane in WRF. We use L_c^{X-Y} to denote the collected locations of the controller, and L_w^{X-Y} to denote the location of the smartwatch (center of the circle). Then in WRF, we acquire the vector from L_c^{X-Y} to L_w^{X-Y}, $L_{wc-WRF}^{X-Y} = L_w^{X-Y} - L_c^{X-Y}$, which is the projection of the vector on the X-Y in WRF.

With the same principle, we can get L_{wc-WRF}^{Y-Z} and L_{wc-WRF}^{X-Z}. Then we search for a vector L_{wc-WRF} in a small 3D space ($20cm^3$), whose projection on the Y-Z, X-Z and X-Y best matches L_{wc-WRF}^{Y-Z}, L_{wc-WRF}^{X-Z}, and L_{wc-WRF}^{X-Y}, respectively. L_{wc-WRF} will be the vector we want to derive in WRF. The best match loss is 0.14 cm. We then apply this vector to every data sample and acquire the true location of the watch in VRF:

$$L_{w-VRF} = L_{c-VRF} + L_{wc-VRF} = L_{c-VRF} + L_{wc-WRF} \cdot T_{wv} \quad (18)$$

4.5.3 *Time Synchronization of VR system and Smartwatch.*

Since the VR controller and the smartwatch are static to each other. Their angular movements relative to the world should always be the same. To find the time bias between these two systems, we make use of the magnitude of angular velocity measured by the VR system and the smartwatch, $\|\omega_{vr}\|$ and $\|\omega_{watch}\|$. $\|\omega_{vr}\|$ and $\|\omega_{watch}\|$ should match the most when the two systems are well synchronized. As shown in Figure 6 (a), Given the data sequences from the VR system and the smartwatch, we add a time compensation t_c on the timestamp of VR readings and find the t_c that returns the minimum difference between $\|\omega_{vr}(t + t_c)\|$ and $\|\omega_{watch}(t)\|$. Finally, we synchronize the VR system with the smartwatch by adding t_c to all of the VR readings. Figure 6 (b) demonstrates the synchronization results.

4.5.4 *Alignment of Time Series Data From the Two Systems.*

The smartwatch and the VR system output data at different timestamps with different frequencies. The smartwatch outputs data with an average interval of 20 ms. The three IMU sensors may output data at different timestamps. The VR system outputs data with an average interval of 15 ms. We need to align the four time series data from the two systems.

Assume we want a data sample from sensor S at time t, while the sensor does not have outputs at this specific timestamp. We search for the two nearest data samples t_i and t_{i+1} that enclose time t, $t_i < t < t_{i+1}$. Sensor S outputs $S(t_i)$ at t_i, and $S(t_{i+1})$ at t_{i+1}. We then perform a linear interpolation to acquire a virtual data sample $S(t)$ at time t , with t_i, t_{i+1}, $S(t_i)$ and $S(t_{i+1})$:

$$S(t) = \frac{S(t_{i+1}) \cdot (t - t_i) + S(t_i) \cdot (t_{i+1} - t)}{t_{i+1} - t_i} \quad (19)$$

We prepare a timeline with a fixed sampling frequency f, and perform linear interpolation at every wanted timestamp. We set f as 50Hz, which is close to the smartwatch's sampling frequency, and lower than VR system's frequency of 66 Hz. By this step, the data streams from the two systems are aligned.

5 IMPLEMENTATION

Data Collection. Data collection consists of IMU data from the smartwatch and ground truth data from the VR system.

To read the IMU data from the smartwatch, we develop and install an App into the watch. The *SensorManager* API in Android is used to read data from sensors. To collect the IMU readings, we establish a server based on Apache Tomcat and Eclipse. The application installed in the smartwatch capture the IMU sensor data of the smartwatch and sends the data automatically to the server via *http* by finding the IP address of the server when the application is running. The server connects to a MySQL database to store data. The application and data collection server are implemented in Java.

To read the orientation and location of the VR controller, we develop an App based on unity and install it to the VR system. Since the VR device has sufficient disk space, we save the data locally. The orientation and location readings of the VR controller are saved into a *CSV* file automatically created by the App when it is running.

