Deep heading estimation for pedestrian dead reckoning

Jiayi Lin¹, Chengming Zou¹, Long Lan², Shanzhi Gu² and Xinshang An³

¹ School of Computer Science and Technology, Wuhan University of Technology, Wuhan, Hubei, 430070, China.
² School of Computer, National University of Defense Technology, Changsha, Hunan, 410073, China.
³ Institute of Intelligent Video Audio Technology, Shenzhen, Guangdong, China.

* Corresponding author’s e-mail: linjiayi@whut.edu.cn

Abstract. Currently, the heading estimation could be easily achieved by many built-in direction sensors, such as the smartphone. However, the obtained heading angle is only suitable for situations where an equipped pedestrian's movement and orientation keep the same, such as normal forward walking and turning. When the pedestrian faces a different direction from his movement, the heading angle remains the pedestrian orientation but not the movement and thus causes a heading estimation error. In this paper, we introduce several related deep learning techniques to explore their respective abilities in heading estimation of the waist-mounted Miniature Inertial Measurement Unit (MIMU). Specifically, this paper adopts two kinds of methods to analyze the data collected from MIMU, include acceleration, angular velocity, or their combination, to predict the heading angle. Firstly, considering the heading estimation is a time series prediction problem actually, this paper introduces the powerful Long Short-Time Memory (LSTM) model. On the other hand, this paper uses the Graph Convolutional Network (GCN) model to consider the relationship between the orientation and the direction of motion at different times. In experiments, we show that the accuracy of the proposed LSTM model in the test set achieved a promising 99.12%. To test our method in the real scenes, this paper designs simulation experiments and mobile terminal tests based on Tensorflow Lite. The experimental results show that the movement heading can be effectively judged based on the waist-mounted sensor data, and has a very significant accuracy.

1. Introduction
As the “last mile” of navigation and positioning, indoor positioning has become a research hotspot in the navigation industry. In recent years, location based services (LBS) have rapidly increased the need for indoor navigation systems. Pedestrian dead reckoning (PDR) based on inertial measurement unit (IMU) is a very important part of practical indoor navigation systems and has been extensively studied because it's convenience and wide applications[1].
The PDR system is relatively mature in practice. The PDR system based on foot-mounted can be error-corrected due to zero-velocity Update (ZUPT)[2], and the error is controllable within 5% in the 100-meter test. However, considering the convenience of wearing, users prefer the waist-mounted system. The problem is the waist-mounted system fails to correct the error calibration as ZUPT does, and loses its attractions versus the foot-mounted system in performance. Traditional PDRs are also mostly based on mathematical principles and hand-crafted feature methods. From the perspective of
MIMU, huge data could be easily obtained, therefore, it is feasible to adopt deep learning algorithms to explore the benefits of big data in this field[3].

![Image](image.png)

**Figure 1.** Heading and Movement.

**Figure 2.** The graph structure construction of sensor data.

Step counting, step length estimation, heading estimation, and position update are the four important modules of PDR[4]. Among them, the step counting, step length estimation, and position update could be readily handled and show robust performance. The only part needs to carefully pay attention to lie in the cumulative error of direction, which brings by the wrongly heading estimation and directly impacts the trajectory generation. Generally, there are two classical types of methods to estimate the heading angle. The first method focuses on the simple magnetic environment and is stably for a long time, whereas the second method addresses the heading estimation case via integrating into the complex magnetic environment and only guarantee the robustness for a short time. In the past, a lot of works have been proposed based on these two ideas to improve the accuracy of heading estimation in pedestrian dead reckoning. However, very few works have been done to study the situation of pedestrians moving laterally and backward. In the case of moving laterally or backward, the heading angle remains the same with the pedestrian's orientation and causes an estimation error. For example, the person is facing north, as shown in figure 1.

1. At this time, walking backward, as shown in figure 1(b), in practice, the heading angle should be south, but due to the problem of the person's orientation, it remains north.
2. At this time, moving to the right, as shown in figure 1(c), in practice, the heading angle should be east, but due to the problem of the person's orientation, it remains north.
3. At this time, moving to the left, as shown in figure 1(d), in practice, the heading angle should be west, but due to the problem of the person's orientation, it remains north.

