Tampere University

Juha Ylikoski

DEAD RECKONING WITH IMU SENSORS

State-of-the-art

Bachelor's thesis
Faculty of Information Technology and Communication Sciences
Examiner: Prof. Joni Kämäräinen
May 2022

ABSTRACT

Juha Ylikoski: Dead reckoning with imu sensors
Bachelor's thesis
Tampere University
Information technology
May 2022

Inertial measurement unit (IMU) odometry's main purpose is to estimate position from inertia measuring sensors like accelerometers and gyroscopes. With the decreasing cost of MEMS-based IMU sensors and the increased amount of devices containing them, IMU-odometry is in process of becoming a viable alternative to more traditional methods on low-cost and low-power setups.

The problematic nature of IMU odometry's double integral from acceleration to position has made it difficult and inaccurate to apply this form of position estimation to modern smartphones. This has been mostly caused by their noisy sensors and the use of only classical algorithms. This thesis compares state-of-the-art methods of dead reckoning which use machine learning to achieve better positioning accuracy.

The main purpose of IMU odometry is to estimate the position of the actor using acceleration sensors and gyroscopes. These two sensor inputs can be augmented with for example barometers or magnetometers but the main focus is to accurately estimate the position of an actor with low-power consumption sensors. This excludes camera and GPS-based approaches due to their high-power consumption.

In previous work related to this field, this problem has been tried to solve with physics-based approaches calculating the double integral in algorithms like extended Kalman filter, but due to the double integral accumulating errors, they produce poor results. The methods compared in this thesis are both learning-based and try to estimate the first integral from acceleration to velocity using convolutional neural networks. By replacing the first integral with CNN, we can mitigate the noise from the acceleration data and get a reasonable velocity approximation which we can calculate into position.

Keywords: IMU, inertial measurement unit, inertial-odometry, convolutional neural network, pedestrian dead reckoning

TIIVISTELMÄ

Juha Ylikoski: Reitin seuranta IMU-sensoreilla
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2022

Inertian mittaussensori (IMU)-odometrian päätarkoitus on arvioida kohteen sijainti sensoreilla, kuten kiihtyvyyssensoreilla ja gyroskooppeilla. MEMS-pohjaisten IMU-sensoreiden laskeneiden kustannusten sekä niiden yleistymisen myötä IMU-odometriasta on tulossa toimiva vaihtoehto perinteisille menetelmille matalakustannuksisissa tai matalatehoisissa järjestelmissä.

IMU-odometrian tuplaintegraalin ongelmallisuus kiihtyvyydestä sijaintiin on tehnyt siitä vaikean ja epätarkan menetelmän moderneille älypuhelimille klassisilla menetelmillä. Tämä työ esittelee ja vertailee koneoppimista käyttäviä menetelmiä sijainnin seurantaa.

IMU-odometrian päätarkoitus on laskea kohteen sijainti käyttäen suurimmaksi osaksi kiihtyvyyssensoreita sekä gyroskooppeja. Näitä kahta sensoria voidaan augmentoida barometreillä, magnetometreillä sekä muilla sensoreilla, mutta tämän menetelmän tarkoitus on laskea kohteen sijainti matalatehoisilla menetelmillä, mikä poissulkee mahdollisuuden kamera tai GPS pohjaisiin menetelmiin.

Aiemmin tätä ongelmaa on yritetty ratkaista fysiikkapohjaisella lähestymistavalla laskien tuplaintegraalin algoritmeilla kuten extended Kalman filter. Suurimmaksi osaksi nämä menetelmät ovat tuottaneet huonoja tuloksia tuplaintegraalin nopean virheen kumuloinnin vuoksi. Tässä työssä vertaillut menetelmät ovat molemmat oppimispohjaisia ja yrittävät laskea ensimmäisen integraalin kiihtyvyydestä nopeuteen konvoluutionallisilla neuroverkoilla. Vaihtamalla ensimmäisen integraalin neuroverkkoon voidaan vähentää kohinaa sekä saada tyydyttävän nopeusarvion, jota voimme käyttää sijainnin laskemiseen.