Multitask Model. Our multitask model is implemented and trained by Keras in Python. Each BiLSTM layer has 32 units, and each FC layer has three units. The optimizer is RMSprop, with a learning rate of 0.0001. The training epoch is 100, the batch size is 128, and the number of time steps considered by the BiLSTM is 32. The computer we use to collect data and develop our framework is an Alienware Aurora R7, with a Intel Core i5 8400 CPU (6-core) and NVIDIA GeForce GTX 1070 GPU (8GB memory).

TensorFlow Lite model. To execute *RTAT* on smartphones, we convert the well-trained TensorFlow model on the desktop to a TensorFlow Lite model capable of inference on mobile devices

using the TensorFlow Lite Converter. To do that, we froze the model as a TensorFlow concrete function. This process fixes the input size, which is specified as a tensor. It also specifies a *tf.function* which can be saved into a *.pb* file, the preferred input format for the TensorFlow Lite Converter. Running the TensorFlow Lite Converter on the saved model returned a *FlatBuffer* file with a *.tflite* extension which can be imported and used on mobile devices running Android or iOS, as well as some embedded devices and microcontrollers.

6 EVALUATION

In this section, we introduce our experiment settings. We demonstrate the performance and performance decomposition of *RTAT*, the performance of different applications, and the system overhead.

6.1 Experiment Settings

Platform and Devices. The platform we used to do evaluation experiments is introduced in Section 5. As introduced in Section 3, the smartwatch we use is Fossil Gen 5. It includes an LSM6DSO 3D accelerometer + 3D gyroscope and an AK0991X magnetometer. Similar chips are in many other commercial devices, including Samsung Galaxy A52, Samsung Galaxy S22 Ultra, Samsung Galaxy Note 10, Xiaomi Mi 10T Pro, Xiaomi POCO X3 Pro, Oneplus 7, MiWatch, TicWatch Pro 3, Sony Xperia 1 III, Motorola moto g fast, etc. We still use the Meta Quest 2 VR system to provide the ground truth for the evaluation.

Evaluation Metrics. We use the following metrics to quantify the performance of *RTAT* in estimating orientation and location.

- 3D orientation error. It is measured as the minimum degree of rotation required to align the estimated orientation to the ground truth orientation.
- 3D location error. It is the Euclidean distance between the estimated location and the ground truth location.

Baselines for Orientation Estimation. We compare the orientation estimation error of *RTAT* to two baselines.

- MUSE [12]: The state-of-the-art conventional sensor fusion approach for estimating device orientation.
- IDOL [17]: IDOL is based on Extended Kalman Filter (EKF). The prediction model of its EKF uses an RNN to predict orientation. The measurement model of its EKF is based on gyroscope readings integration.

Baselines for Location Estimation. We compare the smartwatch's location estimation error of *RTAT* with two baselines.

- MUSE [12]: It estimates the location using Particle Filters.
- ArmTroi [3]: It estimates location using HMM. As stated in [3], ArmTroi provides real-time computation on smartphones when the sampling rate is 5 Hz, but it is less accurate than MUSE. We will compare the accuracy of *RTAT* with MUSE, and the latency of *RTAT* with ArmTroi.

Dataset. We collect data from five users (two females and three males) to train and test our model. We collect data at two places, the hallway and the room. At each place, we collect 50-minute data from each user. The dataset collected from the hallway includes 754,688 samples, and the dataset collected from the room includes 761,856 samples. We divide each dataset into train and test data at a ratio of 4:1. At each place, we use the data from all 5 users to train a general model and test its performance on all 5 users. The test data for *RTAT* and the baselines is the same. We also collect data from four new users (two females and two males) to test the model. The model has never seen their data during training. We collect 10-minute data (around 30000 samples) from each user in each place.

Dataset Collection Scenarios. The age of the nine users ranges from 21 to 32, i.e., 30, 28, 23, 26, 21, 32, 32, 22, and 22, respectively. Their height ranges from 158 cm to 182 cm, i.e., 168 cm, 170 cm, 182 cm, 177 cm, 158 cm, 175 cm, 165 cm, 179 cm, and 160 cm, respectively. We ask users to move their arms freely at their normal movement speed. They perform random arm gestures or daily gestures, including driving, drinking, writing, exercising, push and pull, drawing, and so on.

6.2 *RTAT* Performance

In this subsection, we compare the performance of *RTAT* to the baseline methods in orientation and location estimation. The performance of *RTAT* is evaluated from the overall performance, the performance over time, the performance of different users and the performance of new users.