The PDR system could continuously improve the accuracy of the heading angle calculation in the case of the direction of the orientation is roughly the same as the direction of movement. However, if the orientation and movement share different directions, the estimation error instantly reaches up to 90 degrees and make the entire trajectory prediction failed. Lateral and backward movements commonly happened. To handle the heading estimations of these cases, there are two main types of related works. The first type introduces velocity as the additional information since it could directly reflect the direction of movement. However, in practice, accurate velocity is hard to achieve. The second type uses principal component analysis (PCA) to extract the direction of movement from the distribution of horizontal acceleration. The problem is that it is difficult to approximate the distribution of horizontal acceleration.

With the development of artificial intelligence, deep learning technology has been widely used in computer vision[5] and natural language processing[6]. The use of deep learning techniques can simplify manual extraction and discover the rich potential features hidden by sensor data. Although
the experimental results of deep learning exceed many traditional models, it is also a huge challenge to fully exert its ability in mobile terminals due to the limited computing resource. The core of this article is to accurately estimate the heading and then calculate the heading angle on this basis. Therefore, this article attempts to use deep learning techniques, such as GCN and LSTM neural networks, to learn the angular velocity and acceleration collected by waist-mounted micro-inertial devices, and to calculate the heading angle based on the trained model. The simulation experiment shows that the waist-mounted heading judgment based on deep learning assisted heading angle calculation and achieved a very promising accuracy.

2. Related work
There were lot of researches on PDR implementation of wearable or handheld devices[7]. The most commonly used for foot-mounted is the use of ZUPT algorithm to achieve stable and reliable heading estimation[8], and for the wearing method of data without zero-velocity periodic changes, the main research direction is still optimized from the direction sensor and gyroscope[9].

- **Direction sensor.** The direction sensor consists of an accelerometer and a magnetometer built into the mobile phone. In the Android phone implementation, the rotation matrix is calculated by calling the getRotationMatrix() function on the input acceleration and magnetic sensor data, and then calling the getOrientation() function to obtain the heading angle, roll, and pitch values according to the rotation matrix. Since the function has been packaged and is easy to call, this method can realize the heading angle calculation more quickly.

- **Gyroscope.** The angular velocity can be obtained by using a gyroscope, and the angle can be obtained by integrating the diagonal velocity. Therefore, with a given initial heading angle, the angle obtained by integration can be used to calculate the next heading angle. The heading angle obtained through integration can maintain high accuracy in a short time. However, due to the drift effect of the gyroscope, long-term integration will lead to the cumulative error of the heading angle[10].

In response to the above problems, Zheng et al.[11] propose an approach to estimate the pedestrian's heading by using the rotation vector within one walking step, and analyze the correlation between the smartphone's rotational motion and the pedestrian's walking cycle states and walking direction. Deng et al.[7] propose an approach to estimate the pedestrian's heading based on principal component analysis (PCA). They obtain a more accurate estimation of the acceleration through a related rotation matrix, and then utilize PCA over the horizontal plane of acceleration signals for local walking direction extraction, and finally transform the local walking direction into the overall walking direction. Marzieh et al.[12] use machine learning to detect whether the magnetometer measurement results in the heading estimate being within the error threshold, and they explored the effect of multiple people walking on the heading angle at the same time. Considering that the magnetometer is susceptible to magnetic field interference, Kang et al.[13] combine magnetometer and gyroscope data to optimize the heading angle calculation, and afterward, a series of details were optimized[14]. Nguyen[15] uses a low-pass filter to eliminate motion noise for heading estimation and obtain good accuracy.

Existing algorithms for estimating pedestrian heading based on the smartphone [16, 17] are mostly based on the assumption of high simplification, that is, heading misalignment between device forward direction and the user heading remains constant, whether the phone is in a pocket or a handheld device. This assumption can be applied to many scenes. However, once a pedestrian walks backward or moves laterally, there will be a big error in the course estimation. The above issue motivates us to apply the waist-mounted micro-inertial device to judge the heading, thereby assisting the heading angle calculation.
3. The proposed module
In this section, we introduce two types of studied deep learning techniques, namely GCN and LSTM, to study their nonlinear fitting ability in dealing with massive data, and analyze their abilities in the heading judgment. We then detail the training settings for both the two methods.

