Online Path Description Learning based on IMU Signals from IoT Devices

Weipeng Zhuo, Shiju Li, Tianlang He, Mengyun Liu, S.-H. Gary Chan, *Senior Member, IEEE*, Sangtae Ha, *Senior Member, IEEE* and Chul-Ho Lee, *Member, IEEE*

Abstract—A user's movement path can be precisely and concisely described as a concatenation of straight lines having the user's turns as their end points. Learning such a path description or representation from inertial measurement unit (IMU) sensors enables various mobile and IoT applications, as it allows efficient processing of the movement path data. It is, however, non-trivial to learn a succinct yet accurate path description from IMU sensor readings in the mobile device of a moving user *on the fly* due to the dynamically changing behaviors and the technical difficulty in detecting the user's turns. We propose PATHLIT, a novel online path description learning system based on IMU signals. PATHLIT learns position vectors of a user from IMU sensor readings by our custom-made self-attention network model. Once each position vector is learned, PATHLIT also decides whether or not to take it as a part of the resulting path description by our efficient online algorithm developed under the minimum description length principle, which essentially detects the user's turns along the path. We conduct extensive experiments on two large datasets. The experiment results show that PATHLIT achieves superior performance over state-of-the-art algorithms by up to 50% in absolute trajectory error using only 15% of trajectory data points.

Index Terms—IMU, path recovery, online turn detection, minimum description length

1 Introduction

A movement path of a user in the two-dimensional space can be *succinctly* described by straight lines interspersed with the user's turns, which we refer to as a *path description*, since people usually do not walk randomly. It is important to learn the path descriptions efficiently "on the fly" from the readings of inertial measurement unit (IMU) sensors in users' internet of things (IoT) devices. For instance, path descriptions can be leveraged for real-time applications such as augmented and virtual reality applications [1], [2]. They can also be used to enable smart city applications *at scale*, such as indoor pathway learning [3], indoor navigation [4], and robot cleaning [5], due to their succinct representations of movement paths that allow efficient processing, storage, and transmission of the path data.

It is, however, challenging to learn such a path description since we need to recover its movement path and detect its associated user turns accurately on the fly. When recovering the movement path of a user from IMU readings, a small change in the user's walking behavior, referred to as *context*, can lead to substantial changes in the sensor readings. For example, the walking context changes, when a moving user with a phone in hand encounters and greets someone by waving the hand holding the phone, as shown in Figure 1. Thus, such a small context change makes accurate path recovery *non-trivial*. This path recovery problem has been studied in the literature, but existing solutions still have their own limitations.

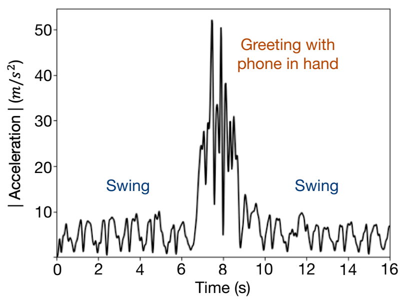


Fig. 1. Acceleration signals from a phone held in hand while a user is walking.

Earlier studies [6]–[8] focus on learning a simple linear model for step length estimation from IMU readings since it boils down to the step length estimation, assuming the direction of each step can be estimated accurately. Observing that the model parameters are specific to walking contexts, a few later studies build different linear models for different walking contexts and use an appropriate model by classifying the current walking context [9], [10]. They, however, require a non-trivial process of collecting IMU readings for

Weipeng Zhuo is with Guangdong Provincial Key Laboratory IRADS and Department of Computer Science, BNU-HKBU United International College, Zhuhai, China (email: weipengzhuo@uic.edu.cn).
Shiju Li is with the Department of Computer Engineering and Sciences, Florida Institute of Technology, Melbourne, Florida 32901, United States (email: sli2015@my.fit.edu).
Tianlang He, and S.-H. Gary Chan are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China (emails: {theaf, gchan}@cse.ust.hk).
Mengyun Liu is with the Institute of Artificial Intelligence, Guangzhou University, Guangzhou, China. (email: amylmy@gzhu.edu.cn)
Sangtae Ha is with the Department of Computer Science, University of Colorado Boulder, Boulder, Colorado 80309, United States (email: sangtae.ha@colorado.edu).
Chul-Ho Lee is with the Department of Computer Science, Texas State University, San Marcos, Texas 78666, United States (email: chulho.lee@txstate.edu).
Part of this work was done while Shiju Li was with Texas State University.

different walking contexts (with manual context labelling). In addition, other recent studies [11]–[13] leverage a long short-term memory (LSTM) model to recover a movement path by obtaining a sequence of estimated displacement or velocity vectors, without requiring any context classification. Nonetheless, while the rationale behind the use of the LSTM model is that the patterns in the IMU readings would have strong temporal correlations, context changes in users' walking behaviors could make their IMU readings *little* correlated.

On the other hand, when the movement path of a moving user is recovered from a stream of IMU readings, it can be wasteful and costly to store all the positional information for the recovered path, which is a set of position vectors (or coordinates of points along the path) that grows and expands over time. This problem can be even more critical when it comes to IoT devices with limited storage space. Thus, it is desirable to identify, *in real time*, which position vectors are crucial for a succinct yet accurate path description to represent the movement path that is being recovered. It boils down to the problem of detecting the user's *turning* points on the path on the fly, but it remains *largely unsolved* in the literature.

Prior studies [14]–[16] on the turn detection problem generally focus on the *offline* scenario where user turns are detected once the whole path information is available, i.e., after the user's movement is complete. Thus, they cannot be used for detecting turns on the fly for real-time applications. In addition, a *thresholding* method could be used to detect the user's turns by assuming that the user makes a turn if the directional change is above a predefined threshold [17]. However, such a method requires careful calibration of the threshold value, and it is also prone to errors when the IMU readings are noisy. Others [4], [18] leverage indoor maps for turn detection. It is, however, impractical to require an indoor map for every indoor setting.

In this paper, we propose PATHLIT, a novel online **PATH** description **L**earning system based on **I**MU signals from Io**T** devices. PATHLIT learns a path description from a stream of IMU readings by solving the problems of recovering a user's movement path and detecting the user's turns from the path *simultaneously* and *on the fly*. Here the resulting path description is a sequence of position vectors for *turning* points. The salient features of PATHLIT are that it is context-agnostic in the sense that it does not require context classification or prediction, and the path description is obtained in a principled manner by optimizing the tradeoff between the *preciseness* and *conciseness* of its representation without any predefined parameter.

PATHLIT first uses a multi-head self-attention network model which is tailor-made to effectively learn a user's movement path from a stream of IMU readings *without* context inference. The rationale behind the design of this model is to capture short-term correlations within IMU signals rather than their long-term correlations that have been mainly explored in the prior work [11]–[13], since walking context changes make the IMU signals less correlated in the long term, *yet* in an arbitrary manner. The IMU readings are first divided into short sequences of equal length. These sequences are then continuously fed into the model to learn their corresponding velocity vectors, which are then converted into displacement vectors and, eventually, position vectors.

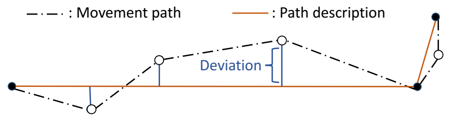


Fig. 2. Movement path vs. path description.

While a concatenation of the learned position vectors represents the user's movement path, it would not be a *succinct* representation. Thus, whenever a new position vector is learned by the self-attention network model, PATHLIT next decides whether to keep this position vector as a turning point or discard it, leading to a succinct and accurate path description that consists of the position vectors chosen as turning points. This is done by our *online* turn detection algorithm, which is developed under the minimum description length (MDL) principle [19]. We empirically demonstrate that this online algorithm not only results in a compact path description but also improves the accuracy of the recovered movement path.

A path description is considered to be *precise* or have *high fidelity*, when it contains all crucial user turns (and possibly a few extra ones) that can recover the path without much deviation. It is also considered to be *concise* or have *low complexity*, when it contains as few turns as possible, possibly less than the number of true turns. Our goal here is to find a path description that strikes a balance between its preciseness and conciseness. We thus leverage the MDL principle, which is to find the best (yet unknown) model, i.e., the best path description, that optimizes the tradeoff between the model's complexity, i.e., path description length, and fidelity, i.e., deviations of the path description from the path. See Figure 2 for an illustration.

However, the MDL principle is not a method, which means that it does not provide how to obtain the optimal model and neither does it provide an explicit problem formulation. Thus, we first introduce a notion of *MDL cost* to define the complexity and fidelity of a model so that the *optimal* model can be properly defined under the MDL principle. Because it still remains unknown how to obtain the optimal model, we formulate the problem as an MDL cost minimization problem. We then formally establish that its *offline* optimal solution can be obtained by solving an equivalent shortest-path problem on a weighted directed acyclic graph when the whole path information is available. We finally present MET, our **M**DL-based onlin**e** **t**urn detection algorithm. It is an efficient *online* algorithm of time complexity $O(\alpha_{\max}N)$ that allows us to find a succinct yet accurate path description *on the fly* based only on the path recovered so far, where N is the total number of points (position vectors) on a movement path, and $\alpha_{\max}$ is the largest number of points between two consecutive turning points detected.

Our contributions can be summarized as follows:

- *Context-agnostic path recovery*: We develop a multi-head self-attention network model to recover users' movement paths from IMU readings in their mobile devices while being agnostic to how they carry the devices. The model is judiciously customized to capture short-term correlations in the IMU readings for accurate path recovery.

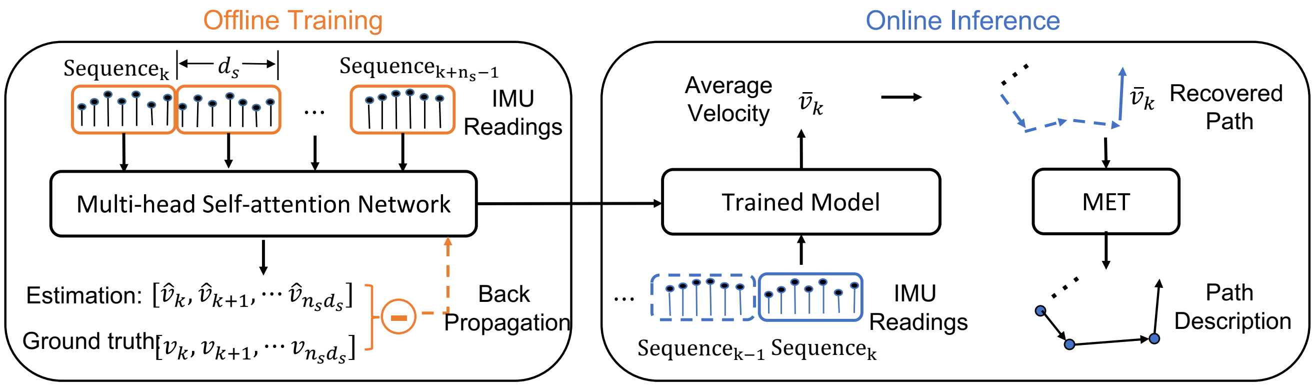

Offline Training
Online Inference
Sequence_k
d_s
$\text{Sequence}_{k+n_s-1}$
IMU Readings
Multi-head Self-attention Network
Estimation: $[\hat{v}_k, \hat{v}_{k+1}, \cdots \hat{v}_{n_s d_s}]$
Ground truth $[v_k, v_{k+1}, \cdots v_{n_s d_s}]$
Back Propagation
Average Velocity $\bar{v}_k$
Trained Model
$\bar{v}_k$ Recovered Path
MET
IMU Readings
Sequence_{k-1} Sequence_k
Path Description

Fig. 3. System diagram of PATHLIT.

- *Novel turn detection algorithm*: We demonstrate that the turn detection problem under the MDL principle can be solved by solving its equivalent shortest-path problem on a weighted directed acyclic graph. This problem equivalence allows us to obtain an (offline) optimal solution when the complete path information is available. In addition, as an integral component of PATHLIT, we develop an efficient online algorithm named MET to detect turning points on the fly based only on the path recovered so far, without requiring any parameterization or calibration of threshold values.
- *Extensive experiments*: We validate the effectiveness of PATHLIT on the RoNIN open dataset and our campus dataset. The datasets contain sequential IMU signals collected for a wide range of path trajectories while having different walking contexts, such as devices being used for messaging or taking photos and devices being in bags or pockets. Experiment results show that PATHLIT achieves high accuracy in path recovery and outperforms state-of-the-art algorithms significantly (by up to 50% in absolute trajectory error while just maintaining around 15% of the total location data points). Furthermore, we discuss the feasibility of MET for trajectory compression by showing its superior performance over state-of-the-art compression algorithms on the Microsoft GeoLife dataset (by up to 25% in absolute trajectory error).

The rest of this paper is organized as follows. We provide a system overview of PATHLIT in Section 2. We then present how to recover user paths in Section 3. We elaborate on the turn detection problem under the MDL principle and our turn detection algorithm MET in Section 4. We next present illustrative experiment results in Section 5. We review the related work in Section 6 and further discuss how to incorporate measurement data from other sensors and leverage more advanced orientation estimation techniques in Section 7. Finally, we conclude in Section 8.

2 System Overview

When a user is moving, six IMU sensor readings (three from an accelerometer and another three from a gyroscope) are collected in the user's mobile device at a given sampling frequency. Note that the IMU readings cannot be used as they are, as they depend on the coordinate system of the mobile device, whose orientation keeps on changing over time. The device coordinate system is defined relative to the device's screen, and the IMU readings are collected with respect to this device coordinate system, which can change due to the orientation changes. Thus, they are always transformed into the global coordinate system, which is aligned based on gravity and standard magnetic orientation and used as a reference coordinate system.[1]

Given a set of six IMU sensor readings, PATHLIT first recovers the corresponding segment of the movement path of the user via a multi-head self-attention network model. Specifically, it infers the velocity vector (speed and direction) of the segment from which the coordinates of the ending point of the path segment are obtained. PATHLIT then decides whether (or not) to keep the coordinates via our MDL-based online algorithm MET. Thus, we obtain a *path description*, which is a collection of the coordinates of the points along the path that are considered 'turning' points. These operations in PATHLIT are done *on the fly* for every set of six IMU readings.