6.2.1 Overall Performance. Figure 8 depicts the overall orientation and location estimation error at the two testing places. From Figure 8 (a), in the hallway, the orientation error of MUSE, IDOL and *RTAT* are 35.13°, 22.87° and 17.19°, respectively averaged over the whole test data. *RTAT* decreases the orientation error by 24.84% and 51.07% compared to IDOL and MUSE, respectively. In the room, the orientation error of MUSE, IDOL and *RTAT* are 24.66°, 16.86° and 12.67°, respectively on the whole test data. *RTAT* decreases the orientation error by 24.85% and 31.63% compared to IDOL and MUSE, respectively. All systems perform better in the room than the hallway as magnet deviation in the hallway is larger.

The neural network of IDOL outputs quaternions directly. The orientation error defined as the norm quaternion difference between its predicted quaternions and the ground truth quaternions. Although its orientation estimation error is not high, we found its predicted quaternions do not meet Equation (2). Because its loss design ignores this constraint. The neural network finds a way to minimize the loss but it does not realize the constraint.

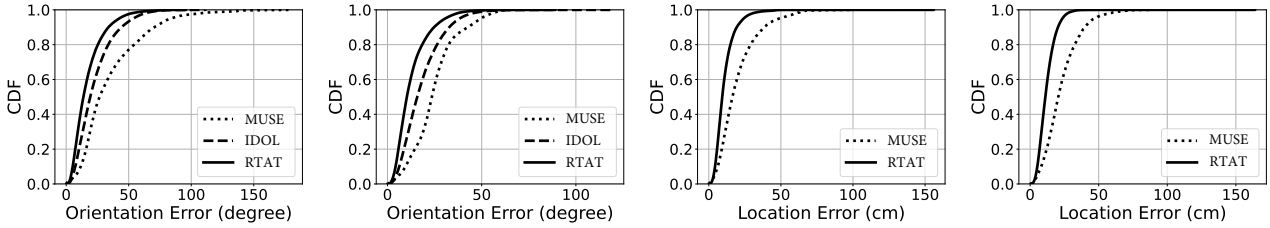
Figure 8 (b) shows the average location estimation error at the two different places. In the hallway, the location error of MUSE and *RTAT* are 19.87 cm and 10.93 cm, respectively. *RTAT* decreases the location error by 45% compared to MUSE. In the room, the location error of MUSE and *RTAT* are 22.75 cm and 12.09 cm, respectively. *RTAT* decreases the error by 46.9% compared to MUSE.

Figure 7 demonstrates the CDF of *RTAT*'s orientation and location error compared to the baselines at the two places. *RTAT* consistently outperforms the baselines for orientation and location estimation. From Figure 7 (a), for 80% cases, the orientation error of *RTAT* is less than 25 ° in the hallway. Similarly, Figure 7 (b) shows the orientation error of *RTAT* is less than 25 ° for 90% cases in the room. The CDF of location error at two places are similar, for 80%, the location error is smaller than 25 cm.

6.2.2 Performance Along with Time. Figure 9 shows the orientation and location error of *RTAT* and baselines along with time in the two places. Figure 9 is plotted based on 100-second data. From

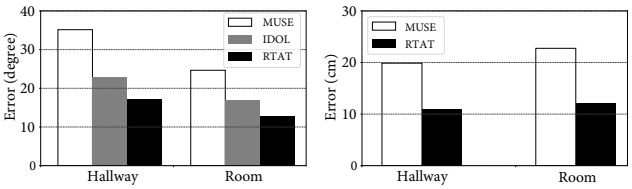
Table 3: Statistical analysis on users' test data