3.1. Heading judgment model based on GCN
Graph Neural Networks (GNN) is a powerful method for representation learning on graphs, point clouds, and manifolds[18, 19]. We first introduce GCN to analyze the sensor data to estimate the heading. When a pedestrian moving continuously, the sensor collects data at a specific frequency. These data exhibit corresponding time series features, which vary with the change of motion features. The motion trajectory is shown in figure 3, where each point contains several rows of sensor data. We regard each row of sensor data as a node in a graph and connect it with the next node. As a result, these nodes and edges will constitute a graph \( G = (V, E) \), where \( V \) represents the set of sensor data recorded at each moment, and \( E \) represents the set of edges connecting adjacent sensor data.

![Graph Construction](image)

**Figure 3.** GCN calculation framework for moving sensor data.

As shown in figure 2, each node in the figure represents a row of sensor data. We set equal step length to divide the sensor data, such as each graph contains \( n \) rows of data. The graph structure is constructed from three parts:

- **graph label**: the graph is annotated with label \( y \). For example, the first graph is annotated with label 0, indicating moving backward; the second graph is annotated with label 1, indicating moving forward; the fourth graph is annotated with label 2, indicating moving right; the sixth graph is annotated with label 3, indicating moving left.

- **node_features**: we combine the features of each node to construct features for each graph and finally get the node_features in the form of matrices \( I_{n \times m} \), where \( n \) is the number of nodes, and \( m \) is the dimension of the feature.

- **edge_index**: the edges are constructed in the Pytorch-Geometric format from the starting node to the target node.

Eventually, we get a graph structure: \( Data(edge\_index = [2, n-1], x = [n, 6], y = [1]) \). Here, this paper selects the GCN network designed in [19], which is open source in PyTorch Geometric.4

---

4 https://github.com/rusty1s/pytorch_geometric/blob/master/examples/enzymes_topk_pool.py
3.2. **Heading judgment model based on stack LSTM**

An important consideration in selecting the stack LSTM is that the data generated in the process of human movement has time-series characteristics. In particular, the recurrent neural network models are better at processing sequence data. We carefully build a stacked LSTM model that takes three-axis acceleration and three-axis angular velocity as inputs. As a result, the predicted probabilities of the four labels are output. Figure 4 shows the structure of the model.

- **Input layer**: the collected data are mainly three-axis acceleration \{acc_{x}, acc_{y}, acc_{z}\} and three-axis angular velocity \{ang_{x}, ang_{y}, ang_{z}\}. Furthermore, we select the corresponding input according to the model training results of different settings.
- **Stack two LSTM**: We try to stack two layers of LSTM models to judge the heading. The experimental results show that it has excellent performance.
- **Output layer**: we use the softmax function to normalize the LSTM output to obtain the probability \(\text{pred}_{\text{softmax}}\) of heading label prediction, the calculation formulas are:

\[
\text{pred}_{\text{softmax}} = \text{soft max}(\text{pred}_{y})
\]

\[
\text{soft max}(x_{j}) = \frac{e^{x_{j}}}{\sum_{k=1}^{n} e^{x_{k}}}
\]

where \(\text{pred}_{y}\) is the output of the LSTM model.

![Figure 4. LSTM model framework.](image)

![Figure 5. Comparison of iteration accuracy of three models on the test set.](image)

4. **Experiments**

4.1. **Dataset and setups**

The method of our experiment is to place the micro-inertial device on the waist and make a waist-mounted heading judgment, figure 6 shows how to wear. We collect data by moving forward, moving backward, moving laterally to the left, and moving laterally to the right. Multiple data collections are performed in each direction, each collection time ranging from 30-45s. Moreover, the data collection frequency is 400Hz, and the acquisition objects are three-axis acceleration and three-axis angular velocity. Finally, we get 1, 498, 702 pieces of data. Table 1 shows the data distribution.

| collection methods       | collection times | total collection |
|--------------------------|------------------|-----------------|
| moving forward           | 21               | 330, 687        |
| moving backward          | 22               | 356, 951        |
4.2. GCN experiment

The main configurations of the experiment are cuda10.2 and pytorch1.4. PyTorch Geometric[20] is a deep learning library based on PyTorch[21], used for deep learning irregular input data, such as graphs, point clouds, and manifolds. Pytorch-Geometric is used to realize GCN. The optimal model is selected by comparing the accuracy and loss function.