Avainsanat: IMU, inertianmittaussensori, inertiaalinen odometria, konvoluutinallinen neuroverkko, jalankulkijan reitin seuranta

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

T_k^a	Accelerometer diagonal bias
ω_k^{biased}	Raw gyroscope angular velocity value at the time of k
a_k^{biased}	Raw acceleration acceleration value at the time of k
b_k^a	Accelerometer linear bias
b_k^w	Gyroscope linear bias
e_k	Error at the time of k
g	Acceleration caused by gravity
p_k	Position at the time of k
q_k	Quaternion rotation vector at time of k
t_k	Time at the moment of k
v_k	Velocity at the time of k
API	Application programming interface
CNN	Convolutional neural network
EKF	Extended Kalman Filter
GPS	Global positioning system
GPU	Graphics processing unit
IMU	Inertial measurement unit
MSE	Mean squared error
NN	Neural network
ResNet	Residual neural network
SPEED	Non-directional velocity
VELOCITY	Directional velocity

1. INTRODUCTION

Localization or localizing position and rotation has become an important part of many devices and applications like virtual reality headsets and smartphone applications. Accurate localization usually requires either high power consumption sensors like GPS or some specialized hardware like infrared transmitters and receivers which especially in mobile devices usually are not viable.

An alternative way to localize the subject is to offload part of the tracking to odometry which tries to keep track of the position of the device over time. There are many viable methods to do this, such as visual-inertial odometry [1] but due to high power consumption or additional constraints like the requirement for a camera with an unobstructed field of view, these methods may not work for all use cases. One means of odometry with low-power consumption is inertial odometry which uses IMU sensors to track the device's position from acceleration and angular velocity. Previously this method has not been viable on low-end hardware due to high noise and phenomena of gravity "bleeding out" to the measurement. This is caused by not subtracting gravity-caused acceleration correctly which then gets accumulated fast with the double integral from acceleration to the position.

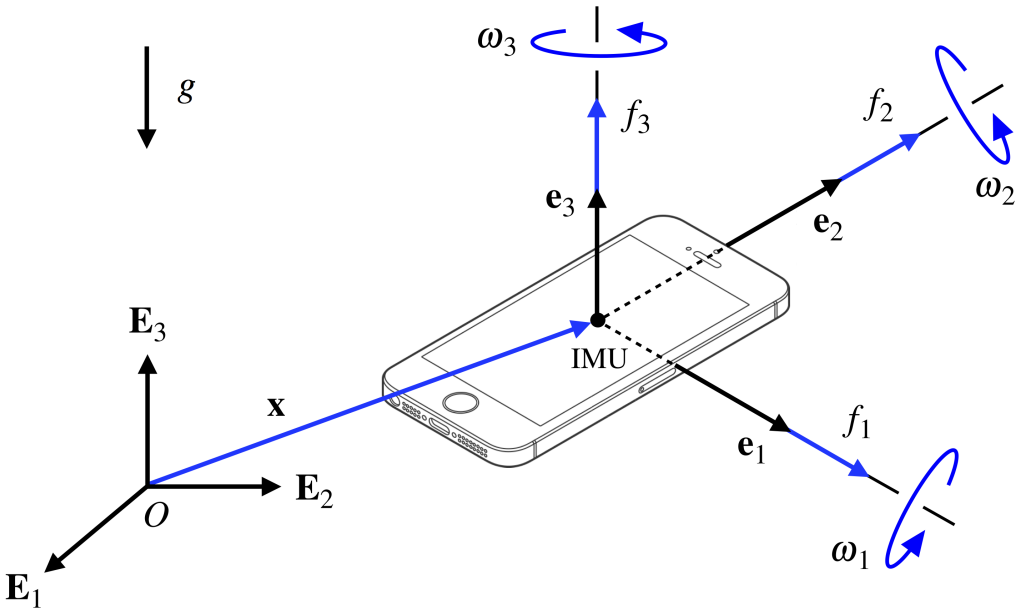

Figure 1.1. *IMU-sensor of an iPhone SE, it's coordinate frame and it's measured axis [2]*