Hallway / Room	Gravity Error (degree)	Magnet Deviation (degree)	Static Moments	Gravity Opportunities (%)	Speed (m/s)
User1	8.83 / 9.54	13.34 / 5.69	85 / 71	5.92 / 4.74	0.55 / 0.56
User2	8.05 / 6.23	21.50 / 6.82	141 / 124	8.97 / 10.63	0.59 / 0.48
User3	7.15 / 5.37	45.41 / 5.10	165 / 216	10.52 / 13.3	0.39 / 0.30
User4	10.57 / 20.47	26.13 / 9.0	34 / 40	3.37 / 3.49	0.66 / 0.98
User5	7.69 / 9.56	26.61 / 7.15	75 / 17	5.62 / 2.61	0.42 / 0.57
Average(User1-5)	8.458 / 10.234	26.598 / 6.752	100 / 93.6	6.88 / 6.954	0.522 / 0.578
User6 (new)	7.21 / 4.54	24.71 / 5.91	110 / 351	8.22 / 30.77	0.61 / 0.29
User7 (new)	11.12 / 14.97	29.91 / 8.90	153 / 233	11.53 / 18.59	0.67 / 0.86
User8 (new)	12.61/10.25	26.15 /6.19	79/74	5.8/6.4	0.73 / 0.57
User9 (new)	23.92/16.94	36.08/9.10	17/5	3.0/4.5	1.27/0.78



(a) Hallway (b) Room (c) Hallway (d) Room

Figure 7: Orientation error and location error in hallway and room.



(a) Orientation error (b) Location error

Figure 8: Overall orientation and location error.

Figure 9 (a)-(d), we show *RTAT* performs better than MUSE and IDOL on orientation estimation and exhibits better location estimation than MUSE over time. *RTAT* is also more stable than the baseline orientation and location estimation methods.

6.2.3 Performance of Different Users. Figure 10 plots the orientation and location error of *RTAT* and the baseline methods on different users in the two sites. We collect both training and test data for *RTAT* from U1-U5, while *RTAT* has never seen the data of U6-U9 during training. For each user, *RTAT* exhibits better than MUSE and IDOL on orientation estimation and exhibits better than MUSE on location estimation. In comparison with MUSE, *RTAT* achieves more consistent performance across all users.

6.2.4 Performance of New Users. The U6-U9 in Figure 10 depicts the orientation and location of new users, whose data was omitted from the training phase.

For the new users, the performance of *RTAT* does not degrade much, indicating its applicability to new users. Moreover, *RTAT* still outperforms baselines at both orientation and location estimation for the four new users. Table 3 shows the statistical analysis on the users' test data at the two places. The motion speed of users will influence the prediction accuracy.

6.3 Performance Decomposition of *RTAT*

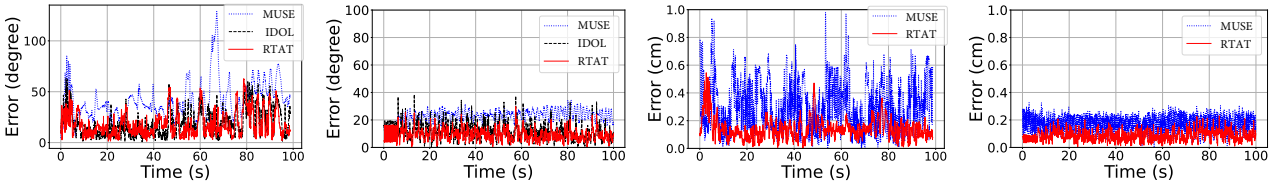
We test the performance gain provided by the three components of *RTAT* (i.e., multitask learning, attention and smooth loss) on the whole test data at the two places.

Multitask Learning. We first compare a multitask learning model against two single-task models, an orientation-only model and a location-only model. Both of the models are trained with the data from all three IMU sensors. Table 4 shows that while multitask learning decreases the orientation and location error by 9.1% and 21.8% in the hallway, it decreases the orientation and the location error by 1.7% and 5.5% in the room. Multitask learning reduces the model training time and the model inference overhead by half in comparison to implementing two single-task learning methods.

Attention Mechanism. We then compare *RTAT* _Multitask_Att to *RTAT* _Multitask. Table 4 demonstrates that the attention mechanism decreases the orientation and location error by 4.7% and 17.7% in the hallway. Compared to the hallway, the performance gain from the attention mechanism in the room is marginal. It only decreases the orientation and location error by 1.1% and 1.6% in the room. This is because the gravity error at these two places is very similar, but the magnetic field in the room is more stable than in the hallway. The attention mechanism contributes less in the room.