While the details of our multi-head self-attention network model shall be explained in Section 3, the model is trained *offline* as follows. Given a movement path, or more specifically, a stream of six IMU readings collected during the path trajectory, it is first divided into smaller sequences of equal length, each of which is associated with its (ground truth) velocity vectors. We set each sequence to a two-second time window in this work, while we also discuss the impact of different sequence lengths on PATHLIT's performance in Section 5. The entire set of sequences are then all taken into the model in parallel instead of being taken sequentially. This way the model is able to capture correlations between the signal patterns that appear in different sequences, which are not necessarily right next to each other, as it is one of the salient features of self-attention networks compared to recurrent neural networks. The output of the model is a set of estimated velocity vectors for the given set of sequences per movement path. Thus, the model is built in a way that minimizes the difference between the ground-truth velocity vectors and the estimated velocity vectors.

As the output of its *online* inference, PATHLIT generates

1. Note that the rotation of the device along x-, y-, z-axis is measured by pitch, roll, and azimuth, respectively, which are available information in most mobile devices and used for the transformation.

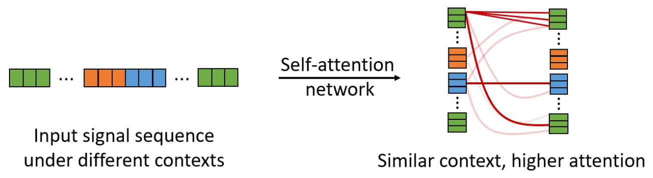


Fig. 4. Illustrating the attention mechanism on a sequence of signals under different contexts. Thicker lines indicate higher attention.

an estimated velocity vector for a segment of the path, which is then converted to the coordinates of the ending point of the segment. PATHLIT next uses its turn detection algorithm MET to decide whether to keep the coordinates. This online algorithm is developed as an online counterpart of the offline optimal algorithm to detect optimal turning points along the path. Assuming that the whole (estimated) trajectory information is available, we formulate a *turn detection* problem from the MDL principle and formally demonstrate that it is equivalent to solving a shortest-path problem on a weighted directed acyclic graph, which naturally leads to the offline optimal algorithm. The details of the offline optimal algorithm and its online counterpart MET shall be explained in Section 4. Figure 3 summarizes the overall system architecture of PATHLIT.

3 Context-Agnostic Path Recovery

In this section, we explain the details of our multi-head self-attention network model to infer the movement path of a user from the IMU signals in the user's mobile device. While the multi-head self-attention network architecture was proposed in [20], it was originally developed for NLP. We first provide a brief introduction to the self-attention network and then explain how a stream of IMU signals, which are time-series data, are leveraged along with the self-attention network model for the movement path recovery.

3.1 Preliminaries on Self-attention Networks

The self-attention network takes in a sequence of inputs and results in their corresponding output sequence. For instance, in the translation task in NLP, it takes in a sequence of words in one language and translates them into a sequence of words in another language. The self-attention network commonly has an encoder-decoder architecture. Taking the translation task as an example, the encoder first processes the input sequence of words to learn the attention weights, or correlations, between each pair of the words. The attention weights are then shared with the decoder layers. In the decoder, the embedding of a token that corresponds to a word is taken as an input at a time. Together with the learned attention weights from the encoder, the embedding is used to predict the next word in another language until the translation is done or a predefined length is reached. Details on the encoder-decoder architecture used in PATHLIT shall be presented in Section 3.3.

In PATHLIT, a sequence of IMU signals are taken into the self-attention network to generate a sequence of their corresponding velocity vectors such that the user path can be recovered accurately. The sequence of IMU signals taken into the network might be obtained while under different contexts, as shown in Figure 4, where the signals under different contexts are denoted with different colors. Note that a window of IMU signals for a short period of time can be thought of as a word in the translation task. The network is then able to learn attention weights (or pairwise correlation) between each pair of signal windows, which allow us to capture which ones are similar to each other. The signal windows under similar contexts would have higher attention scores and thus lead to similar velocity vectors. Hence, we leverage the attention mechanism in PATHLIT to infer the velocity vectors under different contexts automatically, without manually specifying the contexts beforehand or afterwards.

There are two main advantages of using a self-attention network over a recurrent neural network network (RNN) for our problem. First, the computation in the self-attention network is done in parallel for a sequence of signals, making it much more efficient compared with RNN which needs to process the signals one by one in a sequential order. Second, the self-attention network allows us to focus on signals under similar contexts in predicting velocity vectors without attending to other patterns, which improves the model performance substantially. However, in RNN, as the signals have to be processed in order, the outputs of RNN could be influenced by the earlier parts of signals that might have been under distinct patterns, thereby possibly leading to unsatisfactory prediction performance.

3.2 Network Inputs and Outputs

Recall that the IMU sensors in a mobile device are an accelerometer and a gyroscope, each of which has three axes, namely x-, y- and z-axis. Given a sampling rate, the two sensors generate a total of six sensor readings at each sampling time. Let $\boldsymbol{s}_1, \boldsymbol{s}_2$, and $\boldsymbol{s}_3$ be three d-dimensional column vectors to represent streams of the accelerometer readings (or samples) along the x-, y-, and z-axis, respectively, obtained while a user is moving. Similarly, we define $\boldsymbol{s}_4, \boldsymbol{s}_5$, and $\boldsymbol{s}_6$ for the streams of the three-axis readings from the gyroscope. Here the dimension d of each vector $\boldsymbol{s}_i$ is the total number of readings during a user's path trajectory, which is the sampling rate times the total travel time by the user for the trajectory.

As shown in Figure 5, we first construct a $d \times 6$ signal matrix $\mathbf{S} := [\boldsymbol{s}_1 \ \boldsymbol{s}_2 \ \ldots \ \boldsymbol{s}_6]$ as a horizontal concatenation of the six vectors. In other words, this signal matrix $\mathbf{S}$ represents a stream of six IMU sensor readings. We then divide the signal matrix $\mathbf{S}$ into two-second sequence matrices of size $d_s \times 6$, where d_s is the number of sensor readings collected for two seconds along each sensor axis. In other words, the signal matrix $\mathbf{S}$ is divided into $\lfloor d/d_s \rfloor$ sequence matrices. Let $\mathbf{S}_k := [\boldsymbol{s}_1^k \ \boldsymbol{s}_2^k \ \ldots \ \boldsymbol{s}_6^k] \in \mathbb{R}^{d_s \times 6}$ be the k-th sequence matrix, where $\boldsymbol{s}_i^k$ indicates the k-th d_s-dimensional sequence vector obtained from $\boldsymbol{s}_i$ for $i = 1, 2, \ldots, 6$ and $k = 1, 2, \ldots, \lfloor d/d_s \rfloor$.

Fix k. We next obtain an embedding matrix of size $d_s \times d_{em}$ from each sequence matrix $\mathbf{S}_k$ via a linear transformation with a learnable weight matrix $\mathbf{W}_S \in \mathbb{R}^{6 \times d_{em}}$. We also introduce *positional encodings* to account for the order of the embeddings obtained from the collections of six sensor readings. Each position encoding is a representation of the position of each collection of six sensor readings among the d_s collections. The dimension of this representation is the same as that of each embedding. Specifically, for each sequence matrix $\mathbf{S}_k$, we define the following positional

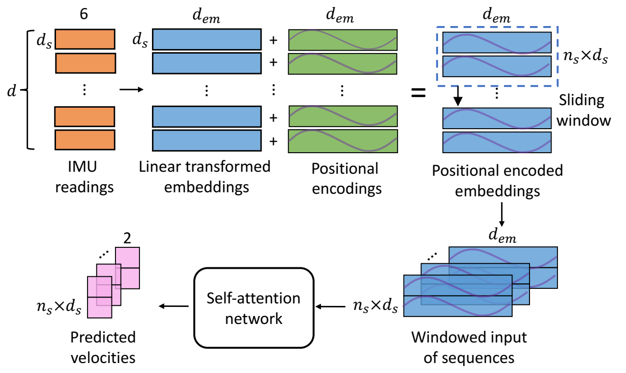


Fig. 5. Network inputs and outputs.

encoding matrix $\mathbf{P} \in \mathbb{R}^{d_s \times d_{em}}$ to encode the positions of rows in $\mathbf{S}_k$, where each row corresponds to a collection of six sensor readings at a sampling time. Each entry of $\mathbf{P}$ is given by

$$\mathbf{P}_{(i,2j)} = \sin(i/f^{2j/d_{em}}), \text{ and } \mathbf{P}_{(i,2j+1)} = \cos(i/f^{2j/d_{em}}),$$

where j is the index of the j-th dimension with $0 \leq j < d_{em}/2$, and f is some constant to control the cyclic pattern of each sinusoidal function, which is set to 10000 as in [20]. To summarize, we first have a $d \times 6$ signal matrix $\mathbf{S}$ from the streams of six sensor readings obtained during a user's path trajectory and divide $\mathbf{S}$ into $d_s \times 6$ sequence matrices. Then, from each sequence matrix $\mathbf{S}_k$, we finally have a $d_s \times d_{em}$ input embedding matrix $\mathbf{\Lambda}_k := \mathbf{S}_k\mathbf{W}_S + \mathbf{P}$, which is obtained by a linear transformation with $\mathbf{W}_S$ and then by incorporating the positional encoding matrix $\mathbf{P}$.

While each input embedding matrix $\mathbf{\Lambda}_k$ can be used as an input into the self-attention network model, we use a sliding window of n_s input embedding matrices as an input. The rationale behind this is to learn a better representation by capturing possible correlations between IMU signal patterns over a longer time span, i.e., $2n_s$ seconds, while still limiting to each two-second sequence for *inference*. We use $n_s = 5$ in this work. Let $\mathbf{U}_k$ be the k-th sliding window of n_s input embedding matrices, which is defined as a vertical concatenation of the n_s matrices, i.e., $\mathbf{U}_k := [\mathbf{\Lambda}_{k-n_s+1}; \mathbf{\Lambda}_{k-n_s+2}; \ldots; \mathbf{\Lambda}_k] \in \mathbb{R}^{n_s d_s \times d_{em}}$, where ';' indicates the vertical concatenation of the matrices. Note that the first $n_s - 1$ sliding windows are constructed with zero-matrix padding to match the dimension.

In addition, for a collection of six sensor readings at each sampling time, the self-attention network model outputs its corresponding estimated velocity vector $\hat{\boldsymbol{v}} := [\hat{v}_x, \hat{v}_y]$ in the two-dimensional space. In other words, for each input $\mathbf{U}_k$, which is a sliding window of n_s input embedding matrices, it outputs a collection of estimated velocity vectors, i.e.,

$$\hat{\mathbf{V}}_k := [\hat{\boldsymbol{v}}_{(k-n_s)d_s+1}; \hat{\boldsymbol{v}}_{(k-n_s)d_s+2}; \ldots; \hat{\boldsymbol{v}}_{kd_s}] \in \mathbb{R}^{n_s d_s \times 2}.$$

On the other hand, for model training, we have a ground truth velocity $\boldsymbol{v}_i$ for the i-th collection of six sensor readings, which is obtained by calculating the difference of two position vectors of the user at two seconds apart, i.e.,

$$\boldsymbol{v}_i := \frac{\boldsymbol{x}_{i+d_s} - \boldsymbol{x}_i}{2}, \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+d_s}$ denote the location vectors of the user at the i-th sampling time and the $(i + d_s)$-th sampling time (two seconds later), respectively. Letting $\mathbf{V}_k := [\boldsymbol{v}_{(k-n_s)d_s+1}; \boldsymbol{v}_{(k-n_s)d_s+2}; \ldots; \boldsymbol{v}_{kd_s}]$, we use the following

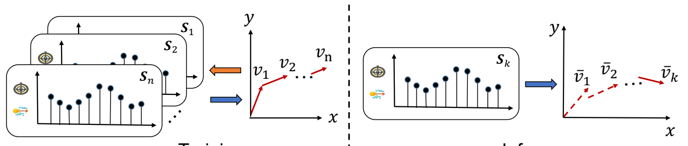


Fig. 6. Training and inference of our self-attention network model.

loss function for model training:

$$\text{loss} := \sum_k \|\mathbf{V}_k - \hat{\mathbf{V}}_k\|_F, \tag{2}$$

where $\|\cdot\|_F$ is the Frobenius norm. Note that all the velocity vectors with negative indexes are again padded with zero vectors for both ground truth and estimated ones. Figure 3 depicts a summary of the overall operation explained above.

When it comes to inference, we use the origin $(0, 0)$ as the starting position $\hat{\boldsymbol{x}}_0$. Whenever a new two-second sequence of six sensor readings, i.e., S_k, is available, we construct an input matrix $\mathbf{U}_k$ with historical data (padded with zeros if there are not enough historical data), which leads to its corresponding estimated velocity matrix $\hat{\mathbf{V}}_k$. From $\hat{\mathbf{V}}_k$, we only take a new collection of the estimated velocity vectors $[\hat{\boldsymbol{v}}_{(k-1)d_s+1}; \hat{\boldsymbol{v}}_{(k-1)d_s+2}; \ldots; \hat{\boldsymbol{v}}_{kd_s}]$ that correspond to the new two-second (input) sequence S_k. By noting that the velocities do not change much within a two-second window, we use the average of the estimated velocity vectors in inferring the position of the user, which is given by

$$\bar{\boldsymbol{v}}_k := \frac{1}{d_s}\sum_{i=1}^{d_s} \hat{\boldsymbol{v}}_{(k-1)d_s+i}. \tag{3}$$

An illustrative example of the training and inference processes is provided in Figure 6.

For the k-th input sequence S_k, the displacement vector that the user makes for two seconds can be estimated as $2\bar{\boldsymbol{v}}_k$. The k-th position vector $\hat{\boldsymbol{x}}_k$ can then be estimated as

$$\hat{\boldsymbol{x}}_k := \hat{\boldsymbol{x}}_{k-1} + 2\bar{\boldsymbol{v}}_k. \tag{4}$$

Therefore, the user's movement path recovered based on N two-second sequences of IMU signals is finally represented as $\{\hat{\boldsymbol{x}}_0, \hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_N\}$.

