Accurate Orientation Estimates for Deep Inertial Odometry

Scott Sun

July 2020
CMU-RI-TR-20-29

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Thesis Committee
Prof. Kris Kitani, *Chair*
Prof. Michael Kaess
Ye Yuan

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Many smartphone applications use inertial measurement units (IMUs) to sense movement, but the use of these sensors for pedestrian localization can be challenging due to their noise characteristics. Recent deep inertial odometry approaches to pedestrian navigation have demonstrated the increasing feasibility of inertial navigation. However, they still rely upon conventional phone orientation estimates that they assume to be accurate, while in fact these orientation estimates can be a significant source of error. To address the problem of inaccurate orientation estimates, we present a data-driven inertial localization pipeline that performs both device orientation and position estimation using a commodity smartphone. Our system first estimates device orientation via a deep recurrent neural network component, which is coupled with raw IMU measurements to estimate device positions via a second deep network component. To improve the robustness of the orientation estimates, we introduce a dynamic magnetometer calibration network and implement an Extended Kalman Filter to generate accurate orientation estimates from the calibrated magnetic field. Our proposed model outperforms state-of-the-art methods by 40% in orientation error and up to 50% in position error on a large dataset we constructed that contains 20 hours of pedestrian motion across 3 large buildings and 15 subjects.

Acknowledgements

I would like to start by thanking my advisor, Kris Kitani. Without the research environment he helped foster, the generous compute, funding, and free food he provided, and the insightful meetings we were able to have, this research would not have been possible. His support through all the ups and downs of this project has been unwavering.

Furthermore, I am grateful for all the help I received from the members of KLab, who have provided guidance and volunteered their time to help construct the dataset used in this work. Of these members, I must especially thank Dennis Melamed for his help running experiments and brainstorming solutions to the various hurdles we've come across over the course of this project. I would like to thank the members of my thesis committee: Kris Kitani, Michael Kaess, and Ye Yuan, for generously offering their time and feedback.

Lastly, I would like to thank my family and friends. Their support through this challenging period has kept me sane and motivated. Without their support, things would certainly have been much less enjoyable.

Contents

List of Figures

List of Tables

1 Introduction

One of the weaknesses of inertial localization (inertial odometry, dead-reckoning) is the dependence on having an accurate 3D orientation estimation technique to properly convert local inertial measurements of a device to a global reference frame. Small errors in this component can result in substantial errors. Inertial localization techniques, including recent deep-learning-based techniques [1, 2], estimate 3D motion from device-frame inertial measurement unit (IMU) samples of linear acceleration, angular velocity, and magnetic flux density. These measurements must interface with some device orientation estimate (e.g., roll, pitch, yaw; quaternion; rotation matrix) to convert to absolute device motion in the world frame. This dependency means that any error in the device's orientation estimate will be compounded in the final location estimate. Since orientation estimation plays such an important role in inertial localization, we hypothesize that improvements in the 3D orientation estimate will result in greater gains in localization performance.

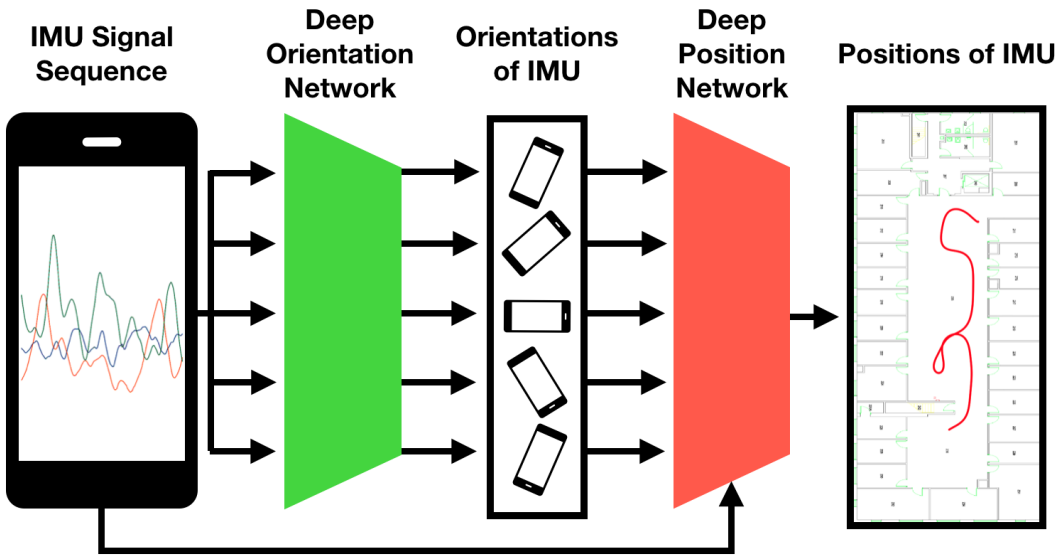


Figure 1: System overview: IMU sensor data from a smartphone is processed through a recurrent neural network to output the 3D orientation and the relative 2D position of the smartphone.

Despite the many challenges of inertial localization, IMUs continue to be used as an element of many mobile applications because they are infrastructure-free, low-

power, low-cost, and non-invasive. WiFi or Bluetooth beacon-based solutions [3] can be cost effective but require heavy instrumentation of the environment for accurate localization (<1.5 meter error). Lidar-based localization is highly accurate [4] but is expensive and requires significant power. Image-based localization is also effective when there is ample light and texture, but can also be perceived as invasive in private spaces. Inertial solutions are highly energy-efficient, do not require line-of-sight with the environment, and are highly ubiquitous by function of their cost and size. Due to these favorable characteristics, if one could obtain more accurate inertial localization, it would immediately benefit a wide variety of mobile applications.

Recent deep-learning approaches, like IONet [1] and RoNIN [2], have demonstrated the possibility of estimating 3D device (or user) motion using an IMU, but they do not address the task of device orientation estimation. State-of-the-art deep-learning-based approaches are able to address the drift problem of inertial localization techniques through the use of supervised learning to directly estimate the spatial displacement of the device. However, all existing approaches use the 3D orientation estimates generated by the device (typically with filtering-based approaches) which can be very inaccurate (>20 degrees of error from our experiments with Apple's iPhone 8). The 3D orientation is typically used as an initial step to rotate the local IMU measurements to a common reference frame before applying a deep network [1, 2]. This is problematic as the robust output of a deep network can be corrupted by errors in a device's raw 3D orientation estimate and can accumulate a significant amount of error over time.

While it might be possible to overcome this issue with a brute-force data-driven approach with vast quantities of data, we advocate for a more principled approach. We design an end-to-end deep network architecture which includes an explicit orientation estimation module, shown in Figure 1. The 3D orientation estimation module makes use of orientation information encoded in the accelerometer and magnetometer of the IMU, as they provide external measurements of gravitational acceleration and the earth's magnetic field. By looking at a small temporal window of IMU measurements, the orientation network learns to estimate a more accurate 3D device orientation, and by extension a more accurate 3D device location. We jointly learn the orientation network and a position estimation network using supervised learning to estimate the 5D pose (3D orientation and 2D position) of the device.

In our experiments, we show that by first estimating an accurate 3D orientation,

the accuracy of position estimation is significantly improved. We demonstrate that our orientation network outperforms the iOS CoreMotion API's estimate of current device orientation and that our end-to-end model produces position estimates that outperform those of other data-driven and traditional approaches. We evaluate the proposed model on a dataset collected using a smartphone IMU and a LiDAR+visual SLAM system (Kaarta Stencil) for ground truth position estimation. Volunteers walked around an office space while carrying the data collection rig for 20 hours in total across three buildings.

The contributions of our work are: (i) a data-driven estimation model able to accurately predict 3D device orientation, which we show leads to improved position estimates; (ii) a model for higher accuracy position estimation than previous classical and learning-based techniques; and (iii) the first large-scale dataset of over 20 hours of annotated 3D device orientation (different from user heading) and 2D position (3D is a trivial extension).

2 Related Work

In the following, we present related works for both position and orientation estimation.

2.1 Position Estimation

We place inertial navigation systems into two broad categories based on their main approach to the problem of localization: either traditional filtering methods or data-driven neural network approaches.

Traditional Methods: Dead reckoning with an IMU using the analytical solution consists of integrating gyroscopic readings to determine sensor orientation (through something like Rodrigues' rotation formula), using those orientations to rotate accelerometer readings into the global frame, removing gravitational acceleration, and then double-integrating the corrected accelerometer readings to determine position [1]. The multiple integrations lead to errors being magnified over time, resulting in an unusable estimate within seconds.

Additional system constraints on sensor placement and movement can be used in order to reduce the amount of drift. A commonly used velocity constraint for foot-

mounted inertial navigation systems is the Zero-speed UPdaTe (ZUPT) [5]. This takes advantage of the fact that the foot velocity is zero when it is in contact with the ground. Pedestrian dead reckoning (PDR) is a class of techniques that adopts a biomechanical model of the human gait. Typically this involves estimating when a user takes a step, the length of that step, and the user heading (the walking direction) during that step. This results in linear error growth as a function of the number of steps, but relies on a good device orientation estimate to accurately regress user heading. As such, PDR tends to only perform well in controlled environments [5].

Because of the problem of error accumulation, inertial localization tends to serve a supporting role in sensor fusion systems where environmental measurements are available to correct for drift. Extended Kalman Filters are often used to combine the short-term accurate results of IMU navigation with other localization methods that are more accurate over the long term. These can include GPS [6] and cameras [7].

Neural Network Methods: IONet [1] presented one of the first systems which uses a learned neural network component to attempt to solve the issue of pedestrian inertial navigation without relying explicitly on constraints on sensor location or motion. A bi-directional long-short-term memory (BiLSTM) network is given a window of world-frame accelerometer and gyroscope measurements (with the reference frame conversion done by the phone API), from which it sequentially directly regresses a polar displacement vector describing the device's motion in the x-y plane (which maps to the ground in this case). The bulk of their work is evaluated in a small Vicon motion capture studio with different walking modalities, e.g., in the hand or in the pocket. They define their loss function as follows:

$$\mathcal{L} = \sum \|\Delta\ell - \Delta\hat{\ell}\|_2^2 + \lambda\|\Delta\theta - \Delta\hat{\theta}\|_2^2, \tag{1}$$

where $\Delta\ell$ is the distance traveled and $\Delta\theta$ is the change in heading angle over the course of a window (which they define as 2 seconds). The hat notation denotes the output of the network, which is the 2-dimensional vector $(\Delta\ell, \Delta\theta)$ returned at the last timestep of each window by the BiLSTM. They also rely on a λ factor to weight between the different units in their polar representation. A recent update to their work also provides an estimate of the uncertainty of the network predictions [8] by predicting the mean and variance of a Gaussian distribution.