Test one: set the data collection frequency to 200 Hz and the number of nodes in a graph to 200. We train the data collected at a frequency of 200Hz on the 11G 2080ti graphics card. We design each graph to be constructed from 200 nodes and 199 edges, which contains data for 1s movement. As a result, we construct 3, 765 graphs using the 753, 000 rows of data obtained. Furthermore, three-axis angular velocity and three-axis acceleration are selected to form a 6-dimensional feature vector.

Therefore, there are a total of 3, 765 input matrices $I_{200}$. The ratio of training set and test set is 4:1, thus, there are 3, 013 training samples and 753 test samples. We train and test the model for 2,000 epochs with batch 60 and learning rate 0.0001. The test results are shown in figure 7. We find that the model converges and the accuracy rate on the test set is 99.07%.

4.2.2. Comparative Experiment. We separately test the effect of selecting the accelerometer and angular velocity, then the node's feature vector is 3-dimensional. We plot the test results of 200-nodes and 100-nodes. As shown in figures 9 and 10, when testing a single sensor at 100-nodes or 200-nodes,
whether it is the convergence curve of the loss function or the test accuracy, the accelerometer performs better than the gyroscope. Eventually, we get the test results as shown in Table 2. The GCN has shown excellent performance in the waist-mounted heading judgment.

![Figure 7. 200-nodes model test results.](image1)

![Figure 8. 100-nodes model test results.](image2)

![Figure 9. 100-nodes test set model results.](image3)

![Figure 10. 200-nodes test set model results.](image4)

| Test Type   | Accelerometer | Gyroscope | Acc+Gyr |
|-------------|---------------|-----------|---------|
| Test Accuracy | 100-nodes     | 98.21%    | 92.56%  | 99.00% |
|             | 200-nodes     | 98.01%    | 95.61%  | 99.07% |

Table 2. GCN performance using different sensor modes. Acc means accelerometer, Gyr means gyroscope.

4.3. Stack LSTM experiment

4.3.1. Test one: raw data collected at 400hz. We train 1, 498, 702 pieces of raw data collected at a frequency of 400Hz on the 11G 2080ti graphics card. Here, we set the time steps to 400, which indicates the input matrix contains data for 1s movement. Furthermore, three-axis angular velocity is selected to form a 3-dimensional feature vector. Therefore, there are a total of 74, 920 input matrices $I_{1000}$. The ratio of training set and test set is 4: 1, thus, there are 59, 936 training samples and 14, 984 test samples. We train and test the model for 3, 000 epochs with batch 1, 024, learning rate 0.0001, and the hidden units 64. We find that the accuracy rate on the test set reaches 96.20% after the model converges.
4.3.2. Test two: row data after sampling at 200hz. We train 749, 351 pieces of row data after sampling at 200hz on the 11G 2080ti graphics card. Here, we set the time steps to 400, which indicates the input matrix contains data for 2s movement. Furthermore, three-axis angular velocity is selected to form a 3-dimensional feature vector. Therefore, there are a total of 37, 448 input matrices $I_{400\times3}$. The ratio of training set and test set is 4: 1, thus, there are 29, 958 training samples and 7, 490 test samples. We train and test the model for 3, 000 epochs with batch 1, 024, learning rate 0.0001, and the hidden units 128. We find that the accuracy of the test set reaches 99.08% after the model converges.

4.3.3. Test three: subtract the 5s data at the beginning and end of Test two. We train 585, 351 pieces of filtered row data on the 11G 2080ti graphics card. Here, we set the time steps to 400, which indicates the input matrix contains data for 2s movement. Furthermore, three-axis angular velocity is selected to form a 3-dimensional feature vector. Therefore, there are a total of 29, 248 input matrices $I_{400\times3}$. The ratio of training set and test set is 4: 1, thus, there are 23, 398 training samples and 5, 850 test samples. We train and test the model for 3, 000 epochs with batch 1, 024, learning rate 0.0001, and the hidden units 128. We find that the accuracy of the test set reaches 99.86% after the model converges.