3.3 Network Structure

We develop a multi-head self-attention network model that has an encoder-decoder architecture and is adopted from [20]. We first consider its encoder structure. Since each input $\mathbf{U}_k$ is fed into the model independently and identically, we hereafter drop the subscript k and use $\mathbf{U}$ to denote an input matrix for brevity, unless otherwise noted.

Given an input matrix $\mathbf{U} \in \mathbb{R}^{n_s d_s \times d_{em}}$, we extract feature representations via linear transformations with three $d_{em} \times d_{em}$ learnable weight matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_M$. Specifically, we have the following $n_s d_s \times d_{em}$ matrices:

$$\mathbf{Q} := \mathbf{U}\mathbf{W}_Q, \;\; \mathbf{K} := \mathbf{U}\mathbf{W}_K, \;\; \text{and } \mathbf{M} := \mathbf{U}\mathbf{W}_M.$$

Here, rows of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{M}$ are called *queries*, *keys*, and *values*, respectively. We then obtain weighted sums of feature representations via the self-attention mechanism, which is defined by

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{M}) := \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d_{em}})\mathbf{M} \in \mathbb{R}^{n_s d_s \times d_{em}},$$

where the scaling factor $1/\sqrt{d_{em}}$ is introduced to avoid vanishing gradients, softmax$(\cdot)$ indicates a row-wise softmax normalization function, and T stands for the transpose

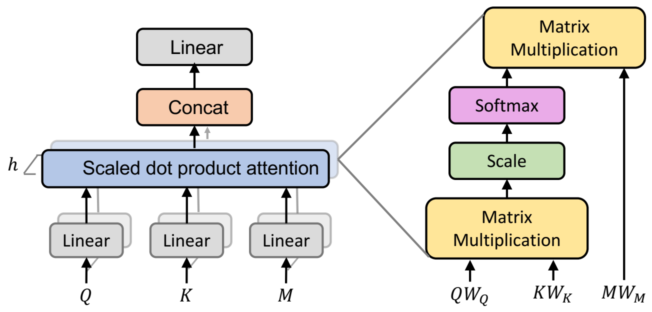


Fig. 7. Encoder of the multi-head attention network.

operation. Recall that each row of $\mathbf{U}$ corresponds to a collection of six IMU sensor readings at a sampling time. The scaled dot product between the feature representations of two different collections (a row of $\mathbf{Q}$ and another row of $\mathbf{K}$) indicates their similarity score. Thus, the self-attention leads to weighted sums of *value* feature representations with the weights proportional to the similarity scores.

To further improve the model performance, as shown in Figure 7, we adopt the multi-head attention mechanism to transform the input matrix $\mathbf{U}$ into multiple *representation subspaces*. The matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{M}$ are first linearly projected to h different subspaces. Each linear projection is done independently via the same self-attention mechanism (yet with different weight matrices). They are then concatenated horizontally and fed into a linear layer (i.e., a linear transformation), which outputs final $n_s d_s \times d_{em}$ representations of the input matrix $\mathbf{U}$. The multi-head attention mechanism is summarized as follows:

$$\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{M}) := [\text{head}_1\ \text{head}_2\ \dots\ \text{head}_h]\mathbf{W}_O,$$
$$\text{with head}_i := \text{Attention}(\mathbf{Q}\mathbf{W}^i_Q, \mathbf{K}\mathbf{W}^i_K, \mathbf{M}\mathbf{W}^i_M),$$

where $\mathbf{W}_O$ is a $d_{em} \times d_{em}$ learnable weight matrix, and $\mathbf{W}^i_Q$, $\mathbf{W}^i_K$, and $\mathbf{W}^i_M$ are $d_{em} \times d_{em}/h$ learnable weight matrices for $i = 1, \cdots, h$. This completes the operation of an encoder layer. In this work, we use a stack of encoder layers, where each of them has the same structure and takes the output of the previous layer as an input (except the first/bottom layer). The number of the encoder layers is a hyperparameter.

We next turn attention to the decoder structure. Since it is similar to the encoder structure, we here focus on its main structure. The decoder takes the output of the encoder, which is the final representations of $\mathbf{U}_k$, as an input and outputs its corresponding predicted velocity matrix $\hat{\mathbf{V}}_k$. The decoder has a self-attention layer followed by an encoder-decoder attention layer. The self-attention layer is the same as the one in the encoder. While the encoder-decoder attention layer also works based on the same self-attention mechanism, it takes in the queries $\mathbf{Q}$ from the self-attention layer and the keys $\mathbf{K}$ and the values $\mathbf{M}$ from (the last encoder layer of) the encoder. Since $\mathbf{K}$ and $\mathbf{M}$ are the key and value representations of the input matrix $\mathbf{U}_k$ (a sliding window of n_s input sequences), this allows the decoder to *attend to* all positions in the input sequences. As in the encoder, we use a stack of decoder layers, where the number of the decoder layers is a hyperparameter. The output of the last encoder layer finally goes through the final linear and softmax layers to obtain the velocity matrix $\hat{\mathbf{V}}_k$.

Note that the velocity matrix $\hat{\mathbf{V}}_k$ is used along with the ground-truth velocity vectors to calculate the loss in (2) and build our model that minimizes the loss. When it comes to inference, it is used to recover the user's movement path as in (3) and (4). The rest of the encoder-decoder architecture is the same as the one in [20].

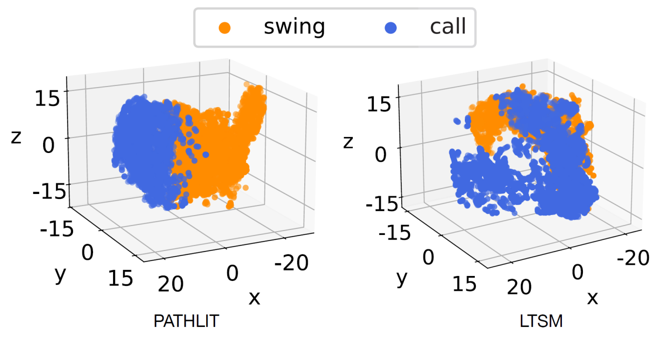


Fig. 8. Visualization of embeddings learned.

To demonstrate the effectiveness of our self-attention network model, we carry out a small experiment to visualize the embeddings learned from our model and LSTM [13]. In the experiment, a user is asked to walk along a path with a phone in the user's hand. While walking, the user first swings his arms and then makes a phone call. Once the call is done, he swings his arms again. Each posture lasts for about 20 seconds. As shown in Figure 8, our model is able to better separate the embeddings of signals under different contexts because the self-attention network focuses on learning patterns from the signals under the same context, whereas LSTM can hardly avoid the influence of historical signal data that were under a different pattern.

4 MDL-BASED TURN DETECTION

In this section, we first explain the preliminaries of the MDL principle. We then present the turn detection problem under the MDL principle, assuming that the whole path information is available. We formally establish its equivalence to a shortest-path problem on a weighted directed acyclic graph. We finally propose MET, an efficient greedy algorithm with linear time complexity for real-time turn detection.

4.1 Two-part MDL Principle

We first tackle the (offline) problem of detecting turning points along a recovered movement path so that the path is described by a concatenation of straight lines, whose end points are the detected turning points, in a concise manner, i.e., *low complexity*, while keeping the description as precise as possible, i.e., *high fidelity*. We assume, for now, that the set of position vectors for the movement path is available. We will later explain how this turn detection can be done on the fly while the movement path is being recovered.

Let $\boldsymbol{D} := \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ be a movement path recovered by the self-attention network model in PATHLIT, where $\boldsymbol{x}_i$ indicates the i-th position vector or the coordinates of the i-th point on the path, and N is the total number of the points along the path, except the origin $\boldsymbol{x}_0$. We here use $\boldsymbol{x}_i$ instead of $\hat{\boldsymbol{x}}_i$ for brevity. In addition, we refer to two end points of the path and turning points along the path as *path delimiters*, or simply, *delimiters*. Define a set of the indexes of the delimiters $\boldsymbol{B} := \{b_0, \cdots, b_m\}$ such that

$0 = b_0 < \cdots < b_m = N$ and $b_i \in \{0, 1, \cdots, N\}$. Note that $m = |\boldsymbol{B}| - 1$. Then, the problem here is to find the optimal *delimiter set* $\boldsymbol{B}^\star$ under the MDL principle.

The idea is to transform this problem into a model selection problem. Observe that $\boldsymbol{B}$ is the only unknown parameter of a model $f(\cdot|\boldsymbol{B})$, which specifies how we represent or describe a path by using the delimiter set $\boldsymbol{B}$. For example, $f(\boldsymbol{D}|\boldsymbol{B})$ represents path $\boldsymbol{D}$ by the delimiter set $\boldsymbol{B}$. In this work, we consider that $f(\cdot|\boldsymbol{B})$ is the sum of the Euclidean distances between two consecutive delimiters in $\boldsymbol{B}$. Then, letting $\mathcal{L} := \{1, 2, \ldots, N-1\}$, we define the class of the candidate models as $\boldsymbol{\mathcal{F}} := \{f(\cdot|\boldsymbol{B}) : \boldsymbol{B}\backslash\{0, N\} \subset \mathcal{L}\}$. For a path, there are 2^{N-1} models in total since each point along the path is either included or excluded in a model while the two end points of the path are always in the model. We hereafter use $\boldsymbol{B}$ to denote its corresponding model $f(\cdot|\boldsymbol{B})$ and $\boldsymbol{B}^\star$ to denote the optimal model $f(\cdot|\boldsymbol{B}^\star) \in \mathcal{F}$ for simplicity, unless a confusion exists.

For the model selection problem, Rissanen [21] suggests using the code length as a means to compare two parts of different models, namely *model complexity* and *model fidelity*. For a model, the former indicates how long the code length of this model is and the latter specifies how well this model encodes the data. Here the *best* model is the one that describes the data with the shortest code length (or the minimum description length). The key idea is to split the representation of data into two parts, i.e., the encoding length of a candidate fitted model and the encoding length of the data given the model, as follows. Letting $L(\text{“}data\text{”})$ be the encoding length of the data, we have

$$L(\text{“}data\text{”}) = L(\text{“}fitted\ model\text{”}) + L(\text{“}data\ given\ fitted\ model\text{”}),$$

where $L(\text{“}data\text{”})$ is also called the MDL cost. The best model is the one minimizing the MDL cost. This is the two-part MDL principle. Note that it still remains unknown how to obtain the best model.

In this work, we adopt this MDL principle to find the best model $\boldsymbol{B}^\star$ that achieves the shortest description of a movement path, which is the optimal solution to the original turn-detection problem. For a path $\boldsymbol{D}$, let $L(\boldsymbol{D})$ denote the MDL cost of $\boldsymbol{D}$. It can then be decomposed into

$$L(\boldsymbol{D}) = G(\boldsymbol{B}) + Z(\boldsymbol{D}|\boldsymbol{B}), \tag{5}$$

where $G(\boldsymbol{B})$ and $Z(\boldsymbol{D}|\boldsymbol{B})$ represent the code length of the model $\boldsymbol{B}$ that describes the path $\boldsymbol{D}$ and the one of the path $\boldsymbol{D}$ given the model $\boldsymbol{B}$, respectively. In other words, $G(\boldsymbol{B})$ indicates the complexity of the model $\boldsymbol{B}$, i.e., the sum of the distances between two consecutive delimiters in $\boldsymbol{B}$, and $Z(\boldsymbol{D}|\boldsymbol{B})$ indicates the fidelity of $\boldsymbol{B}$, i.e., how well the model $\boldsymbol{B}$ describes the path $\boldsymbol{D}$ by preserving the shape and orientation of the path.

4.2 MDL Cost for Turn Detection

We next show how exactly the turn detection problem is formulated as an MDL cost minimization problem. Note that the MDL cost needs to be defined explicitly as the MDL principle does not provide any such definition. Figure 9 provides an illustration of model complexity and model fidelity under our problem. On one hand, a concise (low complexity) path description requires as few delimiters as possible. The lowest complexity is achieved when the model only contains the starting and ending points of the path. On the other hand, a precise (high fidelity) description requires as many delimiters as possible. The maximal fidelity is achieved when the model contains the coordinates of all the points along the path as delimiters. There is clearly a tradeoff between complexity and fidelity, so it is desirable to achieve the optimal tradeoff in describing a path.

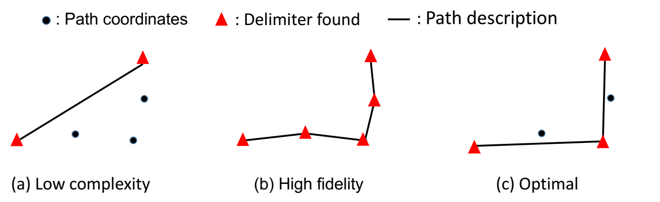


Fig. 9. Model complexity vs. model fidelity.

Consider a path $\boldsymbol{D}$. Let $len(\boldsymbol{x}_i, \boldsymbol{x}_j)$ be the length of a line segment with two end points being $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ for $0 \le i < j \le N$. We define the model complexity $G(\boldsymbol{B})$ as the sum of the lengths of the segments connected by two consecutive delimiters in $\boldsymbol{B}$, which is given by

$$G(\boldsymbol{B}) := \sum_{i=0}^{|\boldsymbol{B}|-1} \log_2(len(\boldsymbol{x}_{b_i}, \boldsymbol{x}_{b_{i+1}})). \tag{6}$$

For the most concise case, i.e., having the starting and ending points of the path $\boldsymbol{D}$ as the only delimiters, we can see that it minimizes $G(\boldsymbol{B})$. This is due to the triangle inequality.