RoNIN [2] focuses on using three different neural network architectures to regress user velocity estimates in the x-y plane. These estimates are then integrated only once to determine user position, minimizing the amount of error magnification. The three architectures presented are: a ResNet convolutional neural network, a bi-directional LSTM network, and a temporal convolutional network (TCN). This work also reframes the problem of inertial localization slightly, moving from attempting to regress the sensor's position to regressing the user's position. This is because their ground truth is collected using a separate visual-SLAM system attached to the body. Their data is collected throughout several buildings, which is a more realistic scenario than in IONet. For their ResNet architecture, the loss function is defined as MSE loss computed over strided windows of 2 seconds. Mathematically, this is expressed as

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|\Delta x_i - \Delta \hat{x}_i\|_2^2 \tag{2}$$

where Δx_i is the displacement over the window starting at time i. The hat notation denotes the output of the network. In this case, each window results in a 2-dimensional displacement vector. For the LSTM and TCN networks, they use MSE loss computed between the integral of all predicted values in a window and the actual displacement over that window. This is done because the network produces an output for each timestep of a window. Summing them up reduces the entire window to a single displacement. Mathematically, this is expressed as

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|\sum_{j=i}^{i+t} x_j - \sum_{j=i}^{i+t} \hat{x}_j\|_2^2 \tag{3}$$

where i denotes each window and j denotes each output of the network in that window of length t.

TLIO [9] is a recent work that uses a ResNet-style architecture to estimate positional displacements from RoNIN. They then fuse these deep estimates with the raw IMU measurements using an Extended Kalman Filter, which is then also capable of estimating and correcting for the sensor biases. They estimate position and orientation as part of the state in their EKF.

The present work suggests that a feature lacking in previous attempts is maintaining a robust estimate of device orientation. Previous networks rely heavily on the mobile operating system's estimate of device orientation which fuses magnetometer and gyroscope readings using mostly classical methods (although specific

implementation details are closed source). While these estimates may be accurate over the short term, they are prone to discontinuities, unstable readings, and drift over time. By regressing a more accurate device orientation and providing it to a position estimation network, our system produces more robust, accurate results than previous models.

2.2 Orientation Estimation

Prior work for device orientation estimation are primarily based on traditional filtering techniques. The Madgwick filter [10] is used widely in robotics applications. In the Madgwick filter, gyroscope readings are integrated over time to determine an orientation estimate. This is accurate in the short term but begins to drift over longer periods due to gyroscope bias. In order to correct the bias, a minimization is performed between two vectors: (1) the static world gravity vector, rotated into the device frame using the current estimated orientation, and (2) the acceleration vector from the accelerometer. The major component of the acceleration vector is assumed to be gravity, so a gradient can be calculated which brings the static gravity vector in the estimated orientation frame closer to the acceleration vector in the current frame. A weighted combination of this gradient and the gyroscope integration is used to update the quaternion orientation estimate of the device. The Madgwick filter relies on the non-gravitational acceleration components being small, which is an impractical assumption for pedestrian motion.

Complementary filters are also used in some state-of-the-art orientation estimation systems like MUSE [11]. MUSE behaves similarly to the Madgwick filter, but uses the acceleration vector as the target of the orientation update only when the device is static. Instead, the magnetic north vector is mostly used as the basis of the gradient calculation. This has the advantage of removing the issue of large non-gravitational accelerations causing erroneous updates since when the device is static, the acceleration vector consists mostly of gravity. However, a static device is a rare occurrence during pedestrian motion and magnetic fields can vary significantly from location to location due to local disturbances which are difficult to characterize. Our work proposes a solution to this problem by utilizing a dynamic calibration for the local magnetic field based on the history of previous device orientation estimates.

Extended Kalman Filter (EKF) approaches [12, 13, 14] follow a similar approach to the previously mentioned filters, but use a more statistically rigorous method of

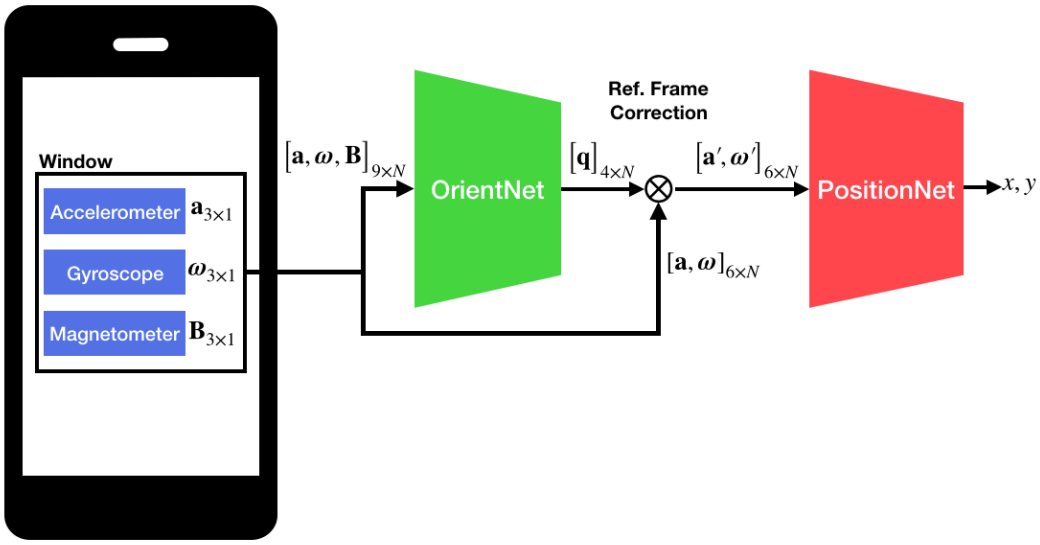


Figure 2: Overall system diagram

combining gyroscope propagation and accelerometer/magnetometer updating. An estimate of the error of the orientation estimate can also be extracted from this type of filter. We take advantage of such a filter in our work, but replace the gravity vector or magnetic north measurement update with the output of a learned model to provide a less noisy estimate of the true orientation and simplify the Kalman update equations.

3 Method

We aim to develop a method for 3D orientation and 2D position estimation of a smartphone IMU held by a pedestrian through the use of supervised learning. Our model is designed based on the knowledge that the accelerometer contains information about the gravitational acceleration of the earth and that the magnetometer contains information about the earth's magnetic field. Thus, it should be possible to infer the absolute 3D orientation of the device with a high-capacity regression model (a deep network) with higher accuracy than that of traditional filtering methods. In the following, we describe a unified pipeline for 3D orientation and position estimation. A broad overview is presented in Figure 2. We first present our model

for 3D orientation estimation and then describe the position estimation model. A detailed system diagram is shown in Figure 3.

3.1 Estimating 3D Orientation

The architecture of the orientation network (Figure 4) is designed to produce a better estimate of phone orientation than that provided by the iOS CoreMotion API. The inputs to the network are the 3 dimensional local frame angular velocity measured by the gyroscope, the 3 dimensional local frame linear acceleration measured by the accelerometer, and the 3 dimensional magnetic field measured by the magnetometer. The network consists of 3 structural components: (1) a magnetic correction network to dynamically calibrate the magnetometer values; (2) an orientation regression network that estimates a device orientation from the provided acceleration, angular velocity, and scaled magnetometer readings; and (3) an Extended Kalman filter to fuse the gyroscope readings with the network for additional stability in the estimate. The resulting 3D orientation is used to rotate the accelerometer and gyroscope channels from the phone's coordinate system to a world coordinate system. The corrected measurements are then passed as inputs to the position network.

Representation: The choice of representation for 3D orientations $\vec{\theta}$ (e.g., Euler angles, quaternions) can have a significant impact on the learning difficulty [15]. To represent the orientation, we use a 6-dimensional reparameterized formulation of Euler angles where we take the sine and cosine of each angle. We do this in order to remove discontinuities from the representation of orientation. Quaternions avoid the discontinuity problem, but suffer from the double-cover problem [16], which also degrades network performance. Furthermore, limiting quaternions to eliminate the double-cover problem reintroduces discontinuities. While we could directly regress 3×3 rotation matrices, our output remains more compact and results in better network performance than other parameterizations we tried.

Magnetic Correction Network: In a world with no magnetic disturbances, the raw magnetic field can be used to uniquely determine 3D orientation. The measurements of the magnetic field would lie on a unit sphere and each point on that sphere would correspond to a 3D orientation of the device. For example, in a disturbance-free world, a device with its x-axis oriented towards the north pole would result in