We summarize the three test experiments, as shown in table 3. It can be observed that when training on the data after the beginning and end are filtered out for 5s, the model achieves higher accuracy in fewer iterations. This is not surprising because the corresponding action has not been performed when starting the count. However, the data is annotated with the label, resulting in four different labels for unmoved data, which affects the learning effect of the model during the training phase. Therefore, filtering out the 5s data at the beginning and end can improve the accuracy of the corresponding labels the trained model.

| Parameter attribute | Parameter settings |
|---------------------|--------------------|
| Dataset preprocessing form | Raw data (400Hz) | Interlaced save (200Hz) | Filtered out for 5s (200Hz) |
| N_EPOCHS | 3, 000 | 3, 000 | 3, 000 |
| BATCH_SIZE | 1, 024 | 1, 024 | 1, 024 |
| LEARNING_RATE | 0.0001 | 0.0001 | 0.0001 |
| N_HIDDEN_UNITS | 64 | 128 | 128 |
| N_TIME STEPS | 400(1s) | 400(2s) | 400(2s) |
| N_FEATURES | 3 | 3 | 3 |
| Step | 20 | 20 | 20 |
| Whether converges | converges | converges | converges |
| Test set accuracy | 96.20% | 99.08% | 99.86% |

We take test three as an example to analyze the results. In the convergence process, there was a large deviation of the loss function at the 631st iteration, since there were fewer training times. After the training times increased, the deviation of the test set disappeared basically. It is worth mentioning that we got 1, 295 moving forward tests on the test set, of which 1, 293 were correct; 1, 295 moving backward tests, of which 1, 291 were correct; 1, 560 moving laterally to the left tests, of which 1, 560 were correct; 1, 700 moving laterally to the right tests, of which 1, 698 were correct.

Since LSTM and CNN can be regarded as the basic branches of deep neural networks, the experimental details of CNN will not be expanded here. Meanwhile, we evaluate the accuracy provided by stacked LSTM when processing different combinations of sensor signals. It is worth noting that when the accelerometer and gyroscope move in four directions, the changes in the three-axis values are very different. Therefore, the judgment results obtained by selecting different sensor data are also different. Specifically, we find that the accuracy of the single sensor and the combined sensor on the test set exceeds 99%. This means that with the same
accuracy, we can reduce the number of sensors used and choose angular velocity data for subsequent deep learning experiments.

4.4. Models for Comparing
We analyzed and designed the LSTM model and the GCN model in detail, adjusted the network parameters, and obtained a model suitable for heading judgment. Furthermore, we trained and tested different models on the same dataset.

- Stack LSTM: We collect data at 200Hz and filter out the 6s data at the beginning and end. The step length of the input matrix is 300, i.e., the input matrix is 1.5s data each time. Here, we set the time steps to 300, which indicates the input matrix contains data for 1.5s movement. Furthermore, three-axis angular velocity is selected to form a 3-dimensional feature vector. Therefore, there are a total of 37, 650 input matrices $I_{300\times3}$. The ratio of training set and test set is 4: 1, thus, there are 30, 120 training samples and 7, 530 test samples. We train and test the model for 2, 000 epochs with batch 1, 024, learning rate 0.0001, and the hidden units 32. We find that the accuracy of the test set reaches 99.12% after the model converges.

- GCN: We collect data at 200Hz and design each graph to be constructed from 200 nodes and 199 edges, which contains data for 1s movement. As a result, we construct 3, 765 graphs using the 753, 000 rows of data obtained. Furthermore, three-axis angular velocity and three-axis acceleration are selected to form a 6-dimensional feature vector. Therefore, there are a total of 3, 765 input matrices $I_{200\times6}$. The ratio of training set and test set is 4: 1, thus, there are 3, 013 training samples and 753 test samples. We train and test the model for 2, 000 epochs with batch 60 and learning rate 0.0001. We find that the accuracy of the test set reaches 99.07% after the model converges.