To measure the model fidelity $Z(\boldsymbol{D}|\boldsymbol{B})$, we first define two types of distances, namely *angular distance* $d_\angle$ and *perpendicular distance* $d_\perp$. As depicted in Figure 10, suppose that we have two line segments $\boldsymbol{s}_i$ and $\boldsymbol{\eta}_j$, where $\boldsymbol{s}_i$ is the segment connected by two consecutive delimiters (e.g., $\boldsymbol{x}_0$ and $\boldsymbol{x}_4$ in the figure) and $\boldsymbol{\eta}_j$ is the segment formed by the coordinates of two consecutive points that appear between the two delimiters (e.g., $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ in the figure). Letting $len(\boldsymbol{\eta}_j)$ be the length of segment $\boldsymbol{\eta}_j$, we define the angular distance between $\boldsymbol{s}_i$ and $\boldsymbol{\eta}_j$ by

$$d_\angle(\boldsymbol{s}_i, \boldsymbol{\eta}_j) := len(\boldsymbol{\eta}_j) \cdot \sin(\min\{\theta_{ij}, 90^\circ\}),$$

where θ_{ij} is the angle between $\boldsymbol{s}_i$ and $\boldsymbol{\eta}_j$ in degrees. We also define the perpendicular distance between $\boldsymbol{s}_i$ and $\boldsymbol{\eta}_j$ by

$$d_\perp(\boldsymbol{s}_i, \boldsymbol{\eta}_j) := \frac{l_{\perp 1}^2(\boldsymbol{s}_i, \boldsymbol{\eta}_j) + l_{\perp 2}^2(\boldsymbol{s}_i, \boldsymbol{\eta}_j)}{l_{\perp 1}(\boldsymbol{s}_i, \boldsymbol{\eta}_j) + l_{\perp 2}(\boldsymbol{s}_i, \boldsymbol{\eta}_j)},$$

where $l_{\perp 1}(\boldsymbol{s}_i, \boldsymbol{\eta}_j)$ is the shortest distance from one end point of $\boldsymbol{\eta}_j$ to $\boldsymbol{s}_i$, and $l_{\perp 2}(\boldsymbol{s}_i, \boldsymbol{\eta}_j)$ is the one from another end point of $\boldsymbol{\eta}_j$ to $\boldsymbol{s}_i$. It is the counter-harmonic mean between $l_{\perp 1}(\boldsymbol{s}_i, \boldsymbol{\eta}_j)$ and $l_{\perp 2}(\boldsymbol{s}_i, \boldsymbol{\eta}_j)$, which penalizes more on the deviation from the path than other mean measures [22]. Here we assume that $d_\perp(\boldsymbol{s}_i, \boldsymbol{\eta}_j) = 0$ if $l_{\perp 1}(\boldsymbol{s}_i, \boldsymbol{\eta}_j) = l_{\perp 2}(\boldsymbol{s}_i, \boldsymbol{\eta}_j) = 0$. Note that $\lim_{x,y\to 0}(x^2 + y^2)/(x + y) = 0$. Letting

$$d_{\angle,\perp}(\boldsymbol{s}_i, \boldsymbol{\eta}_j) := \log_2(d_\angle(\boldsymbol{s}_i, \boldsymbol{\eta}_j)) + \log_2(d_\perp(\boldsymbol{s}_i, \boldsymbol{\eta}_j)),$$

we define the model fidelity $Z(\boldsymbol{D}|\boldsymbol{B})$ by

$$Z(\boldsymbol{D}|\boldsymbol{B}) := \sum_{i=0}^{|\boldsymbol{B}|-1} \sum_{j=b_i}^{b_{i+1}-1} d_{\angle,\perp}((\boldsymbol{x}_{b_i}, \boldsymbol{x}_{b_{i+1}}), (\boldsymbol{x}_j, \boldsymbol{x}_{j+1})). \tag{7}$$

Note that when all points are selected as delimiters, the angular and perpendicular distances between a segment and itself are zeros, in which case $Z(\boldsymbol{D}|\boldsymbol{B})$ is minimized,

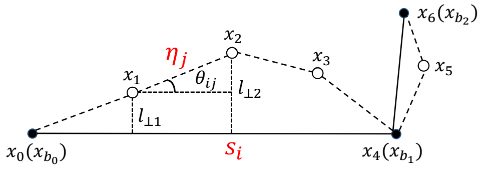


Fig. 10. An example path and a path description.

i.e., the maximal fidelity is achieved. Note that in the case of $x = 0$ for $\log_2(x)$, we use $\lim_{x\to 0}\log_2(x) = -\infty$.

As mentioned before, there is a clear tradeoff between model complexity $G(\boldsymbol{B})$ and model fidelity $Z(\boldsymbol{D}|\boldsymbol{B})$. $G(\boldsymbol{B})$ increases with increasing size of the set $\boldsymbol{B}$, while $Z(\boldsymbol{D}|\boldsymbol{B})$ tends to decrease as $\boldsymbol{B}$ expands. Thus, our problem of finding the best model (or the optimal delimiter set) $\boldsymbol{B}^\star$ now becomes the one of achieving the optimal tradeoff, which is formally given by the following MDL cost minimization problem:

$$\boldsymbol{\mathcal{P}}: \qquad \boldsymbol{B}^\star = \arg\min_{\boldsymbol{B}} G(\boldsymbol{B}) + Z(\boldsymbol{D}|\boldsymbol{B}).$$

4.3 Optimal Turn Detection

We are now ready to find the optimal solution $\boldsymbol{B}^\star$ to $\boldsymbol{\mathcal{P}}$. To this end, we establish that the problem $\boldsymbol{\mathcal{P}}$ is equivalent to a shortest-path problem on a weighted directed acyclic graph.

Consider a path $\boldsymbol{D} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$, where without loss of generality we assume that the indexes of the points along the path are ordered by their timestamps. The indexes of the points used as delimiters in $\boldsymbol{B}$ are then also ordered. We first construct a graph $\mathcal{G}$ of $N+1$ nodes, where node i corresponds to point $\boldsymbol{x}_i$ along the path $\boldsymbol{D}$, and a *directed* edge e_{ij} is added from node i to node j if $\boldsymbol{x}_i$ appears *before* $\boldsymbol{x}_j$ in $\boldsymbol{D}$. Note that this graph is directed and acyclic. In addition, letting $\boldsymbol{D}_{i,j} := \{\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_j\}$ be the path segment (the sub-path) between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, and if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the only two delimiters in $\boldsymbol{D}_{i,j}$, then we see from (5) that the MDL cost $L(\boldsymbol{D}_{i,j})$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ becomes

$$L(\boldsymbol{D}_{i,j}) := G(\{i,j\}) + Z(\boldsymbol{D}_{i,j}|\{i,j\}). \tag{8}$$

Then, we have the following result:

Theorem 1. *Let w_{ij} be the edge weight of a directed edge from node i to node j in $\mathcal{G}$. If the edge weight w_{ij} is set to the MDL cost in (8), i.e., $w_{ij} := L(\boldsymbol{D}_{i,j})$, then the problem $\boldsymbol{\mathcal{P}}$ is equivalent to the problem of finding the shortest path (or the minimum-weight path) from node 0 to node N on $\mathcal{G}$.*

Proof: Fix $\boldsymbol{B}$. Observe that (6) and (7) are additively separable. Thus, we see that $L(\boldsymbol{D})$ can be written as the sum of $L(\boldsymbol{D}_{i,j})$ over all the ordered pairs (i,j) of the indexes of two consecutive delimiters in $\boldsymbol{B}$. In addition, we observe that the nodes on $\mathcal{G}$ that correspond to the delimiters in $\boldsymbol{B}$ form a path on $\mathcal{G}$, i.e., a sequence of nodes where each node in the sequence has a directed edge into the node next to it, i.e., $b_i \to b_{i+1}$. Thus, since $w_{ij} = L(\boldsymbol{D}_{i,j})$ for each edge from i to j, the weight of the path on $\mathcal{G}$, which is the sum of the weights of the directed edges comprising the path, becomes identical to $L(\boldsymbol{D})$ for a given $\boldsymbol{B}$. Therefore, $\boldsymbol{\mathcal{P}}$ is equivalent to finding the shortest path (or the minimum-weight path) from node 0 to node N on $\mathcal{G}$. □

It is worth noting that edge weights can be negative due to the logarithm in (6) and (7). Thus, the shortest path problem can be generally solved by Bellman-Ford algorithm [23].

ALGORITHM 1: MET

```
Input : $b_i := 0, \boldsymbol{B} := \{b_i\}, \boldsymbol{D}_s := \{\boldsymbol{x}_{b_i}, \boldsymbol{x}_{b_i+1}\}$, and
        $k := b_i + 2$
Output: $\boldsymbol{B}$ /* Detected delimiter indexes */
1  while A new point $\boldsymbol{x}_k$ arrives do
2      $\boldsymbol{D}_s := \boldsymbol{D}_s \cup \{\boldsymbol{x}_k\}$
3      $\mathcal{C}(A_{k-1}) :=$
         $G(\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_{k-1}, \boldsymbol{x}_k\}) + Z(\boldsymbol{D}_s|\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_{k-1}, \boldsymbol{x}_k\})$
4      $\mathcal{T}(\bar{A}_{k-1}) = G(\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_k\}) + Z(\boldsymbol{D}_s|\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_k\})$
5      if $\mathcal{C}(A_{k-1}) < \mathcal{T}(\bar{A}_{k-1})$ then
6          $\boldsymbol{B} := \boldsymbol{B} \cup \{k-1\}$
7          $b_i := k-1$
8          $\boldsymbol{D}_s := \{\boldsymbol{x}_{b_i}, \boldsymbol{x}_{b_i+1}\}$
9      end
10     $k := k+1$
11 end
12 return $\boldsymbol{B}$
```

Since it is a shortest path problem on a weighted directed acyclic graph, it can also be solved more efficiently by using topological sorting [24].

4.4 MET – An Efficient Online Algorithm

Observe that the shortest path algorithm is only applicable *offline*, i.e., when the entire path information is available, as it needs to calculate the MDL cost for every node pair. Thus, we below propose an efficient *online* algorithm MET, to achieve the real-time detection of turning points along the movement path of a user. MET is an integral part of PATHLIT by allowing PATHLIT to determine, in real time, whether to keep the coordinates of each end point of a two-second segment learned by the self-attention network model in PATHLIT.

We can see from (6) and (7) that once a point along the path is accepted as a delimiter, the MDL costs up to the point remain the same and all the points that appear before this point are not needed for computing the MDL costs afterward. Suppose that the last detected delimiter is $\boldsymbol{x}_{b_i}$, and the user's current position is $\boldsymbol{x}_k$ ($b_i < k$). Note that no point is considered a delimiter (or turn) in $\{\boldsymbol{x}_{b_i+1}, \cdots, \boldsymbol{x}_{k-2}\}$. Our online algorithm is then to determine whether $\boldsymbol{x}_{k-1}$ is a turn or not by considering the points between $\boldsymbol{x}_{b_i}$ and $\boldsymbol{x}_k$.

Letting $\boldsymbol{D}_{b_i,k} := \{\boldsymbol{x}_{b_i}, \boldsymbol{x}_{b_i+1}, \cdots, \boldsymbol{x}_k\}$, we decide whether to *accept* or *reject* $\boldsymbol{x}_{k-1}$ as a turn (or delimiter). Let A_{k-1} be the event that $\boldsymbol{x}_{k-1}$ is *accepted* as a turn and $\bar{A}_{k-1}$ be the event that $\boldsymbol{x}_{k-1}$ is *rejected* as a turn. We then define two MDL costs $\mathcal{C}(A_{k-1})$ and $\mathcal{T}(\bar{A}_{k-1})$ by

$$\mathcal{C}(A_{k-1}) := G(\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_{k-1}, \boldsymbol{x}_k\}) + Z(\boldsymbol{D}_{b_i,k}|\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_{k-1}, \boldsymbol{x}_k\}),$$
$$\mathcal{T}(\bar{A}_{k-1}) := G(\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_k\}) + Z(\boldsymbol{D}_{b_i,k}|\{\boldsymbol{x}_{b_i}, \boldsymbol{x}_k\}),$$

respectively. Then, if $\mathcal{C}(A_{k-1}) < \mathcal{T}(\bar{A}_{k-1})$, $\boldsymbol{x}_{k-1}$ is accepted as a turn and the position index $k-1$ is added as a new delimiter into the set $\boldsymbol{B}$. Otherwise, it is rejected. Upon the arrival of $\boldsymbol{x}_{k+1}$, we repeat the process to decide whether to accept $\boldsymbol{x}_k$ as a turn. The whole algorithm operation is summarized in Algorithm 1. We have the following result on its time complexity.

Theorem 2. *Consider a path $\boldsymbol{D} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$. Suppose that k delimiters along the path $\boldsymbol{D}$ are detected by MET in Algorithm 1. Let α_i be the number of location points between two* *consecutive delimiters $x_{b_{i-1}}$ and x_{b_i} (inclusive), $i=1,2,\ldots,k$, where $x_{b_0} = x_0$. The time complexity of MET is $O(\alpha_{\max}N)$, where $\alpha_{\max} := \max\{\alpha_1, \alpha_2, \cdots, \alpha_k\}$.*

Proof: We first analyze the time complexity of MET in Algorithm 1 when processing new points arriving after identifying the latest delimiter, say, $x_{b_{i-1}}$, until after the next delimiter x_{b_i} is identified, $i = 1,2,\ldots,k$. Since α_i is the total number of points between the delimiters (inclusive), and the delimiter $x_{b_{i-1}}$ has already been determined up on arrival of $x_{b_{i-1}+1}$, it boils down to analyzing the time complexity of executing the iterations in the while loop for $\alpha_i - 1$ points, i.e., $x_{b_{i-1}+2}, x_{b_{i-1}+3}, \ldots, x_{b_{i-1}+\alpha_i}$. Note that the next delimiter x_{b_i} is determined up on arrival of the point next to x_{b_i}, which is $x_{b_{i-1}+\alpha_i}$.

Observe that each iteration in the while loop is mainly governed by the operations of computing the MDL costs $\mathcal{C}(A_{k-1})$ and $\mathcal{T}(\bar{A}_{k-1})$ at Lines 3 and 4, respectively, because all the other operations take constant time, i.e., $O(1)$. Here, $k = b_{i-1} + j$, where $j = 2,3,\ldots,\alpha_i$. In addition, from (6), we see that the computations of $G(\{x_{b_i}, x_{k-1}, x_k\})$ and $G(\{x_{b_i}, x_k\})$ take $O(1)$, since their corresponding set sizes of B in (6) are three and two, respectively. Also, from (7), we see that the time complexity of computing each of $Z(\boldsymbol{D}_s|\{x_{b_i}, x_{k-1}, x_k\})$ and $Z(\boldsymbol{D}_s|\{x_{b_i}, x_k\})$ is linearly proportional to $|\boldsymbol{D}_s|$, which grows from two to α_i. Thus, the time complexity of executing the iterations in the while loop for $\alpha_i - 1$ points is $\sum_{j=2}^{\alpha_i} O(j) = O(\alpha_i^2)$, $i = 1,2,\ldots,k$.