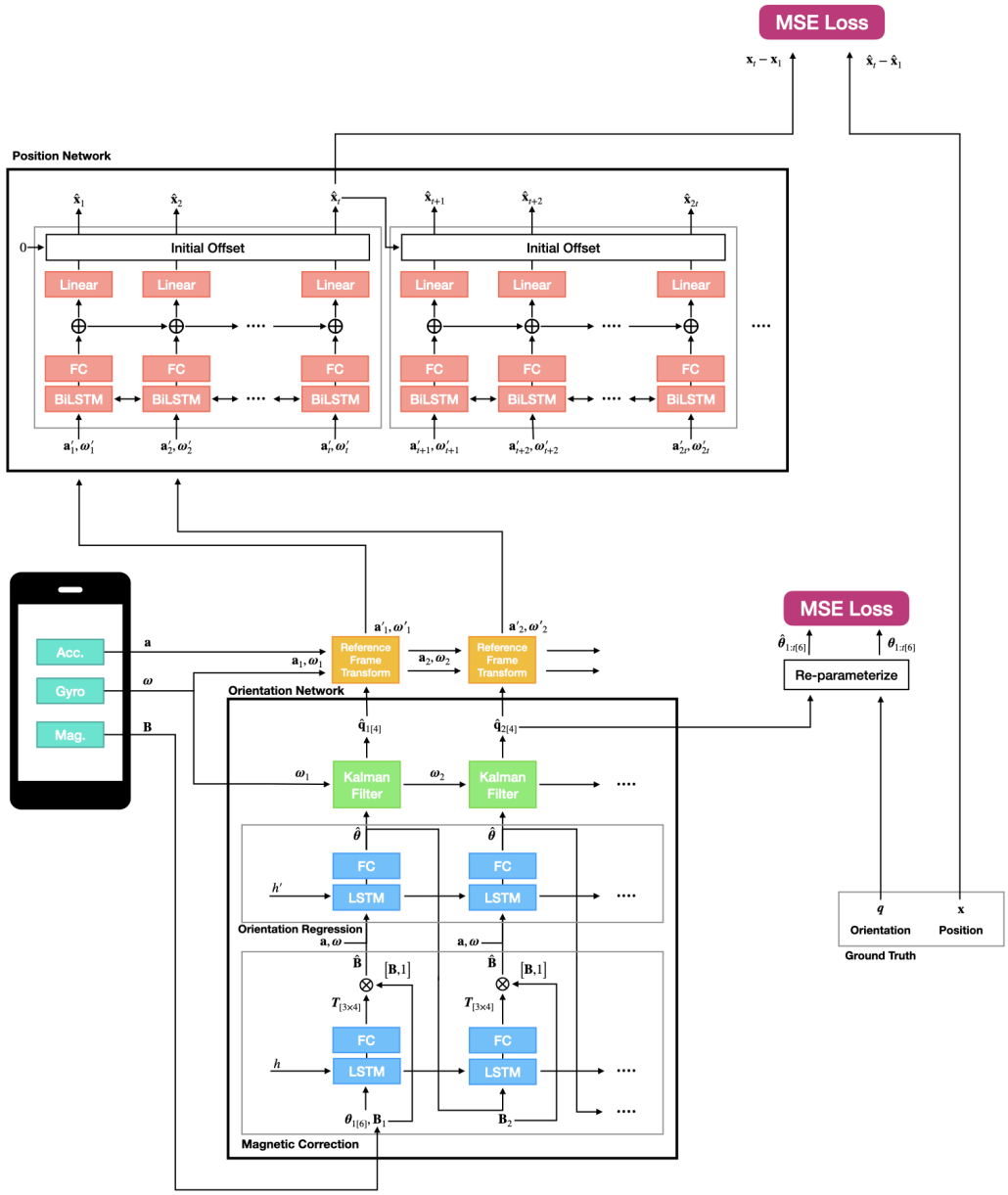


Figure 3: Detailed system diagram. The IMU readings are first passed through the orientation network, which is trained to estimate the device orientation quaternion q. For better training, we re-parameterize quaternions as the sine and cosine of the Euler angle representation (6 dim). This orientation is used in the reference frame transform that converts accelerometer and gyroscope readings from device to world frame. The position network takes these and outputs the final position estimate. The position network is trained to minimize displacement error over each window.

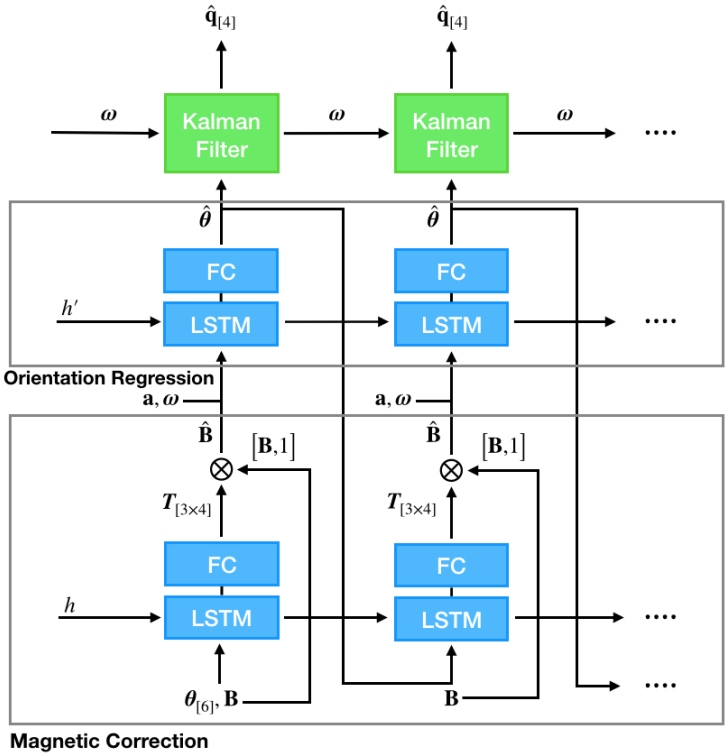


Figure 4: Orientation Network architecture

a calibrated magnetometer reading of $\hat{\mathbf{B}} = [1\ 0\ 0]^T$. Any rotation applied to the device can be expressed as a rotation of this unit vector: $\mathbf{R}\hat{\mathbf{B}}$.

Unfortunately, most modern environments have a variety of magnetic disturbances which can cause extreme changes in the magnetic field. The traditional procedure for calibration of a magnetometer to correct for these disturbances involves performing an ellipsoid fit on the 3-axis magnetometer values and scaling the values to the unit sphere [17]. However, this is not a robust procedure when encountering highly variable magnetic disturbances (such as in a building) because if the user moves elsewhere, the calibration will have to be repeated. Therefore, in order to use the raw magnetic field to estimate device orientation, we must be able to dynamically calibrate (adjust values of) the measured magnetic field to map it to the ideal magnetic field. Fortunately, these magnetic disturbances have been observed to be fairly consistent over time [18], so it is possible that a model can be learned to map raw magnetic field measurements to ideal measurements given enough data.

The intuition is that by looking at a small temporal window of magnetic field measurements (input) and ideal magnetic field values (output), the network can learn a dynamic correction for the magnetic field that can account for magnetic disturbances.

Formally, we compute the corrected magnetic field $\hat{\mathbf{B}}$ by learning the transformation $\mathbf{T}$:

$$\hat{\mathbf{B}}_t = \mathbf{T}(\vec{\theta}_{t-1}, \mathbf{B}_t, \vec{h}_{t-1}) \begin{bmatrix} \mathbf{B}_t \\ 1 \end{bmatrix}, \tag{4}$$

where $\mathbf{B}_t$ is the measured magnetic field, $\vec{\theta}_{t-1}$ is the estimated 3D orientation from the last time step, $\vec{h}_{t-1}$ is a sufficient statistic of the history of measurements and $\mathbf{T}$ is a 12DOF transformation matrix. To dynamically correct the raw magnetic field measurements, we use an LSTM to regress $\mathbf{T}$, a 3×4 transformation matrix (3D rotation, translation, scale, skew), that transforms the three raw magnetometer measurements $\mathbf{B}$ to calibrated measurements $\hat{\mathbf{B}}$ that lie on a unit sphere. In particular, we use a 2-layer unidirectional LSTM with 100 hidden units to regress the transformation $\mathbf{T}$. The LSTM hidden state $\vec{h}_t$ at each time-step is fed to 2 fully-connected layers (hidden states of size 40 and 12) with a tanh activation in between to obtain the values of $\mathbf{T}$.

Figure 5 shows the resulting corrected magnetometer values (blue) compared to the magnetometer measurements after the ellipsoid calibration (orange). Notice that the field deviates significantly throughout a building but is mapped to a unit sphere using our magnetic correction network. The points from this particular trajectory create an unfilled circle when projected to the XY plane due to the phone screen being oriented up most of the time, which results in the points being confined to a sliver along the sphere's equator (i.e., an unfilled circle) as expected. One can visualize this by imagining the phone is pointing along the vector from the origin to each point on the unit sphere. When comparing individual points between the magnetic correction and simply projecting the measurements to the unit circle, we notice that the naive projection results in values that are not in the correct position on the sphere compared to the network result. This suggests that the magnetic correction network is learning something more than just a simple re-normalization.

Figure 6 shows how the magnetic correction network functions in the presence of an artificial disturbance. The device is kept static and a piece of steel is brought

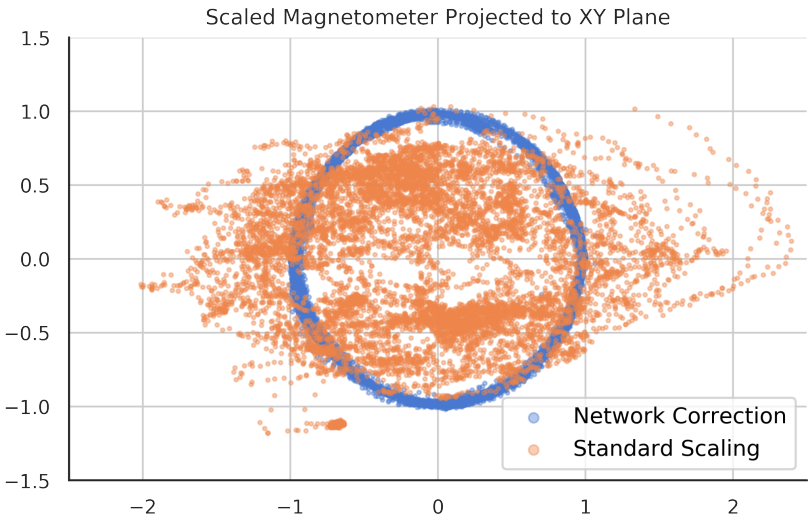


Figure 5: Using only a traditional ellipsoid-fit sensor calibration at a single location, the observed magnetometer values quickly become distorted from the unit circle when walking around a building with the phone facing up. The network learns a dynamic correction that is able to project the values back to the correct spots on the unit circle.

up to the device, then removed. The network is able to continue to provide a stable orientation estimate, while the iOS CoreMotion estimate drifts but then recovers.

Orientation Regression Network: A second network is used to convert the corrected magnetic field and other IMU measurements to a 3D orientation. Instead of directly converting the magnetic field or acceleration vector to orientation, as is done in traditional filtering methods, we again use a neural network to learn the mapping from corrected magnetic field and other sensor measurements to orientation in a data-driven way.

Formally, we estimated the instantaneous 3D orientation

$$\hat{\vec{\theta}} = g(\mathbf{a}_t, \vec{\omega}_t, \hat{\mathbf{B}}_t, \vec{h}'_{t-1}), \tag{5}$$

where the function g consists of a 2-layer Bi-LSTM with 100 hidden units and $\vec{h}'_{t-1}$

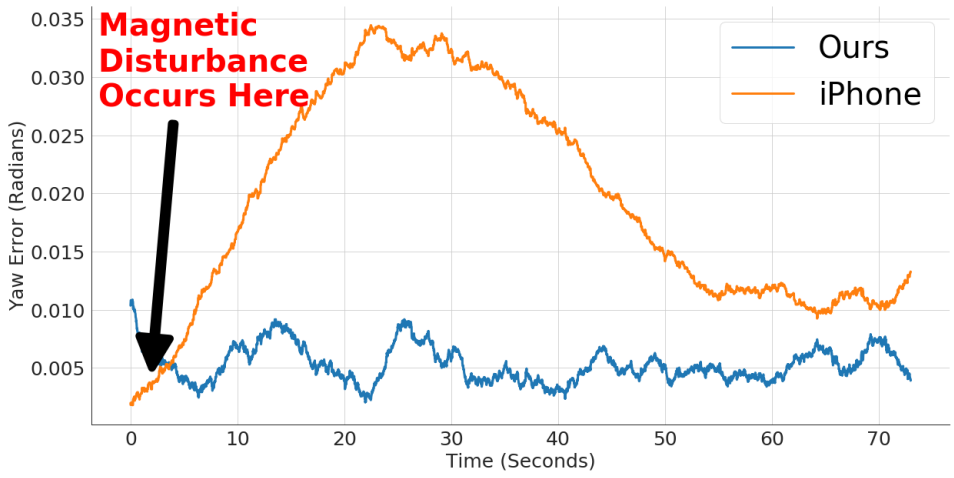


Figure 6: Under stationary conditions, an artificial disturbance to the magnetic field (a piece of steel) is introduced. The error of our network remains stable compared to the iOS CoreMotion estimate, which abruptly increases temporarily.

is a sufficient statistic (hidden state) produced by the LSTM at the last time step. At each time step, the gyroscope and accelerometer sensor readings ($\mathbf{a}, \vec{\omega}$ plus the corrected magnetic field $\hat{\mathbf{B}}$) are taken as input. The hidden state is then fed through 2 fully-connected layers to produce an absolute orientation $\hat{\vec{\theta}}$ in the global reference frame. In the position estimation network described later, this orientation estimate will be used as a coordinate transform to rotate the IMU channels from the local frame of the phone to the global reference frame.