- CNN: We set the data collection frequency to 200Hz. From about 180 iterations to the end, the accuracy of the CNN model remains around 97% and no longer increases. The ratio of training set and test set is 4: 1. As a result, the accuracy of the CNN model is 97.21%.

Table 4. Comparison of experimental results with different models.

| Model   | Test accuracy |
|---------|---------------|
| CNN     | 97.21%        |
| GCN     | 99.07%        |
| Stack LSTM | 99.12%       |

The final test accuracy results are shown in Table 4. As shown in figure 5, the deep learning models all exhibit excellent performance, and the potential features are perfectly learned from the trajectory of motion.

5. Scene test and analysis
We use Android Studio3.6.3 to develop the Android version of the mobile app, read real-time data of the waist-mounted sensor and the built-in sensor of the mobile phone, and use the TensorFlow Lite to package the trained model. TensorFlow Lite is an open-source deep learning framework for device inference. Lite version TensorFlow is an extended version of TensorFlow Mobile. Based on TensorFlow Lite, the application of deep learning models on the mobile terminal can be easily achieved. The flowchart for the calculation of the heading angle assisted by the waist-mounted heading judgment is shown in figure 11, where the heading angle value interval is [0, 360] degrees. Here, we mainly verify the heading angle correction model. As shown in figure 12, we exercise two laps indoors. The first lap is a normal walking lap, the pedestrian's orientation is always the same as the direction of movement. The second lap is moving forward first, then moving laterally to the right, after that moving backward, and finally moving laterally to the left.
Based on the Android database framework LitePal\(^5\), we save the necessary data such as heading angle data on the Android, upload the waist-mounted data and the Android data to the server, and draw the trajectory. The specific steps are as follows:

**Figure 11.** Flow chart of heading angle calculation algorithm assisted by heading judgment.

**Figure 12.** Schematic diagram of trajectory.

**Figure 13.** Mobile phone data collection ((a) and (b) correspond to the trajectory displayed by the mobile phone in the first and second laps respectively).

**Figure 14.** LitePal database storage ((a) and (b) correspond to the data saved on the first and second laps of the mobile phone respectively).

\(^5\) https://github.com/LitePalFramework/LitePal
(1) The first lap test. As shown in figure 13(a), we walk normally for one lap. Based on the heading angle, the trajectory is directly drawn on the mobile phone correctly. Among them, the roughly estimated step length and heading angle data are shown in figure 14(a). We have stored the step length and course required for printing each track point. At the same time, the data recorded by the waist-mounted sensor is saved, and the course judgment is performed every 1s by loading the previously trained course judgment model. As a result, the outputs of the server module are all forward. After heading angle correction is carried out according to the algorithm flow in figure 11, the motion trajectory is drawn through the roughly estimated step length as shown in figure 15(a), which is consistent with the actual motion trajectory.

(2) The second lap test. This lap contains four directions of movement. At this time, the mobile phone based on the traditional method has been unable to draw the motion trajectory correctly. Moving laterally to the left, moving laterally to the right, and moving backward are judged moving forward, the trajectory drawn on the mobile phone is shown in figure 13(b). The heading angle data is shown in figure 14(b). Here, since the step length is the same basically when walking, we set each step length to roughly 0.6m. According to the heading judgment and heading angle correction based on the server model, we draw the motion trajectory as shown in figure 15(b), which is consistent with the actual motion direction trajectory. Thus, the effectiveness of the model is verified.

![Figure 15. Server trajectory drawing ((a) is the first lap of trajectory, (b) is the second lap).](image)

6. Conclusions and future works

This paper mainly analyzes the waist sensor data with deep learning models. The pre-processed data is put into the built deep learning network model for training, and the optimal model is selected according to the accuracy and loss function. It is verified from simulation experiments that it can assist in optimizing the heading angle calculation.