Therefore, by noting that

$$\sum_{i=1}^{k} \alpha_i^2 \leq \alpha_{\max}(\alpha_1 + \cdots + \alpha_k) = O(\alpha_{\max}N),$$

we see that the time complexity of MET is $O(\alpha_{\max}N)$. □

Remark 1. *The time complexity of MET can be up to $O(N^2)$, since $\alpha_{\max}$ can be on the order of N. One such case is when only one delimiter is identified, and it is done up on arrival of the last point x_N. However, in practice, $\alpha_{\max}$ remains bounded and independent of N. Thus, the time complexity of MET is generally linear with respect to the number of points (position vectors) on a movement path, i.e., $O(N)$.*

Remark 2. *Unlike the shortest path algorithms that can only be applied offline, MET is an online algorithm, which enables the real-time detection of turning points along the movement path of a user. In particular, the offline shortest path algorithms require the graph $\mathcal{G}$ to be constructed a priori. In other words, the MDL cost $L(\boldsymbol{D}_{i,j})$ in (8) needs to be computed as the edge weight w_{ij} for each pair of x_i and x_j $(i < j)$. From (7), we also see that it takes $O(N)$ to compute $Z(\boldsymbol{D}_{i,j}|\{i,j\})$ for each pair. Note that there are $N(N+1)/2$ node pairs. Thus, it takes $O(N^3)$ to construct the graph $\mathcal{G}$. While the shortest path algorithms allow us to find the optimal (offline) solution $B^\star$ to $\boldsymbol{\mathcal{P}}$, the overall time complexity of finding $B^\star$ is at least $O(N^3)$ regardless of the choice of the shortest path algorithm, e.g., topological sorting for finding the shortest path on a weighted directed acyclic graph. In contrast, the time complexity of our online algorithm MET is $O(\alpha_{\max}N)$, where $\alpha_{\max}$ often remains bounded.*

5 Experiment Results

In this section, we present extensive experiment results. We discuss experiment settings and evaluate the system-level performance of PATHLIT by comparing it with state-of-the-art algorithms. We also study the impact of system components and parameters on PATHLIT.

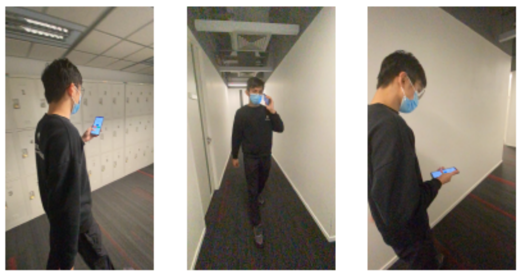


Fig. 11. Campus data collection with a phone held freely.

5.1 Experiment Settings

We conduct experiments on the RoNIN open dataset [13] and our campus dataset. The sampling frequency of IMU sensors for both datasets is 200Hz.[2] The RoNIN dataset contains 69 paths for training, 16 paths for validation and 64 paths for test, collected by 100 different people with three different Android devices, i.e., Asus Zenfone AR, Samsung Galaxy S9 and Google Pixel 2 XL. Among the 64 test paths, half of them are collected by people who also contribute to the training set, referred to as 'test seen' paths, since their walking patterns may have been *seen* by the model. The other half are collected by people who only contribute to the test set, referred to as 'test unseen' paths.

We use the ground truth velocity vectors provided in the RoNIN dataset for our PATHLIT model training. They were obtained based on a separate Tango mobile phone that was bound to the chest of a user, in which case the coordinate system is fixed and thus its moving direction can be simply regarded as the heading direction of the user [13]. In other words, the ground truth velocity vectors are used as the desired output, while the input data is the IMU signals collected from different Android devices and fed into the model to predict their corresponding velocity vectors. Note that the IMU signals here are transformed into the global coordinate system due to their possible orientation changes.

In addition, our campus dataset contains 48 test paths collected by six people walking along four pathways with a phone held freely (Figure 11), using two different Android devices, namely Huawei Mate 30 Pro and vivo Y50. We collect 12 paths along each pathway. When a person is walking, another person holds a camera to record the ground truth. Note that the recorded video is not used for model training but for calculating estimation errors only. Our campus dataset is also only used for *inference* to evaluate 'model generalization', i.e., the ability of a model to react to the new dataset obtained in a different environment.

We use the training data of 69 paths from the RoNIN dataset to train our self-attention network model in PATHLIT and other models in the state-of-the-art algorithms and then evaluate their performance on all the test paths for inference, including our campus dataset. Note that the performance of PATHLIT is based on both the self-attention network model and MET. We present the average results

2. Note that the sampling frequency of 200Hz for collecting IMU signals is supported by most off-the-shelf mobile phones [13], [25], [26].

TABLE 1
Performance comparison with 'mean (standard deviation)' value. PATHLIT(–) stands for PATHLIT without MET.

Scheme	ATE (meters)			RTE (meters)			Number of Data Points Used		
	Seen	Unseen	Campus	Seen	Unseen	Campus	Seen	Unseen	Campus
PATHLIT	**3.30** (2.21)	**5.78** (4.82)	**4.88** (3.92)	**2.65** (1.09)	**4.03** (1.15)	**4.72** (3.08)	**143.32** (35.58)	**150.62** (37.35)	**11.34** (3.62)
PATHLIT(–)	3.85 (2.32)	6.40 (4.90)	5.29 (4.05)	2.78 (1.33)	4.21 (1.39)	5.18 (3.52)	961.66 (256.54)	981.03 (252.38)	87.13 (21.16)
RoNIN	4.97 (2.21)	6.97 (5.01)	6.06 (4.29)	3.02 (1.52)	4.68 (2.20)	5.70 (4.31)	961.66 (256.54)	981.03 (252.38)	87.13 (21.16)
CNN	12.02 (7.65)	12.06 (10.98)	18.40 (4.56)	7.62 (3.84)	10.36 (3.70)	15.25 (3.03)	961.66 (256.54)	981.03 (252.38)	87.13 (21.16)
A^3	19.05 (12.57)	16.63 (10.88)	12.09 (7.17)	15.56 (7.48)	15.89 (5.86)	12.97 (7.92)	961.66 (256.54)	981.03 (252.38)	87.13 (21.16)
PDR	26.04 (13.16)	23.49 (10.35)	23.51 (7.28)	23.76 (11.22)	23.07 (9.05)	19.04 (4.87)	961.66 (256.54)	981.03 (252.38)	87.13 (21.16)

of the test seen, test unseen and campus paths separately, unless otherwise mentioned. We consider the following state-of-the-art algorithms for performance comparison:

- RoNIN [13]: It leverages sequential dependencies in IMU signals obtained during a path trajectory and recovers the movement path using an LSTM model.
- CNN [27]: It extracts features from IMU signals of each step and estimates step lengths and directions with a one-dimensional convolutional neural network (CNN) model.
- Pedestrian dead reckoning (PDR) [6]: It estimates the user's step lengths via a linear model and learns movement directions using an Android API for device orientation.
- A^3 [7]: It improves on PDR by continuously calibrating the device orientation with IMU signals.

For PDR, A^3, and RoNIN, we set their parameters as described in [6], [7], and [13], respectively. For CNN, we build a CNN model that has two layers of 1D convolution and two layers of 1D max pooling with the ReLU activation function. For our self-attention network model in PATHLIT, we set its baseline parameters as follows. The encoder is a stack of two encoder layers, where each layer has four heads. The decoder also has the same structure. We use the learning rate with cosine decay [28] where it increases to 0.008 in the first 100 epochs and then decreases. The dropout rate is set to 0.2. We set the embedding dimension d_{em} to 64.

We use absolute trajectory error (ATE) and relative trajectory error (RTE) for performance comparison. The ATE is the root mean squared error (RMSE) between ground-truth and estimated (complete) paths, defined as

$$\text{ATE} = \sqrt{\frac{\sum_{i=1}^{N} \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|_2^2}{N}},$$

where N is the total number of the coordinates of points on a path, $\boldsymbol{x}_i$'s are the ground truth coordinates and $\hat{\boldsymbol{x}}_i$'s are the estimated coordinates for each point, and $\|\cdot\|_2$ is the l_2 norm. The RTE is the average RMSE over a fixed time interval, which is defined as

$$\text{RTE} = \sqrt{\frac{\sum_{i=1}^{(N-\Delta N)} \|(\boldsymbol{x}_{i+\Delta N} - \boldsymbol{x}_i) - (\hat{\boldsymbol{x}}_{i+\Delta N} - \hat{\boldsymbol{x}}_i)\|_2^2}{(N-\Delta N)}},$$

where ΔN is the number of the coordinates *apart* and is set to 30 (equivalent to one minute) in our experiments.

We also use precision, recall and F-score when it comes to the performance evaluation in turn detection. Let T_p be the number of true positives, which indicates the number of correctly detected turns by a turn detection algorithm. Let F_p be the number of false positives, indicating the number of extra points that are detected as turns. Let F_N be the number of false negatives, denoting the number of (true) turns that are not detected. Then, the precision, recall, and F-score are given by $P = T_P/(T_P + F_P)$, $R = T_P/(T_P + F_N)$, and $F = 2PR/(P + R)$, respectively.

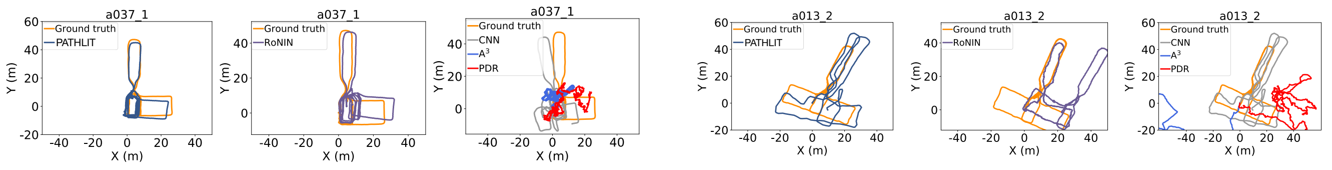
5.2 Overall Comparison with the State of the Art

We first present the ATE and RTE results of PATHLIT and state-of-the-art algorithms on the two datasets in Table 1 to demonstrate the superiority of PATHLIT in path recovery. We also show the results of PATHLIT(–), which is PATHLIT without MET, to see the performance improvement from each system component of PATHLIT.

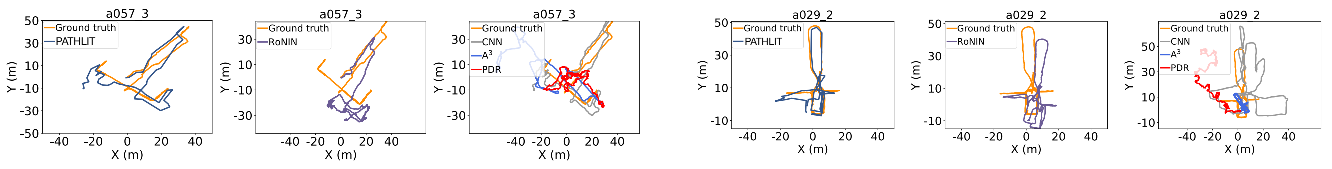
PATHLIT achieves the best performance in both ATE and RTE for both datasets. The performance improvement comes from both system components of PATHLIT, which are the self-attention network model and MET. In other words, PATHLIT *smooths out* the recovered path by the former, leading to further improvement. In addition, the superior performance of PATHLIT in recovering the *unseen* and *campus* paths demonstrates its better model generalization and readiness for practical deployment. RoNIN has satisfactory performance. However, the LSTM model may leverage irrelevant signal patterns for prediction due to its sequential dependencies, resulting in lower accuracy. The CNN model is less capable in learning temporal correlations between signal patterns, thereby leading to worse performance than that of RoNIN. A^3 has a better calibration of the device orientation compared to PDR, and thus exhibits better performance than PDR for all the datasets. Nonetheless, due to the dynamically changing behaviors of the mobile devices of moving users, the orientation calibration in A^3 can perform poorly, hampering its accuracy.

In addition to its superior performance in ATE and RTE, PATHLIT uses much fewer data points to describe a recovered path compared to the other schemes, thanks to its effectiveness in detecting the user's turns correctly and using them for a path description. For both datasets, PATHLIT requires only around 15% of the total data points to precisely and concisely describe a movement path, while the others rely on all the data points along a path. Our novel turn detection algorithm MET in PATHLIT is able to detect the user's turns along a path correctly on the fly while the movement path is being recovered by the other component in PATHLIT, the self-attention network model.

To demonstrate the effectiveness of PATHLIT in learning path descriptions, we visualize a few representative recovered paths by PATHLIT and other algorithms in Figure 12 and Figure 13. The paths learned by PATHLIT *best match* the ground-truth paths. While the paths recovered by RoNIN exhibit the second best performance, they do not match the shapes and orientations of the ground-truth paths as



(a) Test seen.



(b) Test unseen.

Fig. 12. Path recovery results (RoNIN dataset).