Gyroscope Fusion EKF: Using the raw gyroscope measurements (which represent a rotation over time) as a process update and the outputs of the orientation estimation network (which represent an orientation) as a measurement update in a Kalman filter, we can further stabilize the network output. Since both gyroscope readings and orientation estimates can readily be expressed as quaternions, we can rely on an Extended Kalman Filter (EKF) using quaternion orientations as states [12]. Because the measurements are the same as the states, the measurement equations are linear; the process equations, however, are still nonlinear.

A basic Kalman Filter has two steps: the process update and the measurement update. Process updates usually use a motion model to propagate the previous state to the current state, given some control commands or in our case the angular

velocities given by the gyroscope. Thus, we state the propagation equations for the EKF as:

$$\vec{\hat{x}}_{k|k-1} = \vec{A}_k \vec{\hat{x}}_{k-1|k-1} + \vec{B}_k \vec{\omega}_k, \tag{6}$$

$$\vec{P}_{k|k-1} = \vec{A}_k \vec{P}_{k-1|k-1} \vec{A}_k^T + \vec{Q}_k, \tag{7}$$

where $\vec{\hat{x}}_k$ is the state vector consisting of the quaternion orientation estimate (4x1) and the gyroscope bias estimate along each axis of the gyroscope (3x1) at timestep k. The output of the process update is $\vec{\hat{x}}_{k|k-1}$, or the estimated state vector given the previous estimated state vector. The motion model is parameterized by matrices $\vec{A}_k$ and $\vec{B}_k$ which depend on the current orientation estimate. $\vec{A}_k$ is a matrix which applies the current gyroscope bias vector to correct the current quaternion portion of the state vector. $\vec{B}_k$ takes the current gyroscope measurement $\vec{\omega}_k$ and converts it into a quaternion representing the rotation achieved by $\vec{\omega}_k$ over the sample time. The process update is applied via simple addition, which is a sufficient approximation to quaternion rotations due to our high sample rate based on our testing. $\vec{P}_{k|k-1}$ is the estimate of the covariance of the state vector given the previous covariance estimate, and is calculated by left and right multiplying by $\vec{A}_k$ and its transpose to propagate the previous estimate, then adding $\vec{Q}_k$, an estimate of the covariance matrix of the noise of propagation. For our filter we tune a static $\vec{Q}_k$ on our training data.

Measurement updates involve correcting the propagated state with outside information, in this case the orientations predicted by our model. Formally, the update equations are

$$\vec{K}_k = \vec{P}_{k|k-1}(\vec{P}_{k|k-1} + \vec{R}_k), \tag{8}$$

$$\vec{\hat{x}}_{k|k} = \vec{\hat{x}}_{k|k-1} + \vec{K}_k(\vec{q}_k - \vec{\hat{x}}_{k|k-1}), \tag{9}$$

$$\vec{P}_{k|k} = (\vec{I} - \vec{K}_k)\vec{P}_{k|k-1}, \tag{10}$$

where $\vec{R}_k$ is the estimated covariance of the measurements, in our case again a static matrix tuned on training data. The result of the measurement update is $\vec{\hat{x}}_{k|k}$ and $\vec{P}_{k|k}$, or the final estimate of the state vector and its covariance based on both the previous state vector and the latest measurement.

Because this Kalman update procedure is differentiable, we are able to back-propagate through this filter when training the entire network end-to-end, which

also allows us to derive process and measurement covariance matrices via backpropagation on the training set.

Training: To train the orientation network, we first perform a traditional ellipsoid fit correction on the raw magnetometer values in order to scale the inputs to the network to a confined range. From here on, we will refer to these coarsely calibrated magnetometer readings as part of the raw IMU measurements. The next step is to pre-train just the orientation regression network using raw IMU measurements as inputs. Here, we just use MSE loss with the ground truth orientations expressed as 6-dim reparameterized Euler angles. Then, the magnetic correction component is attached and both networks are trained together. Here we use a combined loss with the MSE of device orientations and the MSE of corrected magnetometer values versus ground truth magnetometer values as follows:

$$\mathcal{L}_{\text{orient}} = \mathcal{L}_{\text{MSE}}(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}) + \lambda \mathcal{L}_{\text{MSE}}(\mathbf{B}, \hat{\mathbf{B}}). \tag{11}$$

Since these loss components are of different units, a λ factor should be used to weight them, but we find $\lambda = 1$ works fine in practice. During training, we use 2-second (200 timestep) sequence length windows with a stride of 10 samples.

3.2 Estimating Position

Analytically, in order to convert from the world frame accelerometer values (computed by rotating the raw accelerometer values by the orientation estimate) to position, one would need to perform two integrals. However, any offsets in the acceleration values would result in quadratic error growth as a result. Therefore, we again adopt the use of a neural network to learn an appropriate approximation that is less susceptible to error accumulation, as has been demonstrated successfully by [2, 1]. The position estimation network takes world frame IMU channels as input and outputs the final user position in the same global reference frame as the orientation network. We opt for a Cartesian parameterization of the user position to match that of the rotated accelerometer.

The network architecture is depicted in Figure 7. We primarily rely on a 2-layer bi-directional LSTM with a hidden size of 100. The input is a sequence of 6-DOF IMU measurements that have been rotated into the world frame. At each timestep,

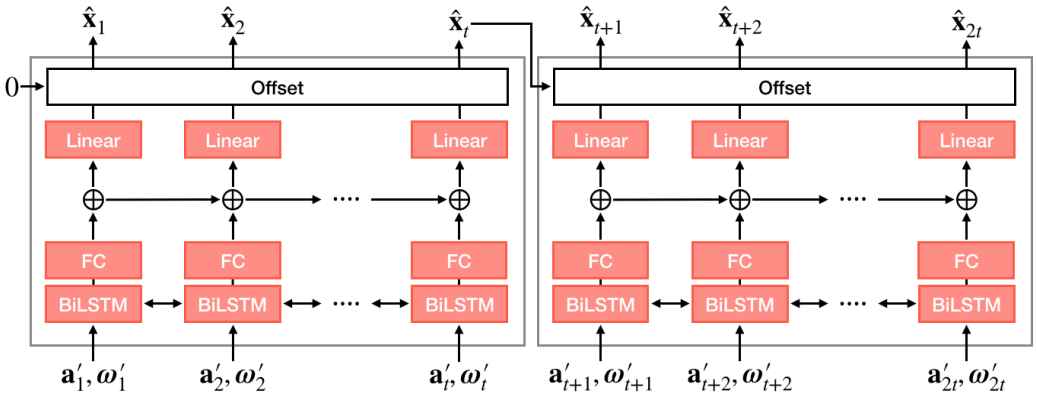


Figure 7: Position Network architecture

the hidden state is then passed to 2 fully-connected layers with tanh activation between them and hidden state sizes of 50 and 20, respectively. The resulting vector is accumulated via a summation over time. At each timestep, the resulting vector is then passed to a linear layer that converts it to the current two-dimensional Cartesian position relative to the start of the sequence in a similar manner as [2]. During test time, in between each sequence window, the LSTM hidden states are not propagated, as this periodic resetting helps to limit the accumulation of drift. Instead, the output at the last timestep is added as an offset to all the outputs in the next window in order to continuously accumulate the location.

Training: Over the course of training, progressively longer batch sequences are provided. We start with sequences of length 200 and progressively increase this over training to length 4000. We find this type of curriculum learning greatly reduces drift accumulation as the overall error must be kept low over a longer time period. For example, the absolute trajectory error (defined in Section 5), when evaluating the converged model on the validation set, reaches 10.05m when trained only on sequences of length 200 vs 5.08m for this technique. Training only on the longer sequence length of 4000 proves more difficult to learn; as a result, network performance suffers. After this routine, the sequence length may then be dropped back down to a shorter length to reduce latency. We use MSE loss over the displacement of each LSTM window. In other words,

$$\mathcal{L}_{\text{position}} = \mathcal{L}_{\text{MSE}}(\mathbf{x}_t - \mathbf{x}_1, \hat{\mathbf{x}}_t - \hat{\mathbf{x}}_1)\,. \tag{12}$$

4 Datasets

4.1 Small-scale Motion Capture (Mocap) Dataset

One of the most accurate methods for measuring the exact position and orientation of an object is the use of a motion capture studio, whereby the object is covered with retro-reflective markers that are tracked by multiple cameras positioned throughout a room. These systems typically have centimeter-level positional accuracy, which is well below the error level we require. We collected 1 hour of data in a $4m \times 2m$ Vicon studio with a single subject. Because of the small space and small quantity of data, we primarily use this dataset for evaluating orientation performance. This dataset involves a combination of walking, jogging, and standing while the phone is constantly rotated in the hand during motion (swapping from portrait to landscape mode, rotating the wrist, etc.). This constant motion makes orientation especially difficult to estimate. Data is collected from an Xsens MTi-20 IMU attached to phone case that is directly held in the hand; we collect accelerometer, gyroscope, and magnetometer channels at 100Hz in addition to the Xsens's built-in orientation estimate. Trajectories in this dataset are about 2-3 minutes in length.

4.2 Building-scale Trajectory Dataset

One of the fundamental drawbacks of relying on motion capture data is the difficulty in scaling to larger environments similar to those one might typically expect to traverse. To collect trajectories through the narrow hallways of a typical building, we rely on LiDAR+visual-based SLAM rig as ground truth instead. In order to obtain the phone's ground truth orientation and position, we rigidly mount it to a Kaarta Stencil mapping system. This system uses a LiDAR sensor, a video camera, and Xsens IMU to estimate its current pose at 200Hz. From testing in a Vicon motion capture studio, we measured $<$1.5 degree RMS orientation error and $<$10 cm RMS position error. Given the position error is smaller than the size of most smartphones, the Stencil is of sufficient accuracy to serve as the ground truth while having the advantage of not being constrained to a single room. Based on this testing, we also apply low pass filtering to the trajectory so that it agrees better with the Vicon output. Smartphone data is collected using an iPhone 8, from which we obtain raw accelerometer, gyroscope, and magnetometer readings at 100Hz. Most

Figure 8: Data collection rig: Seen here is the data collection system, consisting of a LiDAR (upper left), camera (middle), iPhone 8 (right), and battery (bottom). The camera, LiDAR, and computer are part of a commercial Kaarta Stencil SLAM system that we use for ground truth positioning; the IMU signals are recorded using the iOS CoreMotion API.

trajectories in this dataset are about 10 minutes in length.