This thesis tries to find good ways to track device's location using inertial odometry with low-cost sensors like the ones in most smartphones. Traditionally, this has been done with algorithms like extended Kalman filter (EKF) which calculated the double integral into a reasonable estimation of the location and the covariance. This does not work well with high noise smartphone IMU sensors. These large errors have been previously decreased using methods like zero velocity updates [3], but these methods do not generalize well, and in the case of e.g. pedestrian tracking requires the sensors to be mounted to their legs [4] or require complicated math to detect them from a pocket [3]. Instead, this paper inspects two different methods of using machine learning to estimate the velocity of the device which can then be used with traditional algorithms like EKF to estimate position. The benefits of this method are better prevention of gravity "bleeding" to the measurements and reduction of noise in the velocity.

Chapter 2 explains some related work and which are the current means of calculating odometry. Chapter 3 goes into detail about the mathematics of inertial odometry and the two inspected methods. Chapter 4 contains experiments and results. Chapter 5 discusses some problems with these approaches and chapter 6 summarizes the results of this thesis.

2. RELATED WORK

Traditionally IMUs have been secondary sensors in odometry systems where they have been used in algorithms such as EKF to aid in positioning the device [[5], [6] and [7]]. With EKF-based solutions, the system can have multiple different sensors which measure multiple different variables such as wheel angles, speed, pressure, magnetic field, and use active systems like infrared and light detection and ranging.

More recent approaches to odometry are e.g. inertial-visual-odometry that has been proven successful [[8], [9], [1]] and are applied in products such as Google Tango [10]. Inertial-visual-odometry is used as ground truth for many inertial-odometry datasets [[11], [12], [13]]. These systems have some special requirements such as requiring an unobstructed view for the camera and have high power consumption due to the visual odometry's processing needs but are creating good results.

Inertial-odometry without high-power consumption sensors or methods has gained popularity due to advancements in convolutional neural networks such as ResNet. This has increased attempts to use learning-based approaches to inertial odometry problems. These methods try to use neural networks to aid in estimating position data from the noisy acceleration and angular velocities and in some cases mix in other low power sensors such as barometers or magnetometers.

This paper inspects recent works by Cortés et al. [14] and Herath et al. [12], and tries to replicate their findings and compare them. These two papers were selected due to their different approaches where Cortés et al. [14] uses more traditional extended Kalman filter to calculate the position information and augment it with pseudo speed measurements coming from a neural network. On the other hand, Herath et al. [12] estimates the velocity vector for the device which could then be either just cumulatively summed or used in other algorithms such as EKF to augment the data with other sensors.

3. THEORY

This chapter is divided into two sections which first of them discusses the mathematics of inertial odometry which would apply to classical methods like pure EKF. It goes through how raw acceleration and angular velocities can be used to calculate the change in position and pose. The second section inspects two alternative methods which would partially replace or augment the mathematics to produce better results.

3.1 Physics of inertial odometry

The device containing the IMU sensors usually measures its acceleration and angular velocity in its device coordinate frame. This means that the device's coordinate frame is rotating when compared to the world coordinate frame and adds complexity to calculating the velocity vectors for the device. This can be negated by transforming the device coordinate frame into a heading-agnostic coordinate frame meaning attaching the coordinate frame to some known non-rotating coordinate frame e.g. world coordinate frame with a static rotation and translation. For example, the coordinate frame containing the device's position, velocity and heading can be made heading agnostic by settings its heading (q_k) at the start of the scenario to the device's rotation in comparison to the world coordinate frame and updating it as in equation (3.1)

$$\begin{pmatrix} p_k \\ v_k \\ q_k \end{pmatrix} = \begin{pmatrix} p_{k-1} + v_{k-1}\Delta t_k \\ v_{k-1} + [q_k(a_k + \epsilon_k^a)q_k^* - g]\Delta t_k \\ \Omega[(\omega_k + \epsilon_k^\omega)\Delta t_k]q_{k-1} \end{pmatrix} \tag{3.1}$$