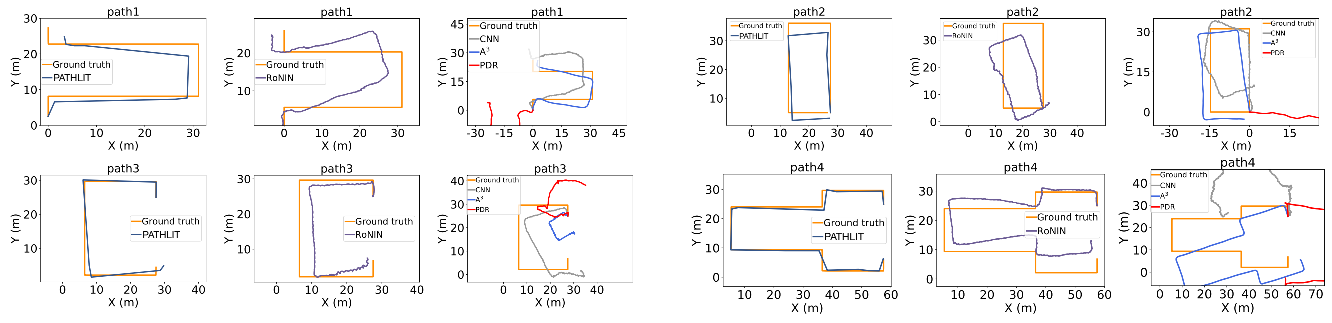
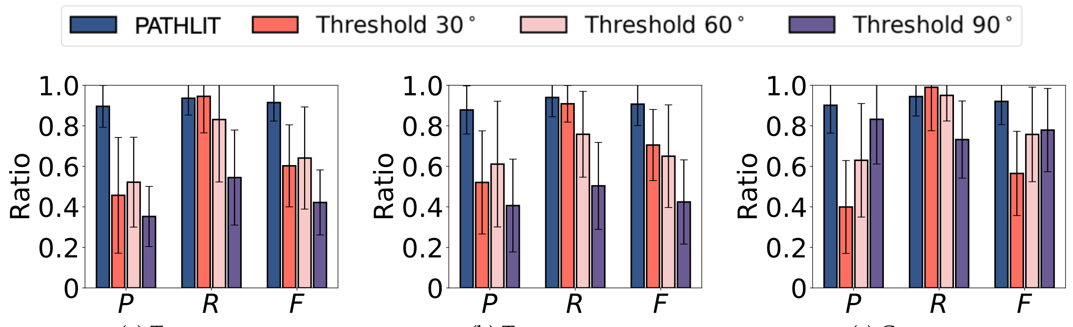


Fig. 13. Path recovery results (campus dataset).



(a) Test seen (b) Test unseen (c) Campus

Fig. 14. Performance comparison of MET in PATHLIT and other threshold-based turn detection algorithms.

much as PATHLIT does. The CNN model is able to recover the shapes of the paths roughly, but it fails to estimate the orientations correctly. The paths learned by A^3 and PDR show poor recovery performance.

5.3 System Component Study

Turn detection: We have demonstrated the superior performance of PATHLIT, which comes from both the self-attention network model and MET. To further illustrate the quality of MET for turn detection, we show in Figure 14 the turn-detection performance of MET in PATHLIT and the thresholding method with different threshold values. MET outperforms the others significantly and performs consistently across different datasets, exhibiting its capability of detecting turns correctly regardless of the shapes of movement paths. However, the performance of the thresholding method highly depends on the choice of its threshold value. Higher threshold values tend to miss more user turns, while lower ones tend to have more false positives. Thus, it is deemed infeasible to choose an appropriate threshold value for practical deployment. In contrast, MET does not require any parametrization or a calibration of threshold values.

In addition, to better understand the quality of MET for turn detection, we visualize the process of PATHLIT in detecting turning points on Path a000_7 from the RoNIN open dataset in Figure 15. We can see that each time the condition for $\mathcal{C}(A_{k-1}) < \mathcal{T}(\bar{A}_{k-1})$ is satisfied (Line 5 in Algorithm 1), a turn is *accurately* detected, showing the effectiveness of MET. Note that the numbers right next to the detected turns in Figure 15(b) correspond to the coordinates' indexes in Figure 15(a).

Positional encoding: We apply *positional encoding* to enforce

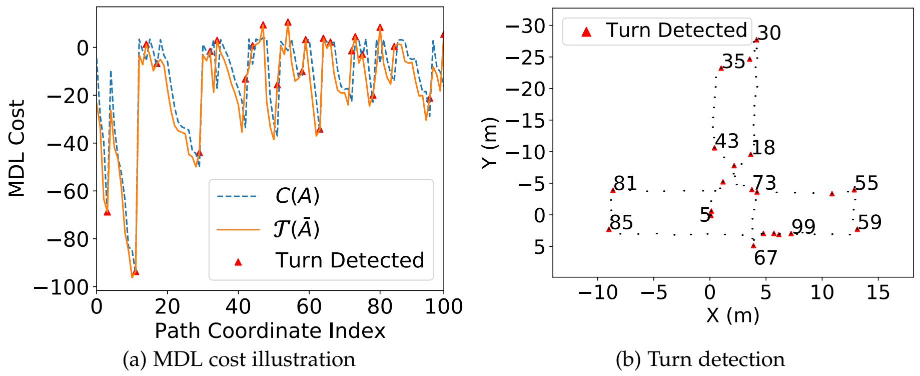


Fig. 15. Illustrating the MDL costs and turn detection in PATHLIT. The numbers in (b) represent path coordinate indexes in (a).

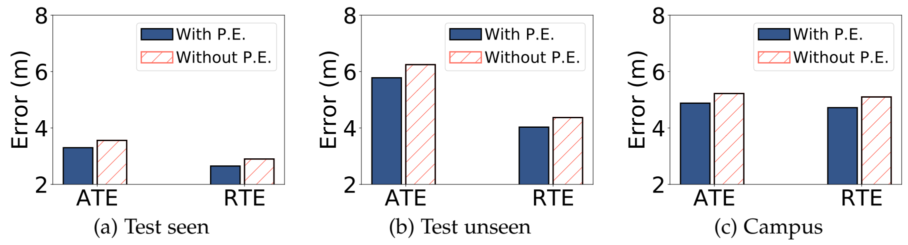


Fig. 16. Performance of models built with and without positional encoding (P.E.).

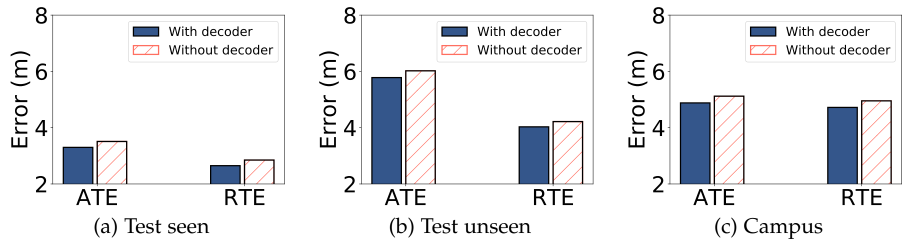


Fig. 17. Performance of models built with and without decoder.

the temporal relationship within a sequence when building our self-attention network model. To validate the effectiveness of such an intra-sequence positional encoding, we show the performance of PATHLIT with the model built with and without the positional encoding in Figure 16, where P.E. stands for positional encoding. As can be seen from Figure 16, the positional encoding introduces around 10% improvement. This is important since it enables the model to learn correlations between signal patterns within and across sequences while maintaining the sequential order of sensor readings within each sequence.

Decoder: While our network model is built based on an encoder-decoder architecture, it can also have an *encoder-only* structure in which case we add a linear layer after having the final representations from the encoder to predict velocity vectors directly. We are interested in how much performance degradation PATHLIT would have with the encoder-only model. As shown in Figure 17, the performance degradation by removing the decoder is insignificant (∼5.3%). However, the decoder doubles the model size, i.e., the number of learnable parameters. Including the decoder in the network increases the training time by more than 120% in our scenario, which is from around eight hours to 18 hours. Thus, when deploying PATHLIT in resource-constrained IoT devices, e.g., Raspberry Pi, one can adopt the encoder-only model as a relatively lightweight model.

5.4 System Parameter Evaluation

Time window (sequence length): We use a short sequence in the network model, which contains 400 readings for each sensor. It corresponds to the number of readings collected for two seconds. In Figure 18, we show the impact of different sequence lengths on PATHLIT's performance. We can see that if it is too short, each sequence would not be long enough to contain IMU signal patterns from a walking context. In addition, if it is too long, each sequence would have several walking contexts mixed up, making it difficult to extract useful features.

Embedding dimension: Each collection of six IMU sensors' readings is first transformed into an embedding vector of size d_{em} in the network model. We here evaluate the impact of different choices of d_{em} on the performance of PATHLIT. As shown in Figure 19, both ATE and RTE tend to decrease and get saturated under the *test-seen* data as d_{em} increases, while they also decrease but then increase slightly under the *test-unseen* and *campus* data with increasing values of d_{em}. Thus, it would not be beneficial to keep increasing the dimension size d_{em} since it could make the model overfitted to the training data, thereby hampering its model generalization.

5.5 Turn Detection as Trajectory Compression

Recall that our turn-detection algorithm MET in PATHLIT outputs much fewer data points to describe a user path. Here we further evaluate its feasibility as a *compression* algorithm for path/trajectory compression, since both the turn detection and trajectory compression problems are similar in the sense that they find a compact set of trajectory data points. To this end, we use the Microsoft GeoLife dataset [29], which is commonly used for the latter problem in the literature. The dataset contains 17,621 trajectories from 182 users. The trajectories are collected *outdoors* and consist of the GPS readings (latitudes and longitudes) along the paths. The total travel distance is about 1.2M kilometers, and the total travel time is more than 48,000 hours. We compare MET with the following popular *online* compression algorithms:

- OPW [30]: It keeps a new trajectory point if it is considered important in comparison with the points in the buffer. This decision involves a threshold.
- SQUISH [31]: It accepts all trajectory points until a fixed-size buffer becomes full. Then, every new point replaces a point in the buffer that is considered least important.
- STTrace [32]: It is similar to SQUISH, but with a slightly different buffer replacement policy.
- Dead Reckoning [33]: It predicts the next trajectory point based on recent ones. If the prediction error is greater than a given threshold, its corresponding point is kept.

Due to the large number of trajectories, we divide them into four bins based on their lengths, which are given in the number of the readings. The first bin contains the trajectories of lengths (in terms of the number of readings) below one hundred. The second one is for the lengths between one hundred and one thousand, and the third one is for the lengths between one thousand and ten thousand. The last one contains the rest. We also observe that the algorithms considered here have all similar time complexity on the dataset.

We show the ATE and RTE results in Figure 20. We can see that MET in PATHLIT outperforms all the other trajectory-compression algorithms substantially in both ATE and RTE. This demonstrates its feasibility and effectiveness for trajectory compression. Note that ATE and RTE are measured based on the latitudes and longitudes of trajectory data points, whose units are in degrees. While having inferior performance, OPW and Dead Reckoning require threshold values to be determined based on the lengths and shapes of the trajectories, which are hard to calibrate in practice. In addition, SQUISH and STTrace need to choose the size of a fixed-size buffer, which holds a limited number of trajectory data points, to be proportional to the length of a trajectory. However, we observe that a non-negligible error would be involved if the buffer size is small, since it becomes more difficult to choose which points to be stored in the buffer. Thus, their performance becomes unsatisfactory for short trajectories. In contrast, MET does not require any calibration of threshold values and is also not influenced by the trajectory lengths.

6 Related Work

We review in this section three main categories of work relevant to our system, namely path recovery, turn detection, and trajectory compression.

6.1 Path Recovery

Multi-sensor approaches: Additional sensors [34]–[48] in the smartphone, in addition to IMU sensors, may be utilized to facilitate path recovery. A fusion algorithm is proposed in [41] to learn user paths from the signals from IMU and Bluetooth low energy (BLE) sensors. After obtaining a raw position from IMU sensors, it leverages a particle filter to adjust the raw position with an extra location estimate provided by BLE. However, it requires the installation of BLE beacon sensors on site. In [49], the camera in the smartphone is also used together with IMU sensors to improve the quality of location estimates. The estimation quality is improved by leveraging the aspect ratio of the frontal wall that the camera faces, where all the aspect ratios of walls are stored in a database in advance. Nonetheless, keeping the camera turned on may drain the device battery quickly and also introduce potential privacy concerns. In contrast, PATHLIT works based *only* on the IMU signals without any additional infrastructure support while being non-intrusive to users.

Leveraging predefined walking contexts: There are several approaches [8], [26], [50]–[53] that require prior knowledge of walking contexts (or behaviors). In [8], the user's mobile device is required to be placed in three postures, namely in the pocket, in hand swinging and in hand holding. In addition, user information such as user height is needed for path recovery. TLIO [26] requires the IMU sensors to be mounted on the headset in which case the sensors are more or less stationary, i.e., the sensor readings are much more stable. Other approaches focus on extracting signal features from different walking contexts using a two-stage learning process [9], [10], [54], where it first classifies current IMU signals into a walking context and then recovers the path under that specific context. However, these techniques need to have a *predefined* list of walking contexts and require manual labeling of the contexts for their corresponding IMU signals in model training.

Leveraging temporal correlations: Recent data-driven techniques [11]–[13], [55] attempt to recover user's movement paths in the wild by exploiting the possible temporal correlations within IMU signals. They commonly use LSTM to learn a displacement from the current IMU signals by assuming the presence of long-term signal correlations. However, such an assumption may not always hold in practice. For instance, the posture and position of the user's mobile device keep changing over time, and the changes can also be quite drastic, leading to totally different signal patterns. In contrast, PATHLIT is designed to extract (implicit) context features from the IMU signals in a short period of time for path recovery, during which the walking context is less likely to change.

6.2 Turn Detection

Turn detection is important for online path recovery and indoor localization since it can be used to detect turns as

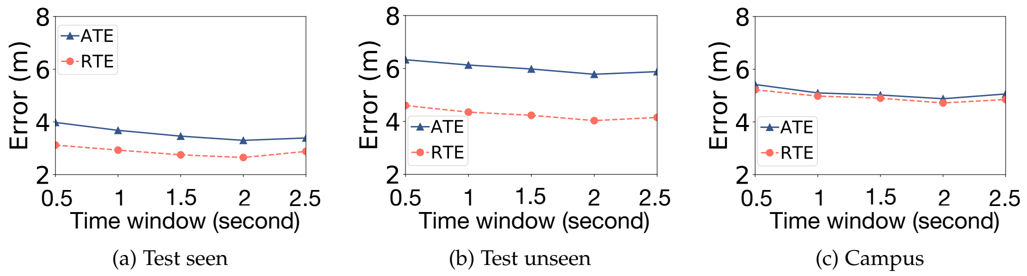


Fig. 18. Impact of different time windows (sequence lengths) in PATHLIT, where 200 samples are collected per second.