This physical setup is not necessarily reflective of the standard human walking motion while carrying a phone: since the user must carry the mapping system to which the phone is mounted, the user's motions are not necessarily natural. However, we argue that the major factors which make the position of a human walking difficult to regress are maintained. There in no zero-velocity point in the data that allows for ZUPT sensor noise corrections. Additionally, there is no movement constraint on any axis of the device. Furthermore, the results of RoNIN and IONet have already demonstrated the feasibility of deep networks to generalize across different ways of holding a phone and to different brands of smartphones. Because testing such modalities makes it much more difficult to acquire ground truth device orientation (such as in-the-pocket), we primarily rely on the induced motion from a

human shaking the data collection rig over the course of normal walking to generate realistic device motions.

The data we collected consists of trajectories recorded in several different office environments selected to be of varying shapes and sizes. 15 users of different physical builds were instructed to walk with variable speeds, pauses, and arbitrary directional changes while carrying the mapping rig and smartphone. An initial magnetometer calibration [17] was performed by twirling the iPhone at the start location in order to remove any large offsets. This acts as a coarse calibration that is simply scaling the magnetometer values to within a range that is reasonable for the network to learn (similar to how one might perform BatchNorm or prescale the input prior to a network). Overall, approximately 20 hours of data were collected in three different buildings. We split our dataset into several splits across buildings, with each having a set of known and unknown subjects depending on whether they were seen in the training process.

While RoNIN and IONet provide two of the largest datasets for pedestrian inertial odometry, these pre-existing sets lack the necessary data channels for our model. IONet's dataset, OxIOD [19], lacks raw IMU values without the iOS CoreMotion processing and coordinate transform already applied, so our orientation network is unable to improve the results of a model on that dataset. Furthermore, their ground truth orientation measurements display consistent artifacts at certain orientations, which would corrupt the output of any supervised model trained on them. RoNIN does not provide ground truth phone orientations, instead opting to provide the orientation of a second Google Tango phone attached to the user's body. This means we cannot train our orientation network on their trajectories.

5 Experiments

In order to demonstrate the effectiveness and utility of our inertial odometry system, we set three main goals for evaluation: (i) verify that our model produces better orientation estimates than the iOS CoreMotion API used in in other deep-learning methods and state-of-the-art baselines like MUSE, (ii) show that our model is able to achieve higher position localization accuracy than previous methods, and (iii) demonstrate that orientation error is a major source of final position error by showing that other inertial odometry methods benefit from our improved orientation

estimate.

The main metric used to evaluate the orientation network is root mean squared (RMSE) orientation error, measured as the direct angular distance between the estimated and ground truth orientations. We choose RMS error because this places greater emphasis on outliers vs a simple mean angular distance error.

We primarily evaluate the accuracy of our position network using 3 metrics defined by [20] and used for RoNIN [2]:

- **Absolute Trajectory Error (ATE)**: defined as the RMSE between corresponding points in the estimated and ground truth trajectories. The error is defined by the following equation

$$E_i = \boldsymbol{x}_i - \hat{\boldsymbol{x}}_i \tag{13}$$

  where i corresponds to the timestep. This is a measure of global consistency and will tend to increase as a function of trajectory length.

- **Time-Normalized Relative Traj. Error (T-RTE)**: defined as the RMSE between the displacements over all corresponding 1-minute-long windows in the estimated and ground truth trajectories. The error is defined by the following equation

$$E_i = (\boldsymbol{x}_{i+t} - \boldsymbol{x}_i) - (\hat{\boldsymbol{x}}_{i+t} - \hat{\boldsymbol{x}}_i) \tag{14}$$

  where i corresponds to the timestep and t is the interval length (6000 in this case due to 100Hz sampling rate). This provides a local estimate of the consistency between trajectories.

- **Distance-Normalized Relative Traj. Error (D-RTE)**: defined as the RMSE between the displacements over all corresponding windows in the estimated and ground truth trajectories where the ground truth trajectory has traveled 1 meter. The error is defined by the following equation

$$E_i = (\boldsymbol{x}_{i+t_d} - \boldsymbol{x}_i) - (\hat{\boldsymbol{x}}_{i+t_d} - \hat{\boldsymbol{x}}_i) \tag{15}$$

  where i corresponds to the timestep and t_d is the interval length required to traverse a distance of 1 meter.

The RMSE for these metrics is calculated using the following equation:

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \|E_i\|_2^2}, \tag{16}$$

where E_i is the i-th error term out of m total.

5.1 Training

We implemented our model in Pytorch [21] and train it using the Adam optimizer [22] on an Nvidia RTX 2080Ti GPU. The orientation network is first individually trained with a learning rate of 0.0001. Then, using these initialized weights, the position network is attached and the entire model trained using a learning rate of 0.001. Initially, we fix the pre-trained weights of the orientation network while the position network converges; then we fine-tune the entire model end-to-end. We use a batch size of 64 and usually the network reaches convergence within 15 epochs. Each epoch involves a full pass through all training data.

5.2 Testing

At test time, an initial orientation can be provided or, assuming the magnetometer has been properly calibrated, the initial orientation can be estimated by the network directly. When generalizing to other locations with reference frames oriented differently from that of the training locations, an angular correction must be provided (e.g., x pointing north vs y pointing north).

Since this system is meant to aid pedestrian navigation using a smartphone's internal IMU, we benchmark the speed of the full model. Using an AMD Ryzen Threadripper 1920x 12-Core CPU, the forward inference time is approximately 0.265 seconds to process 2 seconds of data (200 samples). While we did not benchmark the speed of our pipeline on a phone, these results suggest it is capable of easily running in real time on modern smartphone processors.

5.3 Baselines

In order to show the performance of our orientation estimation, we compare it against the iOS CoreMotion API and MUSE complementary filter [11] as baselines.

The iOS CoreMotion estimate is selected because of its ubiquity. The MUSE complementary orientation filter is selected because of its high performance on longer trajectories.

In order to show the performance of our inertial odometry pipeline, we compare it against several different baseline inertial odometry methods.

Pedestrian Dead Reckoning is chosen as the representative of traditional odometry methods since it makes the fewest assumptions about how the smartphone is held. We use a similar PDR baseline to [2] that involves regressing a heading and distance every step. We assume a user stride length of $0.67m/step$. The heading is acquired using the iOS CoreMotion orientation estimates.

The two main data-driven inertial localization methods explored in prior work are IONet and RoNIN. IONet takes coordinate-frame-transformed IMU channels as input to a BiLSTM and regresses polar displacements over 2 second intervals. We use our own implementation as the original code is not publicly available. IONet was primarily evaluated in a small Vicon motion capture room. We have found, however, that IONet does not appear to perform very well in large indoor environments, even when trained on such trajectories. As such, we correct the IMU channels with the ground truth orientations to give it the best chance at success.

RoNIN takes device orientation estimates directly from the phone's internal API. It uses these to transform IMU channels and directly regress current positions at each timestep relative to the starting location. We use their *exact* open source implementation for this baseline. In the course of our evaluations using their code, we noticed a bug in their evaluation metric, where they accidentally forgot the L2-norm in their calculation of RMS distance error when deriving ATE and RTE (see Equation 16 above). Because of this error, their numbers for their performance consistently under-report the true error. As this is consistent throughout their paper, however, the relative comparisons between their models and the conclusions they reach are still valid. We use the correct method for computing these metrics, which helps explain the discrepancies between the relative size of the errors we obtain compared to their paper (in addition to the fact that the trajectories are from different buildings and of different length).

Model	RMSE (rad)	RMSE (°)
Orient Regression Only	0.22	12.72
Orient Reg. + EKF	0.16	9.40
Orient Reg. + Mag. Correct	0.16	9.34
Full Orientation Network	**0.13**	**7.21**

Table 1: Ablative analysis of the Orientation Network

5.4 Orientation Analysis

In order to show the importance of each stage of the orientation estimation pipeline described previously, we perform an ablative analysis across the components of the system using our building-scale dataset. Table 1 shows the ablative analysis of our orientation network on a subset of our data using the same test subjects as seen in the training set, as well as the same building. With everything else constant, both the EKF and magnetic correction network reduce RMS angle error by 0.058 radians (3.32 degrees) over the base orientation regression network. Taken together, the EKF and magnetic field correction lead to a 0.096 radian (5.50 degree) drop in RMS error compared to the base network.

We now seek to answer the question of whether our orientation pipeline is worth using, i.e., does it outperform the systems that others use? We perform evaluations on both the Mocap dataset and the building-scale dataset. Figure 9 shows the orientation estimate results on the Mocap dataset when comparing our network output and the result of the Xsens IMU's built-in estimate on the test set. The RMS error of our network, averaged over the entire duration of these trials, was about 42.2% less than that of the Xsens estimate. Figure 10 visualizes the ground truth, the Xsens estimate, and our estimate in Euler angle space, where it can be seen that our method produces far more consistent estimates over the 2 minute duration of that trial. The yaw angle of our orientation estimate is broadly representative of the heading of the device and thus is the orientation axis that sees the most change over the course of a trajectory.

Next, we evaluate on the much larger building-scale dataset. Because the length of these trajectories is much longer as well (about 10 minutes), the drift in the conventional methods is more discernable over time. Table 2 demonstrates that our model outperforms the iOS and MUSE estimates by a considerable margin when

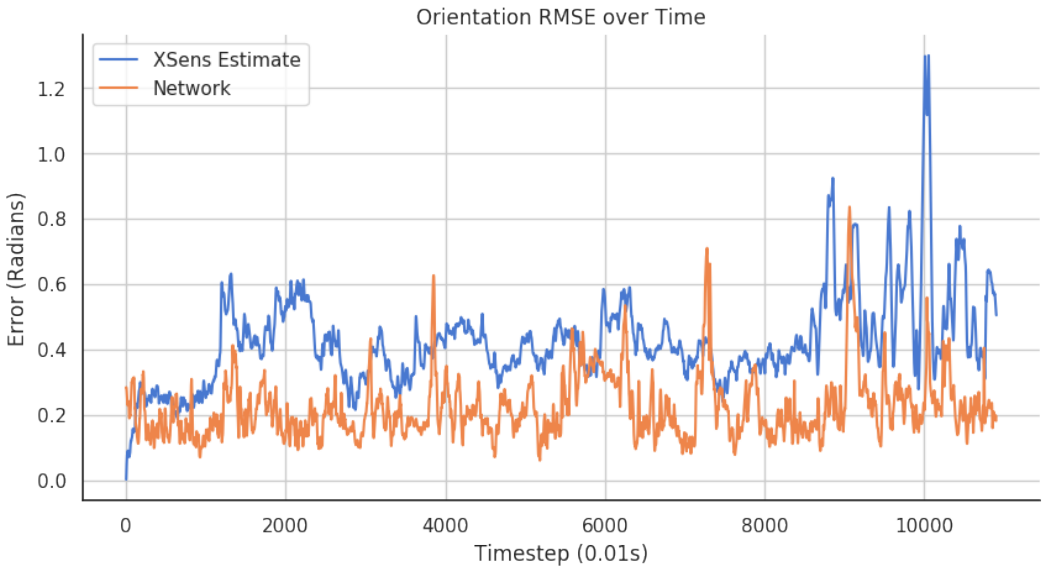


Figure 9: Orientation RMSE for Mocap dataset averaged over test set (6 trajectories, first 100s shown). The Xsens estimate's RMSE was 0.45 radians, whereas the network estimate's RMSE was 0.26 radians.

trained separately on trajectories from each building. Averaging across all three buildings by length of trajectory, our estimate is 0.169 radians (9.68 degrees) more accurate than the iPhone estimate and 0.135 radians (7.73 degrees) more accurate than the MUSE estimate.