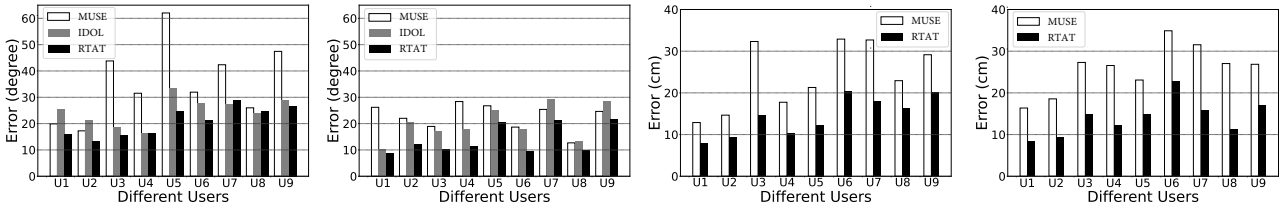
Smooth Loss. Finally, we show the benefits of smooth loss. The primary purpose of smooth loss is not to reduce the orientation and location error but to improve the smoothness of orientation and location tracking and makes posture tracking more realistic. We evaluate the orientation/location smoothness error by comparing the orientation/location difference of predicted ones to the orientation/location difference of ground truth between two consecutive timestamps. We also calculate the standard deviation of the orientation/location smoothness error.

From Table 4, smooth loss decreases orientation and location error by 5.5% and 7.8% in the hallway. It decreases orientation and



(a) Orientation at hallway (b) Orientation at room (c) Location at hallway (d) Location at room

Figure 9: Orientation error and location error along with time in hallway and room.



(a) Orientation at hallway (b) Orientation at room (c) Location at hallway (d) Location at room

Figure 10: Orientation and location error of different users in hallway and room.

Table 4: Average orientation and location estimation error of different models at hallway and room.

	Hallway		Room	
Model	Orientation Error (degree)	Location Error (cm)	Orientation Error (degree)	Location Error(cm)
RTAT _Orientation_Only	21.00	\	13.23	\
RTAT _Location_Only	\	18.40	\	13.44
RTAT _Multitask	19.09	14.39	13.00	12.70
RTAT _Multitask_Att	18.20	11.85	12.86	12.50
RTAT _Multitask_Att_Smooth	17.19	10.93	12.67	12.09

Table 5: Analysis on smoothness of orientation and location at two places.

	Hallway				Room			
	Orientation		Location		Orientation		Location	
Model	Error (degree)	Error_std	Error (degree)	Error_std	Error (degree)	Error_std	Error (degree)	Error_std
RTAT w/o Smooth Loss	1.28	2.91	1.58	2.43	1.30	2.68	1.43	2.0
RTAT w/ Smooth Loss	1.05	2.71	1.27	1.84	0.92	1.91	1.39	2.0

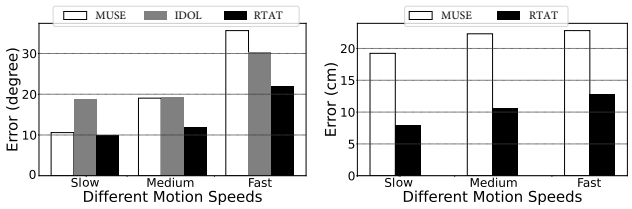
location error by 1.5% and 3.3% in the room. Table 5 shows the orientation/location smoothness error and the standard deviation of smoothness error at the two places. *RTAT* w/o Smooth Loss is the *RTAT* _Multitask_Att model. From Table 5, with smooth loss, *RTAT* improves the orientation smoothness and location smoothness by 18% and 19.6% in the hallway. *RTAT* improves the orientation smoothness and location smoothness by 29.2% and 2.8% in the room. The standard deviations of orientation/location smoothness error are also decreased under almost all scenarios.

6.4 Performance of Different Applications

Table 6: Statistic on different motion speed

Room	Speed (m/s)	Gravity Error (degree)	Magnet Deviation (degree)	Gravity Opportunities (%)
Slow	0.35	4.43	5.59	25.06
Medium	0.51	8.14	5.86	7.21
Fast	1.01	20.78	9.0	2.36

For different application scenarios, user motion speed may vary. When a user performs gym gestures, such as front raise and chest fly, motion is slow. When a user performs some daily gestures like making a call, pushing and pulling, the motion speed is moderate. When a user performs some AR games, motion is fast. We ask one user to perform gestures with different motion speeds in the room. Table 6 shows the statistical analysis. Figure 11 plots the orientation and location error with different motion speeds. As motion speed increases, the error of all systems increases. However, *RTAT* always performs better than the baseline systems, demonstrating its stability.