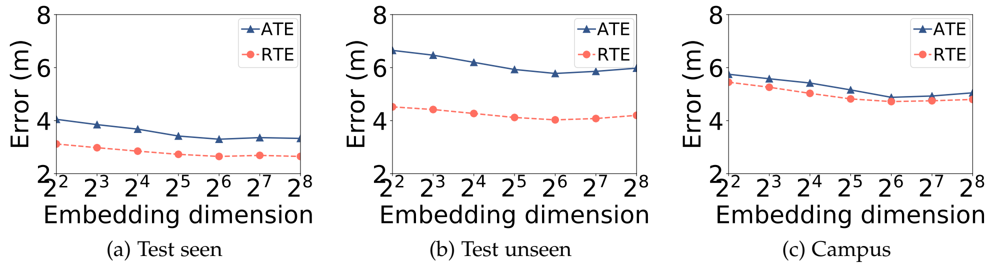


Fig. 19. Impact of different embedding dimension sizes (d_{em}) in PATHLIT.

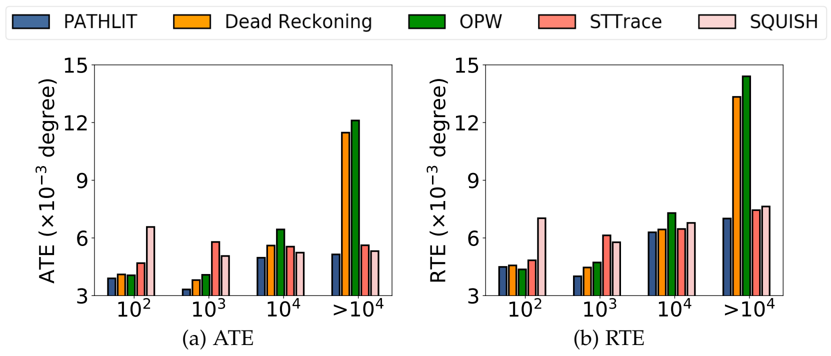


Fig. 20. Performance comparison with state-of-the-art *online* trajectory compression algorithms by using MET in PATHLIT for trajectory compression.

landmarks to reduce the noises or errors in their applications. In what follows, we review recent *online* turn detection algorithms.

Map-based approaches: Indoor maps can be leveraged to detect user turns [4], [11], [18], [56]–[60]. For instance, Zee [4] takes advantage of map constraints to detect user turns with a particle filter, which introduces additional computation overheads. The proposed scheme in [58] uses the map to refine a sequence of turns detected on the fly. In a similar vein, the approach in [11] measures the traveled distance between two consecutive turns on the map to decide whether the (detected) turns are valid. However, the requirement of indoor maps is not practically viable as property owners may not want to share the maps. In contrast, MET in PATHLIT is purely based on IMU signals without requiring indoor maps.

Threshold-based approaches: Prior studies identify user turns with turning thresholds [17], [61]–[65]. In [62], a large amount of training data is collected to set threshold values for both left- and right-turn angular velocities. In [63], a threshold of 10 degrees per second for angular velocity is used to detect whether there is a directional change. However, it is challenging to set the threshold values *properly*, as users may handle their devices freely while walking, leading to unexpected changes in angular velocity. In contrast, MET in PATHLIT achieves accurate online turn detection without requiring any threshold values. We demonstrated in Section 5 (e.g., Figure 14) that MET outperforms threshold-based approaches significantly.

6.3 Trajectory Compression

We demonstrated in Section 5.5 the feasibility of our turn detection algorithm MET as an online trajectory compression algorithm as it aims to minimize the number of data points (i.e., turning points) to describe a user path accurately. Hence, we below briefly review online trajectory compression algorithms.

OPW [30] maintains a buffer of points (path coordinates) that constitutes the trajectory. Upon arrival of a new point, OPW constructs a line segment using the incoming point and the 'starting' point in the buffer. It then computes the perpendicular Euclidean distance from each point in the buffer to the line segment. If the maximum distance is larger than a predetermined threshold, the new point is considered 'important' and kept in the buffer. In addition, the point with the maximum distance is used as a new starting point. In a similar vein, STTrace [32] and SQUISH [31] build up a fixed-size buffer and replace a point in the buffer if the new incoming point is considered more important. Recently, reinforcement learning-based algorithms are proposed in [66],

[67] for online trajectory compression. While they also maintain a fixed-size buffer of points, they now train a neural network to identify and drop the *least* important point in the buffer whenever a new point is available. However, all the aforementioned algorithms need the parameters such as buffer size and threshold value to be chosen judiciously. In contrast, MET does not require calibration of any parameters while demonstrating its feasibility as an online compression algorithm.

7 Discussion

We below discuss the feasibility of extending PATHLIT to leverage other measurement signals in different application scenarios. We also explain the rationale behind using the global or world-frame coordinate system in PATHLIT and discuss how it could be further improved.

7.1 Fusion with Different Types of Measurements

PATHLIT only leverages IMU signals for easy deployment, but it can be readily extended to fuse with other measurements to account for different application scenarios if they are readily available. For instance, in indoor navigation, we can incorporate geomagnetic signals or radio frequency (RF) signals such as WiFi and BLE as the input (together with IMU signals) into the self-attention network model to facilitate the path recovery process. Geomagnetic signals can be appended to the input vectors directly to provide additional features for learning as its sensing frequency is usually the same as the one with the IMU signals. For RF signals, however, the sensing frequency may be much lower. To incorporate them into the model, we can interpolate the signals between two consecutive measurements or use one-hot vectors to just indicate the existence of sensible access points in each measurement.

In addition, our turn detection algorithm MET can be integrated into other location-based systems [3], [29]–[33], [68], [69] in a seamless manner. In Section 5.5, we have demonstrated its feasibility as a trajectory compression algorithm [29]–[33] when GPS measurements are used. Here we point out that it can also be used in other large-scale localization systems [3], [68], [69]. For instance, for an automatic floorplan construction, we can leverage crowdsourced user paths to learn their corresponding pathways. Each user path often only covers a small portion of a floor. Hence, we first detect the user turns using MET and then use these turns as anchor points to stitch the user paths together, from which the pathways in the floorplan can be reconstructed.

7.2 Global Coordinate System

Note that PATHLIT is built upon the global coordinate system provided by the operating system of a mobile device, which is used to infer the current heading direction. While it is not our main focus to improve the accuracy of the global coordinate system itself, our results so far indicate its usability and feasibility for IMU-based path learning. We expect that the improvement in the accuracy of the global coordinate system would lead to a further improvement in PATHLIT as the native application programming interfaces (APIs) in the mobile operating system keep on evolving with the advances of orientation estimation algorithms [70]. We below review how the global coordinate system has been used in recent path-learning systems and discuss state-of-the-art algorithms for better orientation estimations, which can be incorporated into PATHLIT to further improve the learning results.

Recent systems and algorithms [12], [13], [25], [37], [71] that aim to achieve better path learning under arbitrary walking contexts usually rely on the orientation estimations from mobile devices. For example, IONet [12] preprocesses raw IMU signals to obtain linear accelerations by leveraging the orientations provided by the mobile device. It then uses the processed IMU values to train a deep neural network to predict the path length and angle changes for each short-time interval. In a similar vein, RIDI [25] adopts the APIs provided by the mobile device to transform the mobile coordinate system to the global coordinate system. It then implements the support vector regression to obtain velocity values. RoNIN [13] utilizes the orientations provided by the mobile operating system in a testing phase and estimates the displacements using a trained LSTM model. We see that the orientation estimations from mobile devices are currently in a reasonable quality for (light-weight) path recovery, although the quality of the path recovery still depends on how the orientation estimations are effectively used.

It is also worth noting that there are several recent algorithms [55], [72]–[75] that leverage neural networks to calibrate the orientation estimations in a finer manner. For instance, IDOL [73] attaches a LiDAR to a mobile device to collect ground-truth orientation values. It estimates the orientation of the mobile device using a separately trained LSTM network, with IMU signals as an input and the corresponding measurements from the LiDAR as a ground truth. The orientation estimations are then improved with an extended Kalman filter. In addition, AI-IMU [74] mounts an Asus Tango phone on a human head to obtain the ground truth rotation matrix for phone headings in order to train an MLP network for improved orientation estimations. In testing, it leverages a multi-state cloning Kalman filter and a graph optimization estimator to further improve the estimations from the network. Note that due to the high computational overhead, only ten key frames every second are selected for the graph optimization. In light of the system development and deployment, we believe that PATHLIT can take advantage of the advances in the orientation estimations for better path learning, which yet comes with a trade-off between accuracy and system complexity.

8 Conclusion

We have presented PATHLIT, an accurate and efficient path description learning system. Thanks to the carefully designed self-attention network model and the MDL-based online turn-detection algorithm MET, PATHLIT is able to recover each segment of a movement path from a stream of IMU readings and determine whether to keep its end points on the fly, which leads to a succinct yet accurate path description. Extensive experiments have shown the superiority of PATHLIT over state-of-the-art algorithms for path recovery and the effectiveness of MET for turn detection and trajectory compression.

ACKNOWLEDGEMENTS

The work of Weipeng Zhuo was supported by IRADS (2022B1212010006, R0400001-22) and UICR0700100-24. The work of Shiju Li and Chul-Ho Lee was supported in part by the NSF under Grant Nos. 2209921 and 2209922.

REFERENCES

[1] Z. Yuan, D. Zhu, C. Chi, J. Tang, C. Liao, and X. Yang, "Visual-inertial state estimation with pre-integration correction for robust mobile augmented reality," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019.

[2] X. Xu, J. Li, T. Yuan, L. He, X. Liu, Y. Yan, Y. Wang, Y. Shi, J. Mankoff, and A. K. Dey, "HulaMove: Using commodity IMU for waist interaction," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021.

[3] G. Shen, Z. Chen, P. Zhang, T. Moscibroda, and Y. Zhang, "Walkie-Markie: Indoor pathway mapping made easy," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013.

[4] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen, "Zee: Zero-effort crowdsourcing for indoor localization," in *Proceedings of the 18th annual international conference on Mobile computing and networking*, 2012.

[5] V. Prabakaran, M. R. Elara, T. Pathmakumar, and S. Nansai, "Floor cleaning robot with reconfigurable mechanism," *Automation in Construction*, vol. 91, pp. 155–165, 2018.

[6] A. Brajdic and R. Harle, "Walk detection and step counting on unconstrained smartphones," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, 2013.

[7] P. Zhou, M. Li, and G. Shen, "Use it free: Instantly knowing your phone attitude," in *Proceedings of the 20th annual international conference on Mobile computing and networking*, 2014.

[8] Q. Tian, Z. Salcic, I. Kevin, K. Wang, and Y. Pan, "A multi-mode dead reckoning system for pedestrian tracking using smartphones," *IEEE Sensors Journal*, vol. 16, no. 7, pp. 2079–2093, 2015.

[9] A. Martinelli, H. Gao, P. D. Groves, and S. Morosi, "Probabilistic context-aware step length estimation for pedestrian dead reckoning," *IEEE Sensors journal*, vol. 18, no. 4, pp. 1600–1611, 2017.

[10] J.-D. Sui and T.-S. Chang, "IMU based deep stride length estimation with self-supervised learning," *IEEE Sensors Journal*, vol. 21, no. 6, pp. 7380–7387, 2021.

[11] Q. Wang, H. Luo, L. Ye, A. Men, F. Zhao, Y. Huang, and C. Ou, "Personalized stride-length estimation based on active online learning," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4885–4897, 2020.

[12] C. Chen, P. Zhao, C. X. Lu, W. Wang, A. Markham, and N. Trigoni, "Deep-learning-based pedestrian inertial navigation: Methods, data set, and on-device inference," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4431–4441, 2020.

[13] S. Herath, H. Yan, and Y. Furukawa, "RoNIN: Robust neural inertial navigation in the wild: Benchmark, evaluations, & new methods," in *IEEE ICRA*, 2020.

[14] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: the international journal for geographic information and geovisualization*, pp. 112–122, 1973.

[15] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky, "ℓ_1 trend filtering," *SIAM Review*, vol. 51, no. 2, pp. 339–360, 2009.

[16] C. Long, R. C.-W. Wong, and H. Jagadish, "Trajectory simplification: On minimizing the direction-based error," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 49–60, 2014.

[17] S. Yang, P. Dessai, M. Verma, and M. Gerla, "FreeLoc: Calibration-free crowdsourced indoor localization," in *IEEE INFOCOM*, 2013.

[18] Y. Zhao, W.-C. Wong, H. K. Garg, and T. Feng, "Pedestrian dead reckoning with turn-based correction," in *IEEE IPIN*, 2018.

[19] T. C. Lee, "An introduction to coding theory and the two-part minimum description length principle," *International statistical review*, vol. 69, no. 2, pp. 169–183, 2001.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[21] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[22] P. S. Bullen, *Handbook of means and their inequalities*. Springer Science & Business Media, 2013.

[23] R. Bellman, "On a routing problem," *Quarterly of applied mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2022.

[25] H. Yan, Q. Shan, and Y. Furukawa, "RIDI: Robust IMU double integration," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 621–636.

[26] W. Liu, D. Caruso, E. Ilg, J. Dong, A. I. Mourikis, K. Daniilidis, V. Kumar, and J. Engel, "Tlio: Tight learned inertial odometry," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5653–5660, 2020.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[28] A. Lewkowycz, "How to decay your learning rate," *arXiv preprint arXiv:2103.12682*, 2021.

[29] Microsoft, "Microsoft geolife GPS trajectories." https://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/, 2012.

[30] N. Meratnia *et al.*, "Spatiotemporal compression techniques for moving point objects," in *International Conference on Extending Database Technology*. Springer, 2004.

[31] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. Ravi, "SQUISH: an online approach for GPS trajectory compression," in *Proceedings of the 2nd international conference on computing for geospatial research & applications*, 2011.

[32] M. Potamias, K. Patroumpas, and T. Sellis, "Sampling trajectory streams with spatiotemporal criteria," in *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*. IEEE, 2006.

[33] G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro, "On-line data reduction and the quality of history in moving objects databases," in *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, 2006.

[34] A. Conti, M. Guerra, D. Dardari, N. Decarli, and M. Z. Win, "Network experimentation for cooperative localization," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 2, pp. 467–475, 2012.

[35] A. Conti, D. Dardari, M. Guerra, L. Mucchi, and M. Z. Win, "Experimental characterization of diversity navigation," *IEEE Systems Journal*, vol. 8, no. 1, pp. 115–124, 2013.

[36] X. Wang, L. Gao, and S. Mao, "CSI phase fingerprinting for indoor localization with a deep learning approach," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1113–1123, 2016.