In order to see why our model outperforms the iOS estimate, we compare the Euler angle representation of our estimate and the iPhone estimate over the course of a trajectory in Figure 11. We can see from these plots that the iOS estimate exhibits large drift in yaw over time. It can also be seen that our method most

System	**Bldg 1 (rad)**	**Bldg 2 (rad)**	**Bldg 3 (rad)**	**All (rad)**
iOS CoreMotion	0.39	0.41	0.33	0.37
MUSE [11]	0.31	0.53	0.28	0.34
Our Orient	**0.13**	**0.21**	**0.24**	**0.20**

Table 2: Orientation angular RMSE comparison. Each building is trained separately and tested on unseen subjects over 10 minute trajectories.

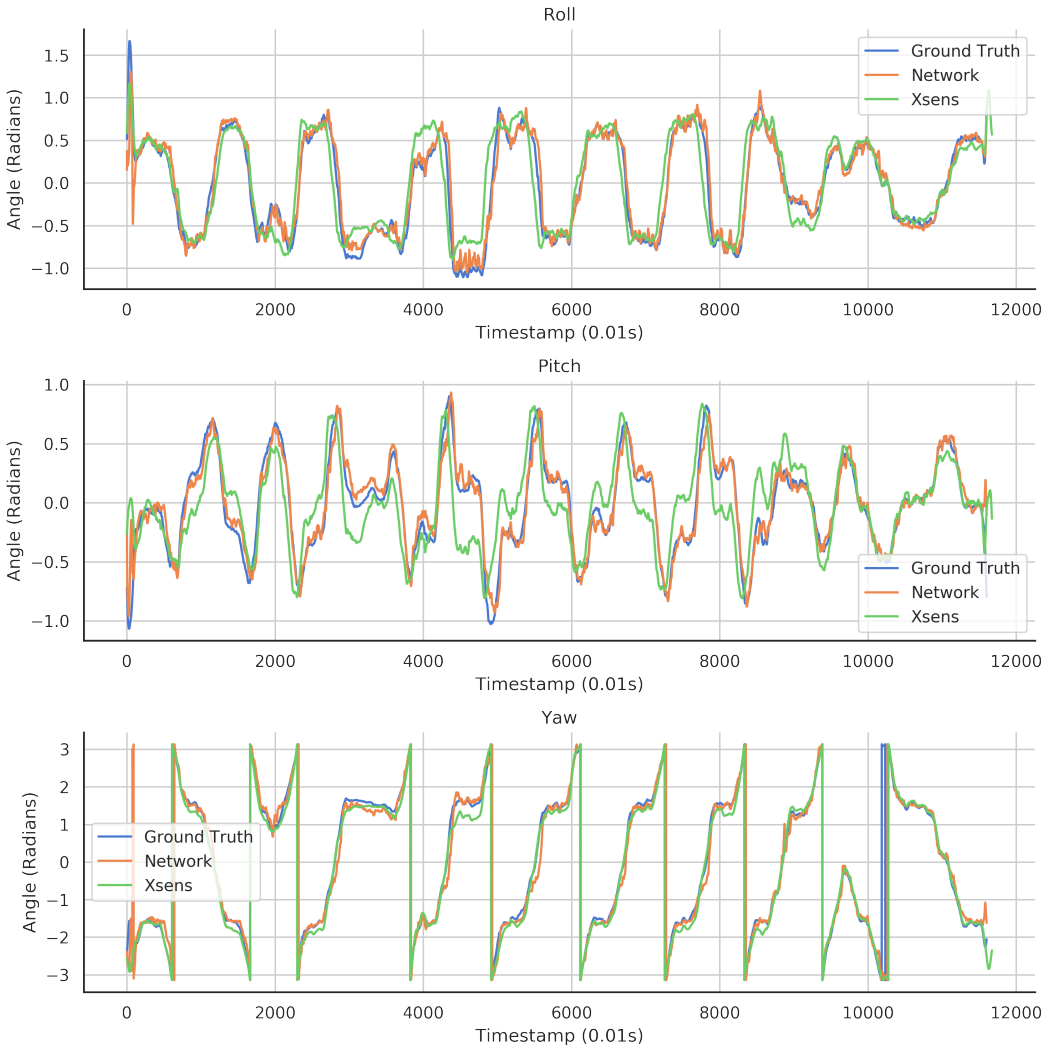


Figure 10: Orientation comparison for Mocap dataset on a single trajectory

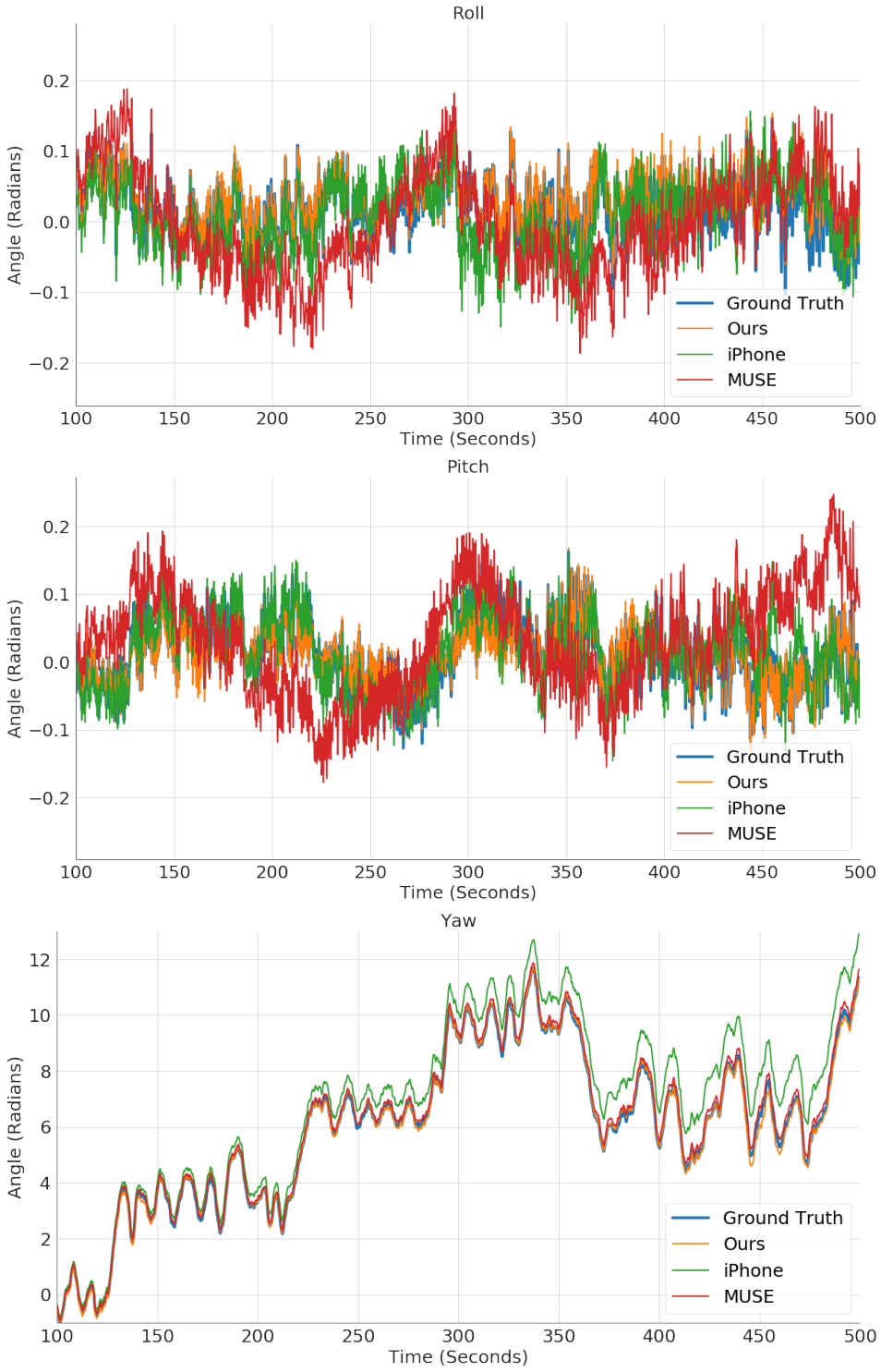


Figure 11: Predicted orientation over time in building dataset represented as Euler angles

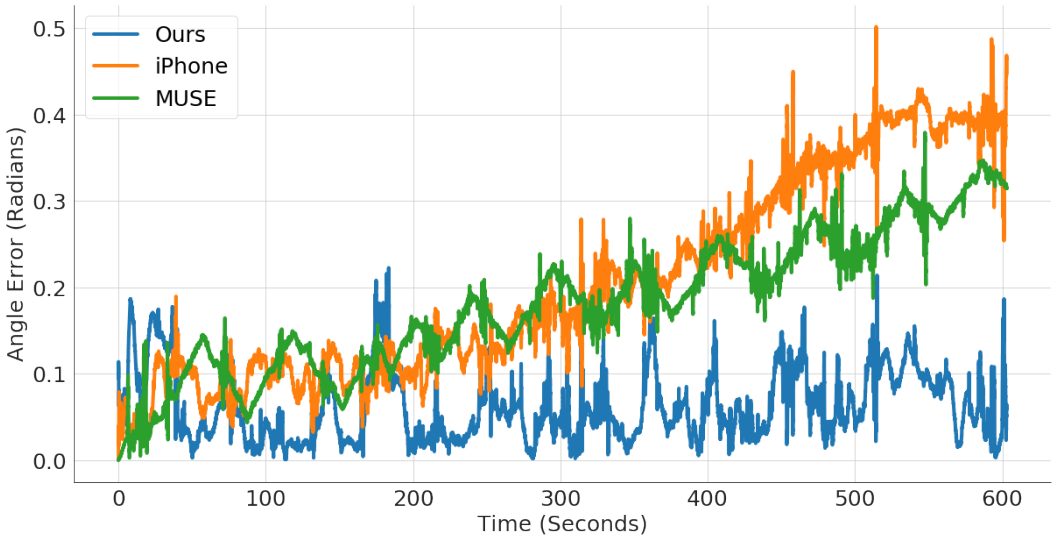


Figure 12: Angular error between ground truth orientation and predicted orientation over time.

consistently matches the ground truth across all axes. The error growth over time is perhaps most evident in Figure 12, where both MUSE and the iPhone exhibit a steeper error growth than our method. While the iPhone estimate is generally able to recover from such drastic error growth eventually, our pipeline quickly and frequently corrects itself to keep the orientation estimate highly accurate in the face of constant device motion.