In the future, we will further optimize the heading angle calculation model from three perspectives. First, although the accuracy of the heading judgment has been improved, the heading angle calculation still depends on the direction sensor. Therefore, to adapt to the complex actual environment, the heading angle model will be further optimized. Second, the experimental samples are not abundant enough in this paper, to further verify the stability of the model, we will use multiple pedestrians and mobile phones for testing. Last but not least, as mentioned at the end of the experiment, the simulation experiment obtains satisfactory results of the trajectory drawing. In order to achieve the desired accuracy in the mobile terminals, we will deeply optimize the heading judgment result based on TensorFlow lite.

References

[1] Jimenez A R, Seco F, Prieto C, et al. (2009) A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu. 2009 IEEE International Symposium on Intelligent Signal Processing, Budapest. pp. 37-42.
[2] Wang Z, Zhao H, Qiu S, et al. (2015) Stance-phase detection for zupt-aided foot-mounted pedestrian navigation system. IEEE/ASME Transactions on Mechatronics, 20(6): 3170-3181.

[3] Gu S, Guo C. (2019) Heading judgment for the waist-mounted imu using lstm. 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS). Tianjin. pp. 937-942.

[4] Diaz E M, Gonzalez A L M. (2014) Step detector and step length estimator for an inertial pocket navigation system. 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN). Busan. pp. 105-110.

[5] Lan L, Tao D, Gong C, et al. (2016) Online multi-object tracking by quadratic pseudo-boolean optimization. IJCAI. New York. pp. 3396-3402.

[6] Chung J, Kastner K, Dinh L, et al. (2015) A recurrent latent variable model for sequential data. Advances in neural information processing systems. Montreal. pp. 2980-2988.

[7] Deng Z-A, Wang G, Hu Y, et al. (2015) Heading estimation for indoor pedestrian navigation using a smartphone in the pocket. Sensors, 15(9): 21518-21536.

[8] Foxlin E. (2005) Pedestrian tracking with shoe-mounted inertial sensors. IEEE Computer graphics and applications, 25(6): 38-46.

[9] Liu X, Xu X. (2018) Hermes: Pedestrian real-time offline positioning and moving trajectory tracking system based on mems sensors. 2018 14th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN). Shenyang. pp. 7-12.

[10] Hong S K, Park S. (2008) Minimal-drift heading measurement using a mems gyro for indoor mobile robots. Sensors, 8(11): 7287-7299.

[11] Zheng L, Zhan X, Zhang X, et al. (2020) Heading estimation for multi-mode pedestrian dead reckoning. IEEE Sensors Journal.

[12] Abadi M J, Luceri L, Hassan M, et al. (2014) A collaborative approach to heading estimation for smartphone-based pdr indoor localisation. 2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN). Busan. pp. 554-563.

[13] Kang W, Nam S, Han Y, et al. (2012) Improved heading estimation for smartphone-based indoor positioning systems. 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications-(PIMRC). Sydney. pp. 2449-2453.

[14] Kang W, Han Y. (2014) Smartpdr: Smartphone-based pedestrian dead reckoning for indoor localization. IEEE Sensors Journal, 15(5): 2906-2916.

[15] Nguyen P, Akiyama T, Ohashi H, et al. (2016) User-friendly heading estimation for arbitrary smartphone orientations. 2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN). Madrid. pp. 1-7.

[16] Pei L, Chen R, Chen Y, et al. (2009) Indoor/outdoor seamless positioning technologies integrated on smart phone. 2009 First International Conference on Advances in Satellite and Space Communications. Colmar. pp. 141-145.

[17] Roy N, Wang H, Roy Choudhury R. (2014) I am a smartphone and i can tell my user's walking direction. Proceedings of the 12th annual international conference on Mobile systems, applications, and services. Boston. pp. 329-342.

[18] Bronstein M M, Bruna J, LeCun Y, et al. (2017) Geometric deep learning: Going beyond euclidean data. IEEE Signal Processing Magazine, 34(4): 18-42.

[19] Kipf T N, Welling M. (2016) Semi-supervised classification with graph convolutional networks. https://arxiv.org/abs/1609.02907.

[20] Fey M, Lenssen J E. (2019) Fast graph representation learning with pytorch geometric. https://arxiv.org/abs/1903.02428.

[21] Paszke A, Gross S, Chintala S, et al. (2017) Automatic differentiation in pytorch.