(a) Orientation performance. (b) Location performance.

Figure 11: Performance under different motion speeds at room.

6.5 System Overhead

6.5.1 Location latency on desktop. We evaluate the location latency of different systems on an Alienware Aurora R7 desktop. It takes MUSE, ArmTroi and *RTAT* 5427.05 ms, 2337.50 ms and 0.1633 ms respectively to process one-second data (50 samples). In MUSE [12] and ArmTroi [3], they use 10 Hz and 5 Hz respectively. *RTAT* is significantly faster than two conventional approaches, since they are based on searching algorithms. The deep learning architecture of *RTAT* is very lightweight.

6.5.2 Inference overhead on mobile devices. We also run *RTAT* on two commercial smartphones, Samsung S9 and Google Pixel3. We convert a well-trained TensorFlow model to a TensorFlow Lite model capable of inference on mobile devices by TensorFlow Lite Converter. Then we test the overhead of *RTAT* on the two smartphones. As shown in Table 7, the execution latency, memory usage and CPU usage of *RTAT* are low at both devices. It is even much faster than MUSE and ArmTroi running on the desktop.

6.5.3 Energy Consumption on Mobile Devices. We measure the energy consumption of *RTAT* on Samsung S9 by Monsoon monitor in Figure 12. The IMU data measured by the smartwatch is transmitted to the smartphone for processing via Bluetooth. We run the inference of *RTAT* for about 2 seconds, as indicated as the inference segment in Figure 12. The average working current (mA) in idle state with screen on is about 221 mA. The average working current on the *RTAT* inference state is about 357 mA. The average working current on playing music state is about 498 mA. *RTAT* only increases the working current by 136 mA, including both running the model and receiving IMU data via Bluetooth; whereas, playing music increases the current consumption by 277 mA. The power consumption of *RTAT* is less than half of playing music.

Similar to previous works [3, 5, 12], *RTAT* requires the smartwatch to continuously collect IMU sensor data and transmit the data to the smartphone while the system is in use. This process is power-consuming. Based on our experiments, it drains the watch's battery in around 3.5 hours. However, this limitation is not unique to our system. All the applications that require smartwatches to transmit IMU readings suffer from the same problem due to the limited battery power of smartwatches. We expect smartwatches will have more powerful batteries in the future.

Table 7: Inference overhead of *RTAT* on smartphones

	Samsung S9	Google Pixel3
Latency(ms)	2.98	1.82
Memory Usage (MB)	4.8	5.1
CPU Usage(%)	17	13

7 DISCUSSION

Data efficiency. We develop our system based on supervised learning. Collecting labeled data is labor-intensive. To train a generalized and robust model, the dataset should cover a wide variety of gestures, users, and environments. To reduce the labeled data, we may explore self-supervised training to capture temporal relations and feature distributions in IMU sensor measurements in the future [31].

Smartwatch-wearing angles. We indirectly study the different smartwatch-wearing angles in this work. Each user collects the data several times. We do not require the users to wear the smartwatch at the same angle. Even the same user may wear the smartwatch slightly differently each time. Different users also wear smartwatches at different angles. Our system is tolerant to wearing the smartwatch with slightly different rotation angles. Our data collection scenario is to accurately reflect how people wear a smartwatch in daily life, thus we do not consider wearing the watch with significantly different rotation angles on the wrist when we build the dataset. Inference accuracy drops when we apply the model to data collected from a smartwatch worn at significantly different rotation angles on the wrist. To accommodate this limitation, we may collect training data by wearing the watch with different rotation angles on the wrist, summarizing the data features with different rotation angles, and exploring domain shift in future work.

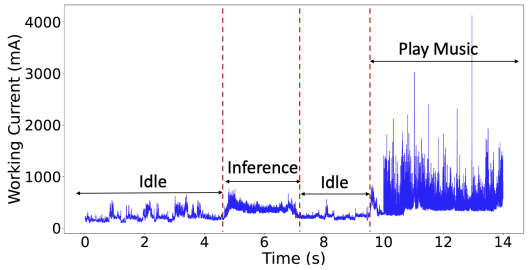


Figure 12: Energy Consumption on Samsung S9.