[37] C. Luo, H. Hong, M. C. Chan, J. Li, X. Zhang, and Z. Ming, "MPiLoc: Self-calibrating multi-floor indoor localization exploiting participatory sensing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 141–154, 2017.

[38] W. Zhuo, K. H. Chiu, J. Chen, Z. Zhao, S.-H. G. Chan, S. Ha, and C.-H. Lee, "Fis-one: Floor identification system with one label for crowdsourced rf signals," in *IEEE ICDCS 2023*.

[39] W. Zhuo, K. H. Chiu, J. Chen, J. Tan, E. Sumpena, S.-H. G. Chan, S. Ha, and C.-H. Lee, "Semi-supervised learning with network embedding on ambient rf signals for geofencing services," in *IEEE ICDE 2023*.

[40] J. Choi, G. Lee, S. Choi, and S. Bahk, "Smartphone based indoor path estimation and localization without human intervention," *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 681–695, 2020.

[41] J. Chen, B. Zhou, S. Bao, X. Liu, Z. Gu, L. Li, Y. Zhao, J. Zhu, and Q. Li, "A data-driven inertial navigation/Bluetooth fusion algorithm for indoor localization," *IEEE Sensors Journal*, vol. 22, no. 6, pp. 5288–5301, 2021.

[42] J. Dong, M. Noreikis, Y. Xiao, and A. Ylä-Jääski, "ViNav: A vision-based indoor navigation system for smartphones," *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1461–1475, 2018.

[43] A. Conti, S. Mazuelas, S. Bartoletti, W. C. Lindsey, and M. Z. Win, "Soft information for localization-of-things," *Proceedings of the IEEE*, vol. 107, no. 11, pp. 2240–2264, 2019.

[44] C. Wu, F. Zhang, Y. Fan, and K. R. Liu, "RF-based inertial measurement," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 117–129.

[45] B. Teague, Z. Liu, F. Meyer, A. Conti, and M. Z. Win, "Network localization and navigation with scalable inference and efficient operation," *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 2072–2087, 2020.

[46] X. Huang, J. Lee, Y.-W. Kwon, and C.-H. Lee, "CrowdQuake: A networked system of low-cost sensors for earthquake detection via deep learning," in *ACM KDD 2020*.

[47] W. Zhuo, Z. Zhao, K. H. Chiu, S. Li, S. Ha, C.-H. Lee, and S.-H. G. Chan, "GRAFICS: Graph embedding-based floor identification using crowdsourced RF signals," in *IEEE ICDCS 2022*.

[48] D. Chen, N. Wang, R. Xu, W. Xie, H. Bao, and G. Zhang, "RNIN-VIO: Robust neural inertial navigation aided visual-inertial odometry in challenging scenes," in *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2021.

[49] W. Ma, Q. Li, B. Zhou, W. Xue, and Z. Huang, "Location and 3-D visual awareness-based dynamic texture updating for indoor 3-d model," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7612–7624, 2020.

[50] V. Renaudin, V. Demeule, and M. Ortiz, "Adaptative pedestrian displacement estimation with a smartphone," in *IEEE International conference on indoor positioning and indoor navigation*, 2013.

[51] J. Chen, S.-h. Kao, H. He, W. Zhuo, S. Wen, C.-H. Lee, and S.-H. G. Chan, "Run, don't walk: Chasing higher flops for faster neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 12 021–12 031.

[52] B. Zhou, Z. Gu, F. Gu, P. Wu, C. Yang, X. Liu, L. Li, Y. Li, and Q. Li, "DeepVIP: Deep learning-based vehicle indoor positioning using smartphones," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 12, pp. 13 299–13 309, 2022.

[53] J. Kuang, T. Li, Q. Chen, B. Zhou, and X. Niu, "Consumer-grade inertial measurement units enhanced indoor magnetic field matching positioning scheme," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1–14, 2022.

[54] Q. Wang, H. Luo, H. Xiong, A. Men, F. Zhao, M. Xia, and C. Ou, "Pedestrian dead reckoning based on walking pattern recognition and online magnetic fingerprint trajectory calibration," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 2011–2026, 2020.

[55] Y. Wang, H. Cheng, C. Wang, and M. Q.-H. Meng, "Pose-invariant inertial odometry for pedestrian localization," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–12, 2021.

[56] A. Brajdic and R. Harle, "Scalable indoor pedestrian localisation using inertial sensing and parallel particle filters," in *2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. IEEE, 2012, pp. 1–10.

[57] C. Wu, Z. Yang, and C. Xiao, "Automatic radio map adaptation for indoor localization using smartphones," *IEEE Transactions on Mobile Computing*, vol. 17, no. 3, pp. 517–528, 2017.

[58] F. Hölzke, J.-P. Wolff, and C. Haubelt, "Improving pedestrian dead reckoning using likely paths and backtracking for mobile devices," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2019.

[59] B. Zhou, Z. Wu, and X. Liu, "Smartphone-based robot indoor localization using inertial sensors, encoder and map matching," in *IEEE International Conference on Automation, Control and Robots (ICACR)*, 2021.

[60] J. Tan, H. Wu, K.-H. Chow, and S.-H. G. Chan, "Implicit Multimodal Crowdsourcing for Joint RF and Geomagnetic Fingerprinting," *IEEE Transactions on Mobile Computing*, 2021.

[61] R. Harle, "A survey of indoor inertial positioning systems for pedestrians," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[62] Y. Lu, D. Wei, Q. Lai, W. Li, and H. Yuan, "A context-recognition-aided PDR localization method based on the hidden markov model," *Sensors*, vol. 16, no. 12, p. 2030, 2016.

[63] T. Moder, C. Reitbauer, M. Dorn, and M. Wieser, "Calibration of smartphone sensor data usable for pedestrian dead reckoning," in *IEEE IPIN*, 2017.

[64] Z. Zhang, S. He, Y. Shu, and Z. Shi, "A self-evolving WiFi-based indoor navigation system using smartphones," *IEEE Transactions on Mobile Computing*, vol. 19, no. 8, pp. 1760–1774, 2019.

[65] H. Jiang, W. Liu, G. Jiang, Y. Jia, X. Liu, Z. Lui, X. Liao, J. Xing, and D. Liu, "Fly-Navi: A novel indoor navigation system with on-the-fly map generation," *IEEE Transactions on Mobile Computing*, vol. 20, no. 9, pp. 2820–2834, 2020.

[66] Z. Wang, C. Long, and G. Cong, "Trajectory simplification with reinforcement learning," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021.

[67] Z. Wang, C. Long, G. Cong, and Q. Zhang, "Error-bounded online trajectory simplification with multi-agent reinforcement learning," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 1758–1768.

[68] X. Du, K. Yang, and D. Zhou, "MapSense: Mitigating inconsistent wifi signals using signal patterns and pathway map for indoor positioning," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4652–4662, 2018.

[69] J. Choi, G. Lee, S. Choi, and S. Bahk, "Smartphone based indoor path estimation and localization without human intervention," *IEEE Transactions on Mobile Computing*, vol. 21, no. 2, pp. 681–695, 2022.

[70] Google, "Android code snippet for orientation calculation." https://developer.android.com/develop/sensors-and-location/sensors/sensors_position#sensors-pos-orient, 2024.

[71] K. Han, S. M. Yu, S.-W. Ko, and S.-L. Kim, "Waveform-guide transformation of IMU measurements for smartphone-based localization," *IEEE Sensors Journal*, 2023.

[72] J. Gong, X. Zhang, Y. Huang, J. Ren, and Y. Zhang, "Robust inertial motion tracking through deep sensor fusion across smart earbuds and smartphone," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 2, pp. 1–26, 2021.

[73] S. Sun, D. Melamed, and K. Kitani, "IDOL: Inertial deep orientation-estimation and localization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 7, 2021, pp. 6128–6137.

[74] Y. Wang, J. Kuang, Y. Li, and X. Niu, "Magnetic field-enhanced learning-based inertial odometry for indoor pedestrian," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–13, 2022.

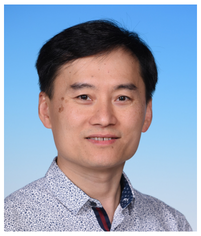
[75] T. Feng, Y. Liu, Y. Yu, L. Chen, and R. Chen, "CrowdLOC-S: Crowdsourced seamless localization framework based on CNN-LSTM-MLP enhanced quality indicator," *Expert Systems with Applications*, vol. 243, p. 122852, 2024.

Weipeng Zhuo is currently an assistant professor in the Department of Computer Science at BNU-HKBU United International College, Zhuhai, China. He received his BSc degree (double majored in Computer Science and Applied Mathematics), Master of Philosophy (MPhil) degree, and Ph.D. degree, all from The Hong Kong University of Science and Technology. During his MPhil study, he was a visiting scholar in the Department of Electrical Engineering at Princeton University. His research interests include IoT signal analytics, graph neural networks and indoor localization.

Shiju Li is a Ph.D. in Computer Engineering. He graduated from the Florida Institute of Technology, Melbourne, FL, where he also received his master's degree. He received his bachelor's degree from Huazhong University of Science and Technology, China. His research interests are in network science, data science, graph analysis, and networking. He is also interested in big data analytics acceleration, and modeling, analysis, and optimization of large-scale networked systems. His current research topic includes computational storage, large language model and graph neural network.

Tianlang He received his bachelor of engineering degree (with honor) from Donghua University, Shanghai, China, in 2018. He obtained his master of science degree from The Hong Kong University of Science and Technology (HKUST), Hong Kong, China, in 2019. He is working towards his PhD degree in the Department of Computer Science and Engineering, HKUST, Hong Kong, China. His research interest includes AIoT, Trustworthy AI, and Wireless Computing.

Mengyun Liu is currently working at the Institute of Artificial Intelligence, Guangzhou University. She received the B.E., B.B.A (Minor), and Ph.D. degrees from Wuhan University, in the School of Geodesy and Geomatics (2009-2013), Economics and Management School (2011-2013), and State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing (LIESMARS, 2013-2019), respectively. Before joining Guangzhou University, she was an Research Associate and Postdoctoral Fellow in the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST). She was also an intern in Microsoft Research Asia, and a research assistant in the Department of Land Surveying and Geo-Informatics, the Hong Kong Polytechnic University. Her research interests include positioning in GNSS-denied areas, geo-security, and the internet of things.



S.-H. Gary Chan is currently Professor of the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST), Hong Kong. He is also Affiliate Professor of Innovation, Policy and Entrepreneurship Thrust in HKUST (GZ), and Board Director of Hong Kong Logistics and Supply Chain MultiTech R&D Center (LSCM). He received MSE and PhD degrees in Electrical Engineering with a Minor in Business Administration from Stanford University (Stanford, CA). He obtained his B.S.E. degree (highest honor) in Electrical Engineering from Princeton University (Princeton, NJ), with certificates in Applied and Computational Mathematics, Engineering Physics, and Engineering and Management Systems. His research interests include smart IoT and sensing systems, location AI and mobile computing, video/user/data analytics, cloud and edge AI, technology transfer and IT entrepreneurship.

Professor Chan has been an Associate Editor of IEEE Transactions on Multimedia, and a Vice-Chair of Peer-to-Peer Networking and Communications Technical Sub-Committee of IEEE Comsoc Emerging Technologies Committee. He has been Guest Editor of ACM Transactions on Multimedia Computing, Communications and Applications, IEEE Transactions on Multimedia, IEEE Signal Processing Magazine, IEEE Communication Magazine, etc. He is a steering committee member and was the TPC chair of IEEE Consumer Communications and Networking Conference (IEEE CCNC), and has been area chair of the multimedia symposium of IEEE Globecom and IEEE ICC for many years.

Through technology transfer and entrepreneurship, Professor Chan has successfully transferred and deployed his research results in industry and co-founded several startups. Due to their innovations, commercial and societal impacts, his technologies have received numerous local and international awards. Notably, he received Hong Kong Chief Executive's Commendation for Community Service for "outstanding contribution to the fight against COVID-19" in 2020. He is the recipient of Google Mobile 2014 Award and Silver Award of Boeing Research and Technology. He was a visiting professor or researcher in Microsoft Research, Princeton University, Stanford University, and University of California at Davis. At HKUST, he was Director of Entrepreneurship Center, Director of Sino Software Research Institute, Co-director of Risk Management and Business Intelligence program, and Director of Computer Engineering Program. He was a William and Leila Fellow at Stanford University, and the recipient of the Charles Ira Young Memorial Tablet and Medal and the POEM Newport Award of Excellence at Princeton University. He is Fellow of Sigma Xi (FSX) and Chartered Fellow of The Chartered Institute of Logistics and Transport (FCILT).

Sangtae Ha is an Associate Professor in the Department of Computer Science at the University of Colorado Boulder. He received his Ph.D. in Computer Science from North Carolina State University and was an Associate Research Scholar at Princeton University from 2010 to 2013. He received the MobiSys Best Paper Awards in 2019 and 2021, the Samsung GRO Award in 2017, and the INFORMS ISS Design Science Award in 2014.

Chul-Ho Lee is currently an Assistant Professor in the Department of Computer Science at Texas State University, San Marcos, TX. Prior to that, he was an Assistant Professor in the Department of Computer Engineering and Sciences (now the Department of Electrical Engineering and Computer Science) at Florida Institute of Technology, Melbourne, FL, and a senior research engineer at Samsung Electronics DMC R&D Center (now Samsung Research), South Korea. He received his Ph.D. in Computer Engineering from North Carolina State University, Raleigh, NC. His research interests include graph mining, network science, machine learning, networking, and computing/networked systems. He has been serving on the program committees of various conferences such as IEEE INFOCOM, ACM MobiHoc, ACM KDD, IEEE ICDM, and SIAM SDM. He currently serves as an Associate Editor of IEEE Transactions on Network Science and Engineering and a Track Co-Chair of Algorithms & Theory at the 21st IEEE International Conference on Mobile Ad-Hoc and Smart Systems (MASS 2024). He has been recognized as a distinguished TPC member of IEEE INFOCOM 2018–2023 due to his excellent performance in the review process. His work was recognized as a Best Paper Award Finalist at ACM MobiHoc 2019, and he received NVIDIA Applied Research Accelerator Award in 2022.