Where p_k, v_k, q_k are device's position, velocity and heading (quaternion) at the time of k, $\Delta t_k = t_k - t_{k-1}$, a is acceleration after removing diagonal (T_k) and linear (b_k) sensor biases (3.2), w is angular velocity after removing sensor linear bias (3.2), g is acceleration caused by gravity, ϵ_k^a is the error, $q[\cdot]q_k^*$ is quaternion rotation and Ω denotes quaternion rotation change matrix [3]

$$\begin{cases} a_k = T_k^a a_k^{biased} - b_k^a \\ \omega_k = \omega_k^{biased} - b_k^w \end{cases} \tag{3.2}$$

To make sure that the heading agnostic coordinate frame is aligned with the world coordinate frame the algorithm must know the initial pose of the device. This can be done by aligning some of the axes with gravity [[12], [15]]. In practice, this can be a hard task to accomplish but most smartphones already implemented this with their APIs. Some papers [12] have used random heading agnostic coordinate frames for training the networks. This has been accomplished by randomly rotating the ground truth trajectories on the horizontal plane.

3.2 Convolutional neural network integrator

Both inspected neural networks are approximating the first integral from acceleration to speed or velocity. One of the neural networks estimates speed which in this thesis means non-directional velocity and is a single-dimensional scalar value while the other calculates multi-dimensional directional velocity. After the speed or velocity is calculated it can be used for classical algorithms like EKF in Cortés et al. [14] or just cumulatively summed to the position like in Herath et al. [12].

Networks input is a window of acceleration and angular velocities. This creates a problem where the networks do not get samples from a specific starting point, but as pointed out by Cortés et al. [14] the convolutional layers at the start of the networks make them invariant to shifts of the starting positions.

3.2.1 Calculating speed from accelerations

In Cortés et al. [14] a simple CNN was used which calculated the speed of the device with a neural network. The neural network takes as input the six acceleration values over two seconds at 100 Hz sampling frequency (a tensor of shape 200x6) and condenses the window into single output which describes the speed of the device. Because this is a one-dimensional value it does not have any sense of direction but is just the length of the velocity vector. The neural network is displayed in Figure 3.1.

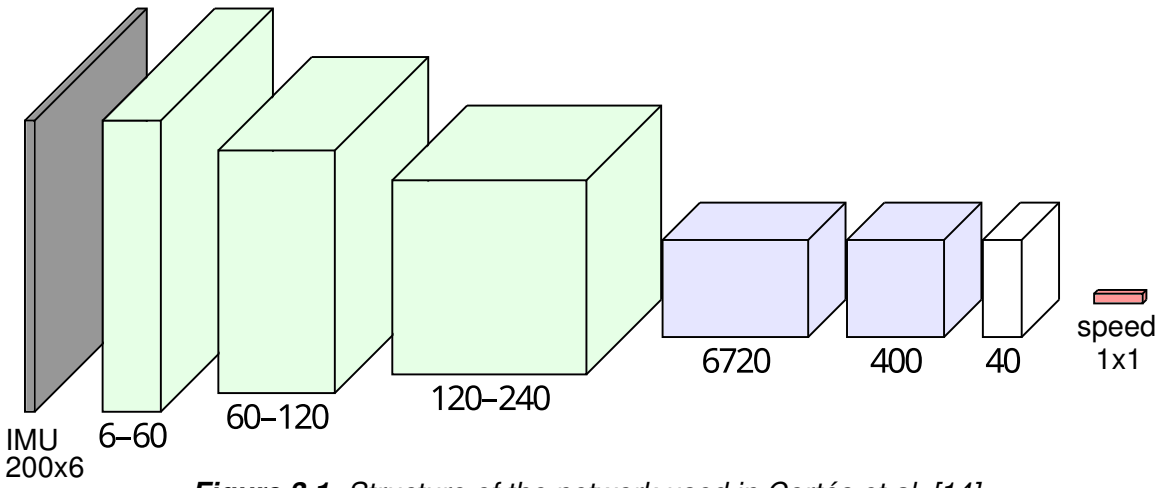

Figure 3.1. *Structure of the network used in Cortés et al. [14]*

The extended Kalman filter was built to take an input vector of acceleration and angular velocities to its prediction step and the prediction was updated with the speed value from the CNN as a pseudo-measurement. This was repeated for every sample in the input data making the process have a stride of one. The network only produces single-dimensional scalar value output which is harder to utilize and requires more complicated methods to calculate position.