Generalization. It refers to two perspectives, i.e., the generalization across different users and the generalization across different places. The accuracy of our system drops slightly for new users because the data from new users have not been seen by the model during the training process. To improve the model generalization across users, training data could be collected from users with broader age and height ranges. To improve the model generalization across different places, transfer learning may be used to transfer the deep learning model learned by the data at one place to a new model for another place. We leave generalization as future work.

8 CONCLUSION

This paper presents *RTAT*, the first 3D human wrist tracking system via a smartwatch based on multitask deep learning. We design an attention mechanism and a smooth loss on top of the multitask learning network to improve its performance. *RTAT* is lightweight and supports real-time tracking on smartphones with high sampling frequency. We collect a large-scale dataset using our customized labeled data measurement system. Extensive experiments show *RTAT* achieves higher accuracy and lower latency when compared with baseline methods.

9 ACKNOWLEDGMENTS

We would like to thank our anonymous shepherd and reviewers for their constructive comments. This research was partially supported by the NSF grant #CCF-2008837, two Seed Fund Awards (2020 and 2022 respectively) from CITRIS and the Banatao Institute at the University of California (UC), and the UC National Laboratory Fees Research Program grant #69763.

REFERENCES

[1] Yash Jain, Chi Ian Tang, Chulhong Min, Fahim Kawsar, and Akhil Mathur. Collossl: Collaborative self-supervised learning for human activity recognition. *ACM IMWUT*, 6(1):1–28, 2022.

[2] Linlin Tu, Xiaomin Ouyang, Jiayu Zhou, Yuze He, and Guoliang Xing. Feddl: Federated learning via dynamic layer sharing for human activity recognition. In *ACM SenSys*, 2021.

[3] 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.

[4] Wenguang Mao, Mei Wang, Wei Sun, Lili Qiu, Swadhin Pradhan, and Yi-Chao Chen. Rnn-based room scale hand motion tracking. In *ACM MobiCom*, 2019.

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

[6] Peijun Zhao, Chris Xiaoxuan Lu, Bing Wang, Niki Trigoni, and Andrew Markham. Cubelearn: End-to-end learning for human motion recognition from raw mmwave radar signals. *arXiv preprint arXiv:2111.03976*, 2021.

[7] Chengkun Jiang, Yuan He, Songzhen Yang, Junchen Guo, and Yunhao Liu. 3d-omnitrack: 3d tracking with cots rfid systems. In *ACM/IEEE IPSN*, 2019.

[8] Hao Kong, Xiangyu Xu, Jiadi Yu, Qilin Chen, Chenguang Ma, Yingying Chen, Yi-Chao Chen, and Linghe Kong. m3track: mmwave-based multi-user 3d posture tracking. In *MobiSys*, 2022.

[9] Tianxing Li, Chuankai An, Zhao Tian, Andrew T Campbell, and Xia Zhou. Human sensing using visible light communication. In *ACM MobiCom*, 2015.

[10] Tianxing Li, Qiang Liu, and Xia Zhou. Practical human sensing in the light. In *ACM MobiSys*, 2016.

[11] Dongyao Chen, Mingke Wang, Chenxi He, Qing Luo, Yasha Iravantchi, Alanson Sample, Kang G Shin, and Xinbing Wang. Wearable, untethered hands tracking with passive magnets. In *ACM MobiCom*, 2021.

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

[13] Qiang Yang and Yuanqing Zheng. Model-based head orientation estimation for smart devices. *ACM IMWUT*, 5(3):1–24, 2021.

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

[15] 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*, 5(2):399–406, 2019.

[16] Martin Brossard, Silvere Bonnabel, and Axel Barrau. Denoising imu gyroscopes with deep learning for open-loop attitude estimation. *IEEE Robotics and Automation Letters*, 5(3):4796–4803, 2020.

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

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

[19] Xingzhou Zhang, Mu Qiao, Liangkai Liu, Yunfei Xu, and Weisong Shi. Collaborative cloud-edge computation for personalized driving behavior modeling. In *ACM/IEEE Symposium on Edge Computing*, 2019.

[20] Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347, 2021.

[21] Yuxin Tian, Xueqing Deng, Yi Zhu, and Shawn Newsam. Cross-time and orientation-invariant overhead image geolocalization using deep local features. In *IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020.