5.5 Position Analysis

In addition to verifying the effectiveness of our orientation estimation pipeline, we also conduct several experiments to verify both the importance of our orientation network to positioning accuracy as well as the overall accuracy of our system.

Tables 3 and 4 show the comparison between between our end-to-end model and a mix of traditional and deep-learning baselines. Our model performs more successfully on subjects it has seen during training (Known Subjects), but also generalizes well to other users (Unknown Subjects) and outperforms other methods. We perform particularly well in the T-RTE metric, where our network exhibits over 4 meters less drift over the same amount of time than other models. Across different building shapes, our position estimate also outperforms other methods. The best

Model	Building 1, Known Subjects			Building 1, Unknown Subjects		
	ATE	**T-RTE**	**D-RTE**	**ATE**	**T-RTE**	**D-RTE**
PDR	26.98	16.49	2.26	24.29	12.65	2.77
IONet [1]	33.42	22.97	2.47	31.28	24.04	2.29
RoNIN-LSTM [2]	18.62	7.02	0.53	18.17	6.51	0.51
RoNIN-TCN [2]	12.00	6.41	0.48	13.41	5.82	0.48
RoNIN-ResNet [2]	9.03	6.43	0.56	12.07	5.95	0.49
OURS	**4.63**	**2.05**	**0.27**	**9.39**	**3.04**	**0.33**

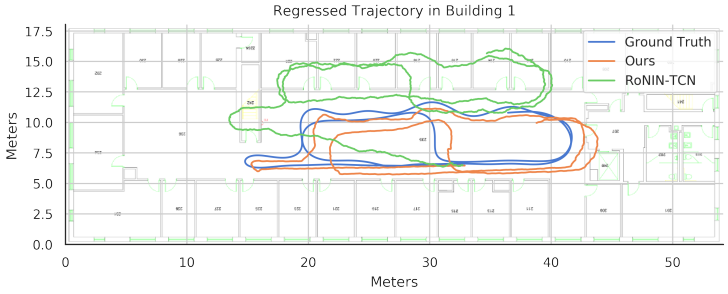
Table 3: End-to-end model comparison across subjects

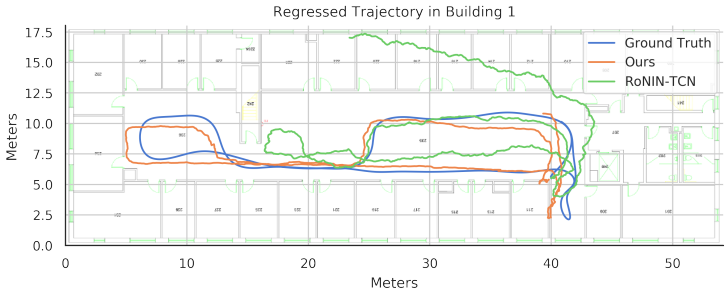
performance is achieved in the smaller buildings, with all models struggling on the much larger Building 2, which also has much more extreme magnetic field variations.

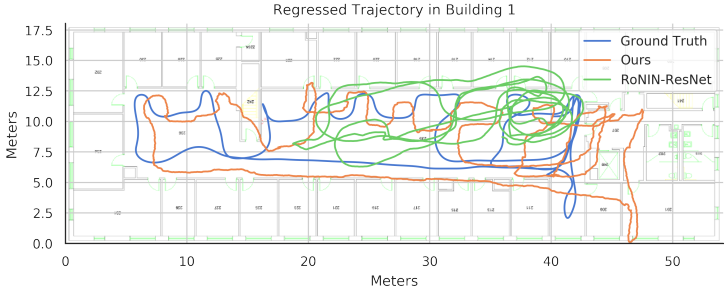
Figure 13 shows example outputs of our network in the various buildings, with the output of the RoNIN-ResNet model (next best performing) also shown to provide a point of comparison. As noted previously, the other models we evaluate against rely on the iPhone orientation estimate (with the exception of IONet, which still performs poorly despite using ground truth orientations), which partially explains the tilt partway through several trajectories, where the iPhone drift becomes increasingly evident. As mentioned previously, we noticed that IONet, in particular, does not work very well in these building-scale environments, which was a similar phenomenon reported by RoNIN's evaluations of IONet.

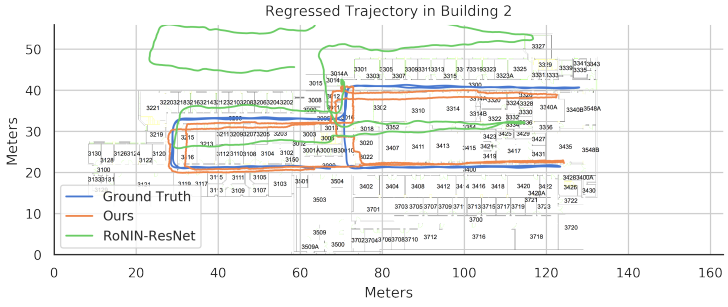
One point to note is the consistent performance of PDR in the ATE metric. It is capable of achieving (relatively) low errors for this metric because of the fewer updates which take place as a result of step counting, so the overall trajectory tends to stay in the same general region. Some of the other models tend to drift slowly over time until the trajectory is no longer centered in the same original location despite almost always producing more accurate trajectory shapes, as reflected by the lower RTE metrics.

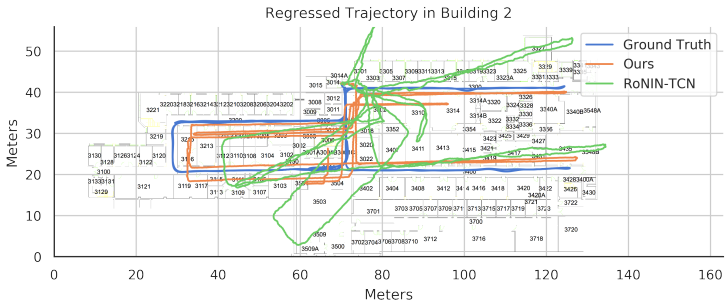
In order to analyze the effects of the accuracy of the orientation estimate on the final position estimate, we provide the iOS CoreMotion estimate, our orientation model's estimate, and the ground truth orientation to each model and measure their performance. Table 5 shows the analysis of the effects of using the various qualities of orientation estimates. The results indicate that our higher-accuracy orientation


Regressed Trajectory in Building 1
Ground Truth
Ours
RoNIN-TCN
Meters
Meters


Regressed Trajectory in Building 1
Ground Truth
Ours
RoNIN-TCN
Meters
Meters


Regressed Trajectory in Building 1
Ground Truth
Ours
RoNIN-ResNet
Meters
Meters


Regressed Trajectory in Building 2
Ground Truth
Ours
RoNIN-ResNet
Meters
Meters


Regressed Trajectory in Building 2
Ground Truth
Ours
RoNIN-TCN
Meters
Meters

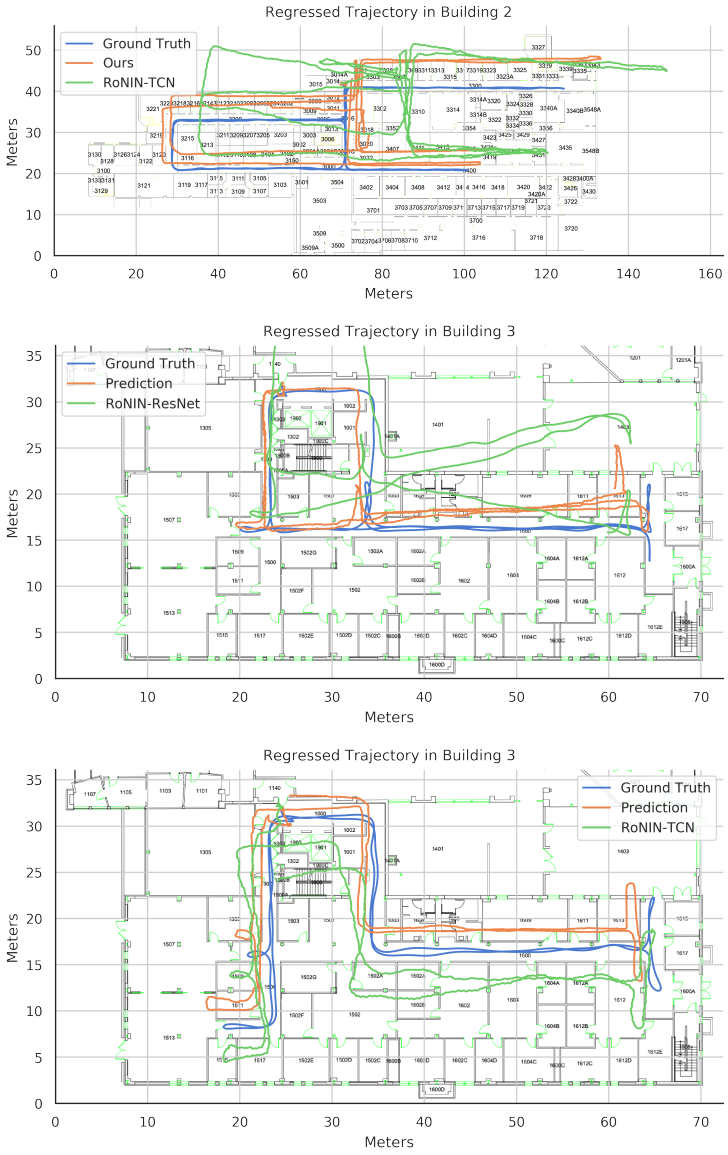


Figure 13: Predicted vs ground truth trajectory of our model vs the best baseline model. Trajectories are initialized to the same pose and truncated in length (<5min) to reduce clutter in the visualization.