3.2.2 Calculating velocity from accelerations

An alternative solution can be found from Herath et al. [12] where they have three different backbone suggestions of which this thesis inspects the modified ResNet version due to it being the most consistent according to their tests. The architecture of this network is described in Figure 3.2.

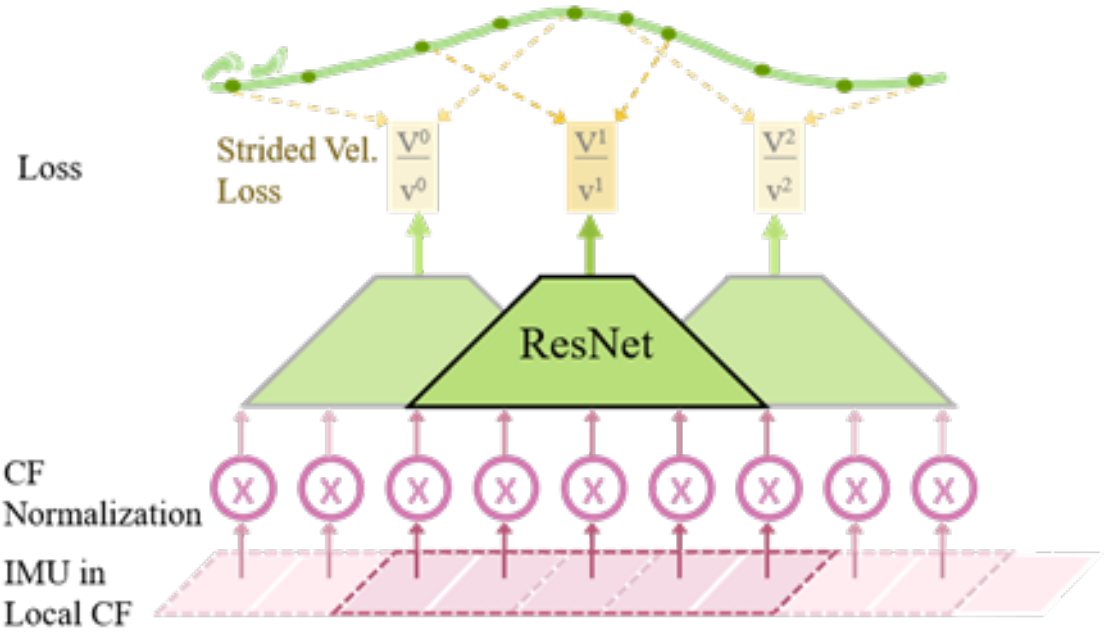

Figure 3.2. *Structure of the network used in the ResNet version of Herath et al. [12]*

In Herath et al. [12] the system also takes in three acceleration values and three different angular velocities in a window. The system has a sampling frequency of 200 Hz and a window size of one second which makes the tensor 200x6. The output vector of the system is instead a vector of velocities (x,y-velocities) instead of speed. This makes it possible to use a simpler algorithm to calculate the position e.g. cumulative sum. In their work, they simply calculated a cumulative sum for this output vector and got good results.

Unlike the simpler network introduced in 3.2.1 this network will only create an output value for every five frames, making it have a stride of five. This makes it possible to have less frequent updates and less noise in the velocity outputs, smoothing the input of algorithms like EKF, but with a possibility of losing some high-frequency changes. Having a larger stride also results in lower computational requirements.

Unfortunately, the implementation presented could only estimate 2-dimensional velocity output and ignored the gravity-aligned z-axis. Most likely this could be changed to a 3-dimensional model easily by changing the later layers' output dimensions. The paper did not discuss this possibility but their shared code in their Github repository [16] contained a configuration that could change the network from 2-dimensional output to 3-dimensional requiring re-training.