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

[23] Kang Yang and Wan Du. LLDPC: A Low-Density Parity-Check Coding Scheme for LoRa Networks. In *ACM SenSys*, 2022.

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

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

[26] Miaomiao Liu, Kang Yang, Yanjie Fu, Dapeng Oliver Wu, and Wan Du. Driving maneuver anomaly detection based on deep auto-encoder and geographical partitioning. *ACM Transactions on Sensor Networks (TOSN)*, 2022.

[27] Xianzhong Ding and Wan Du. DRLIC: Deep Reinforcement Learning for Irrigation Control. In *ACM/IEEE IPSN*, 2022.

[28] Zhihao Shen, Wan Du, Xi Zhao, and Jianhua Zou. DMM: Fast map matching for cellular data. In *ACM MobiCom*, 2020.

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

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

[31] 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.

[32] Andrea Giovanni Cutti, Andrea Giovanardi, Laura Rocchi, Angelo Davalli, and Rinaldo Sacchetti. Ambulatory measurement of shoulder and elbow kinematics through inertial and magnetic sensors. *Medical & biological engineering & computing*, 46(2):169–178, 2008.

[33] Mahmoud El-Gohary and James McNames. Shoulder and elbow joint angle tracking with inertial sensors. *IEEE Transactions on Biomedical Engineering*, 59(9):2635–2641, 2012.

[34] Qaiser Riaz, Guanhong Tao, Björn Krüger, and Andreas Weber. Motion reconstruction using very few accelerometers and ground contacts. *Graphical Models*, 79:23–38, 2015.

[35] Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. Motion reconstruction using sparse accelerometer data. *ACM Transactions on Graphics (ToG)*, 30(3):1–12, 2011.

[36] Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait? predicting bus arrival time with mobile phone based participatory sensing. In *ACM MobiSys*, 2012.

[37] Wan Du, Panrong Tong, and Mo Li. Uniloc: A unified mobile localization framework exploiting scheme diversity. *IEEE Transactions on Mobile Computing*, 20(7):2505–2517, 2020.

[38] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[39] Mingmin Zhao, Yonglong Tian, Hang Zhao, Mohammad Abu Alsheikh, Tianhong Li, Rumen Hristov, Zachary Kabelac, Dina Katabi, and Antonio Torralba. Rf-based 3d skeletons. In *SigComm*, 2018.

[40] Wenjun Jiang, Hongfei Xue, Chenglin Miao, Shiyang Wang, Sen Lin, Chong Tian, Srinivasan Murali, Haochen Hu, Zhi Sun, and Lu Su. Towards 3d human pose construction using wifi. In *ACM MobiCom*, 2020.

[41] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[42] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[43] Huimin Ren, Sijie Ruan, Yanhua Li, Jie Bao, Chuishi Meng, Ruiyuan Li, and Yu Zheng. Mtrajrec: Map-constrained trajectory recovery via seq2seq multi-task learning. In *ACM SIGKDD*, 2021.

[44] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649, 2013.

[45] Azure kinect, 2022. https://azure.microsoft.com/en-us/services/kinect-dk/.

[46] Vicon motion system, 2022. https://www.vicon.com/.

[47] Orientation tracking of azure kinect, 2019. https://github.com/microsoft/Azure-Kinect-Sensor-SDK/issues/654.

[48] Oculus insight, 2019. https://ai.facebook.com/blog/powered-by-ai-oculus-insight/.

[49] Ana Rojo, Javier Cortina, Cristina Sánchez, Eloy Urendes, Rodrigo García-Carmona, and Rafael Raya. Accuracy study of the oculus touch v2 versus inertial sensor for a single-axis rotation simulating the elbow's range of motion. *Virtual Reality*, pages 1–12, 2022.

[50] Valentin Holzwarth, Joy Gisler, Christian Hirt, and Andreas Kunz. Comparing the accuracy and precision of steamvr tracking 2.0 and oculus quest 2 in a room scale setup. In *2021 the 5th International Conference on Virtual and Augmented Reality Simulations*, pages 42–46, 2021.

[51] Tyler A Jost, Bradley Nelson, and Jonathan Rylander. Quantitative analysis of the oculus rift s in controlled movement. *Disability and Rehabilitation: Assistive Technology*, 16(6):632–636, 2021.