Model	Building 1			Building 2			Building 3		
	ATE	**T-RTE**	**D-RTE**	**ATE**	**T-RTE**	**D-RTE**	**ATE**	**T-RTE**	**D-RTE**
PDR	25.70	14.66	2.50	21.86	19.48	1.66	**12.66**	12.74	1.09
IONet	32.40	23.48	2.38	105.25	62.14	3.01	81.97	36.62	2.65
RoNIN-LSTM	18.41	6.78	0.52	29.81	18.67	0.75	33.69	13.14	0.62
RoNIN-TCN	12.67	6.13	0.48	22.52	13.69	0.73	24.79	12.48	0.59
RoNIN-ResNet	10.48	6.20	0.53	35.44	15.71	0.49	14.11	11.78	0.60
OURS	**6.90**	**2.52**	**0.30**	**9.35**	**5.68**	**0.40**	14.23	**4.11**	**0.33**

Table 4: Comparison across buildings using separately-trained models

Model	iOS Orientation			Our Orientation			Ground Truth		
	ATE	**T-RTE**	**D-RTE**	**ATE**	**T-RTE**	**D-RTE**	**ATE**	**T-RTE**	**D-RTE**
RoNIN-LSTM	18.41	6.78	0.52	10.97	3.27	0.36	10.35	2.62	0.26
RoNIN-TCN	12.67	6.13	0.48	13.18	3.54	0.32	5.56	1.93	0.22
RoNIN-ResNet	10.48	6.20	0.53	9.05	2.96	0.40	5.47	2.28	0.39
OURS	15.14	5.61	0.45	6.90	2.52	0.30	6.55	2.25	0.27

Table 5: Performance of Position Networks using Different Orientation Estimates on Building 1

estimate leads to improvements in all models, supporting the idea that orientation estimate error is a major component of positioning error after propagating through any inertial odometry system.

Figure 14 shows the ability of our model to generalize to different methods of holding the sensing device. A well-trained orientation network would theoretically pass the same input to the position network regardless of how the phone is held or rotated, assuming the same trajectory. This is because the movements, when rotated to the world frame, would appear identical. As such, we validate the robustness of our orientation network by separately training on a specific grip and testing on another grip method. A back-and-forth style of trajectory was chosen for visualization to show any drift clearly and a 90 degree rotation was chosen to represent a dramatic difference in grip. The results indicate that the model accurately regresses position if it is gripped completely differently from the way it was held in the trajectories in the training set. Both sets of trajectories have an ATE of approximately 2 meters, a T-RTE of approximately 2 meters per minute, and a D-RTE

of approximately 0.25 meters per meter. This figure indicates that the orientation network correctly handles the rotation of IMU measurements into the world frame, at which point the accuracy of the position network becomes independent of how the IMU is held.

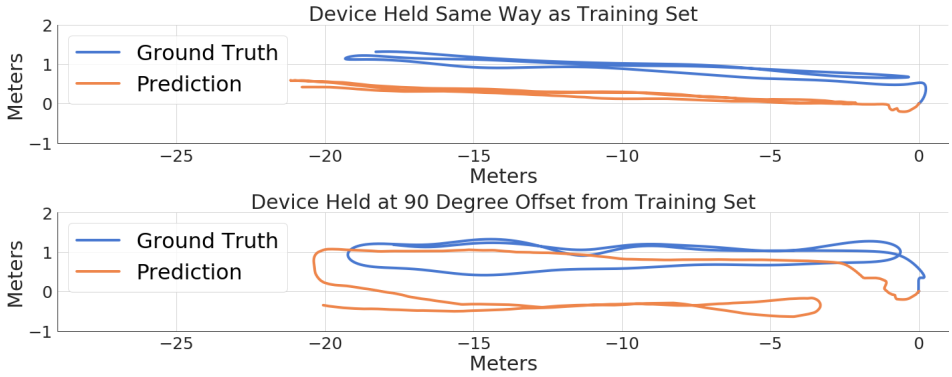


Figure 14: Performance of Position Network Under Different Grips of the Data Collection Rig

5.6 Limitations

While our method and learning-based approaches in general demonstrate superior performance to traditional approaches, they also possess several limitations. The main drawback of learning-based methods is that they require ground truth data collection, which is an expensive and tedious procedure due to requiring human subjects. Another drawback of learning-based approaches is performance degradation when generalizing to out-of-distribution examples. While cross-subject evaluation demonstrated good generalization, none of the learning-based methods had good generalization capabilities in cross-building evaluation. Achieving good results required training and testing on the same building. Because the dataset collected only consisted of 3 buildings, we suspect there is insufficient variation to properly encompass the set of possible trajectory shapes. Greater variety is needed for further evaluation. While learning-based methods convincingly outperform traditional approaches in known environments, this gap in performance further increases the cost of deployment, as new environments would require collecting new data for optimal performance.

Specifically for our approach, this generalization gap manifests itself in both orientation and position networks. The orientation network requires some form of fine-tuning or re-training in new environments with different magnetic field variations. We notice occasionally dramatic differences in magnetic field distortion between the various buildings in our dataset, even after performing an initial calibration for each environment. This is because the fields are not only different between buildings, but also vary within each building. This challenge plagues even traditional approaches, but because our model explicitly learns corrections to the magnetic field, out-of-distribution examples do not get corrected appropriately. Meanwhile, the position network requires variety in building shapes. We notice poor generalization with all learning-based methods when generalizing from rectangular trajectories to more elliptical shaped ones, e.g., training on buildings 2 and testing on building 1.

6 Conclusion

In this work, we present the first data-driven inertial odometry method to estimate high accuracy 3D device orientations in order to improve position predictions. In addition, we build the first large dataset which includes the 3D orientation of the device (not the user), spanning 20 hours of pedestrian motion across 3 buildings and 15 users. Existing traditional inertial odometry methods either use assumptions or constraints on the user's motion, while previous data-driven techniques use smartphone API orientation estimates. We demonstrate the ability to regress device orientation more accurately than traditional techniques, in particular the iOS Core Motion API orientation estimator, with no constraints on device placement. Our orientation estimator uses an online magnetic calibration network, an orientation estimation network, and a differentiable Extended Kalman Filter to form a stable and accurate 3D device orientation estimate. We achieve a 0.169 radian (10 degree) decrease in root mean squared orientation error over the iOS Core Motion API. Our improved orientation estimates and position network, when combined, lead to higher accuracy than all previous methods across multiple buildings and users. In addition, our orientation estimates improve the accuracy of prior traditional and data-driven inertial odometry techniques, showing the utility of our inertial orientation pipeline.

While we demonstrate performance improvements over the current state-of-the-art deep learning methods for inertial odometry, there remain several opportunities

for future work. The first is the issue of generalization to diverse environments. While the dataset we collected was across 3 different buildings, greater diversity in shape is necessary for robust and accurate generalization of the model to unknown buildings. None of the prior works address this issue either, as all of these methods suffer from declines in performance when introduced to a new environment. One potential avenue for solving this problem is to add on a map-matching component after the inertial estimate, whereby constraints (like wall locations) from the environment are encoded and used to refine the localization. Another extension is with regards to improving the effectiveness of the EKF estimate. Instead of relying on static covariance matrices, the network could produce an estimate of aleatoric uncertainty (e.g., by predicting the mean and variance of a Gaussian distribution) that would allow for dynamic covariance updates when the network is uncertain. Finally, additional data could be collected to investigate exactly what aspects of human dynamics the position network is learning, e.g., is it learning to detect steps or some constant velocity model? This could involve collecting additional data of sideways or backwards walking, constantly switching phone grip, or training on a human but testing on a rolling dolly moving at similar speeds. Because we rely on deep learning models, it is ultimately difficult to interpret exactly what the network is learning, and thus what unexpected failure modes might appear.

References

[1] C. Chen, X. Lu, A. Markham, and N. Trigoni, "IONet: Learning to cure the curse of drift in inertial odometry," *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[2] H. Yan, S. Herath, and Y. Furukawa, "RoNIN: Robust neural inertial navigation in the wild: Benchmark, evaluations, and new methods," *CoRR*, vol. abs/1905.12853, 2019.

[3] D. Ahmetovic, C. Gleason, C. Ruan, K. Kitani, H. Takagi, and C. Asakawa, "Navcog: A navigational cognitive assistant for the blind," *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, p. 90–99, 2016.

[4] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time," *Proceedings of Robotics: Science and Systems Conference*, July 2014.

[5] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara, "A comparison of pedestrian dead-reckoning algorithms using a low-cost MEMS IMU," *2009 IEEE International Symposium on Intelligent Signal Processing*, pp. 37–42, Aug 2009.

[6] F. Caron, E. Duflos, D. Pomorski, and P. Vanheeghe, "GPS/IMU data fusion using multisensor Kalman filtering: Introduction of contextual aspects," *Inf. Fusion*, vol. 7, p. 221–230, June 2006.

[7] T. Qin, P. Li, and S. Shen, "VINS-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[8] C. Chen, C. X. Lu, J. Wahlström, A. Markham, and N. Trigoni, "Deep neural network based inertial odometry using low-cost inertial measurement units," *IEEE Transactions on Mobile Computing*, 2019.

[9] W. Liu, D. Caruso, E. Ilg, J. Dong, A. Mourikis, K. Daniilidis, V. Kumar, J. Engel, A. Valada, and T. Asfour, "TLIO: Tight learned inertial odometry," *IEEE Robotics and Automation Letters*, p. 1–1, 2020.

[10] S. Madgwick, A. Harrison, and R. Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm," *IEEE International Conference on Rehabilitation Robotics*, vol. 2011, pp. 1–7, 06 2011.

[11] S. Shen, M. Gowda, and R. Roy Choudhury, "Closing the gaps in inertial motion tracking," *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, p. 429–444, 2018.

[12] I. Bar-Itzhack and Y. Oshman, "Attitude determination from vector observations: Quaternion estimation," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. Vol. AES-21, pp. 128 – 136, 1985.

[13] J. Marins, X. Yun, E. Bachmann, R. Mcghee, and M. Zyda, "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors," *IEEE International Conference on Intelligent Robots and Systems*, vol. 4, 01 2002.

[14] A. M. Sabatini, "Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing.," *IEEE Trans. Biomed. Engineering*, vol. 53, no. 7, pp. 1346–1356, 2006.

[15] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," *CoRR*, vol. abs/1812.07035, 2018.

[16] D. Huynh, "Metrics for 3D rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, pp. 155–164, 10 2009.

[17] M. Kok and T. B. Schön, "Magnetometer calibration using inertial sensors," *CoRR*, vol. abs/1601.05257, 2016.

[18] H. Xie, T. Gu, X. Tao, H. Ye, and J. Lv, "MaLoc: A practical magnetic fingerprinting approach to indoor localization using smartphones," *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, p. 243–253, 2014.

[19] C. Chen, P. Zhao, C. X. Lu, W. Wang, A. Markham, and N. Trigoni, "OxIOD: The dataset for deep inertial odometry," *CoRR*, vol. abs/1809.07491, 2018.

[20] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, and R. Siegwart, "Towards a benchmark for RGB-D SLAM evaluation," *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS)*, June 2011.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, 2019.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *3rd International Conference for Learning Representations, San Diego*, 2015.