4. EXPERIMENTS AND RESULTS

Both methods contained at least partial reference implementation in Github and were publicly available and this thesis tried to replicate their results. For Cortés et al. [14] the repository [17] contained a PyTorch-based neural network model and scripts required to train it. For ease of use, the model and scripts were ported to python3 and a newer version of PyTorch from python2. Unfortunately, the repository did not contain reference implementation about the EKF but after communication with the authors, they gave a reference to their other research which contained EKF implementation with pseudo-velocity updates, though it contained also visual odometry [18]. The repository for Herath et al. [12] contained a complete PyTorch model and code to calculate odometry and its different error metrics.

4.1 Datasets and training

Both models had their own or closely related datasets. Unfortunately, Herath et al. [12] dataset was only partially published due to security concerns which most likely will make training and reproducing their results impossible. On the other hand, they published pre-trained models of which the ResNet version was used to similar results as in their paper.

ADVIO-dataset [11] used by Cortés et al. [14] was published along with a fully trained model. For some reason, this trained PyTorch model did not have the same layout as the model in their published sources (it had a different kernel size). Because of this, the model was re-trained as the paper described. The model they had was trained using the ADVIO dataset as in Cortés et al. [14] for 2000 epochs with Adam optimizer. The optimizer's learning rate was set to $1e-6$, the loss function was MSE loss and 10-fold cross-validation was used for the training dataset where the data is split into 90 % training and 10 % validation datasets. Training loss and validation loss are illustrated in Figures 4.1 and 4.2.

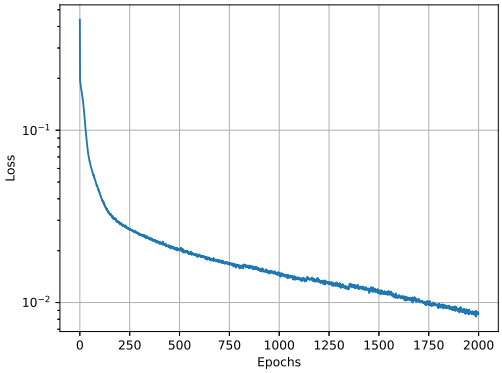

Figure 4.1. *Training loss*

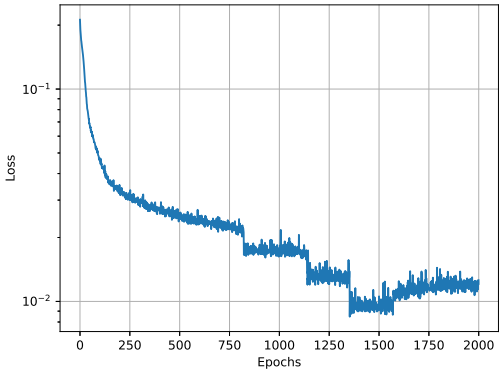

Figure 4.2. *Validation loss*

With the testing dataset, $0.009\ m/s$ MSE was reached with the trained model, and the pre-trained model had an MSE of $0.0127\ m/s$ which is close to the reference of $0.13\ m/s$. Most likely the authors have left an old version of the model to their Github. Training the network took approximately 35 seconds per epoch, making it take approximately 19 hours. This process could be optimized by fixing the training loop to not be most likely memory bottlenecked at the data loader and transferring training itself to GPU which without fixing the other bottlenecks did not help.

4.2 Odometry

This section describes the odometry results for both methods. Firstly, the results of the ResNet method are discussed and then the results of the EKF-based method.

4.2.1 ResNet velocity calculation

With the Herath et al. [12] CNN output velocities, calculating odometry information succeeded by multiplying the velocities with their timestamp difference $\Delta t = t_{current} - t_{previous}$ and cumulatively summing them to calculate 2-dimensional positions for every output.

This thesis is using error metrics absolute trajectory error (ATE) and relative trajectory error (RTE) defined by [19] and [12]. In them, they defined ATE as a root mean squared error between ground truth and predicted trajectory and RTE as an average root mean squared over a fixed time interval (which in this paper is 1 minute). For the published datasets, the unseen subject test set got average errors of 5.10 m ATE and 3.39 m RTE, which is close to what was reported in Herath et al. [12] (5.14 m ATE and 4.37 m RTE). The scenario with the closest ATE and RTE to mean values is displayed in Figure 4.3

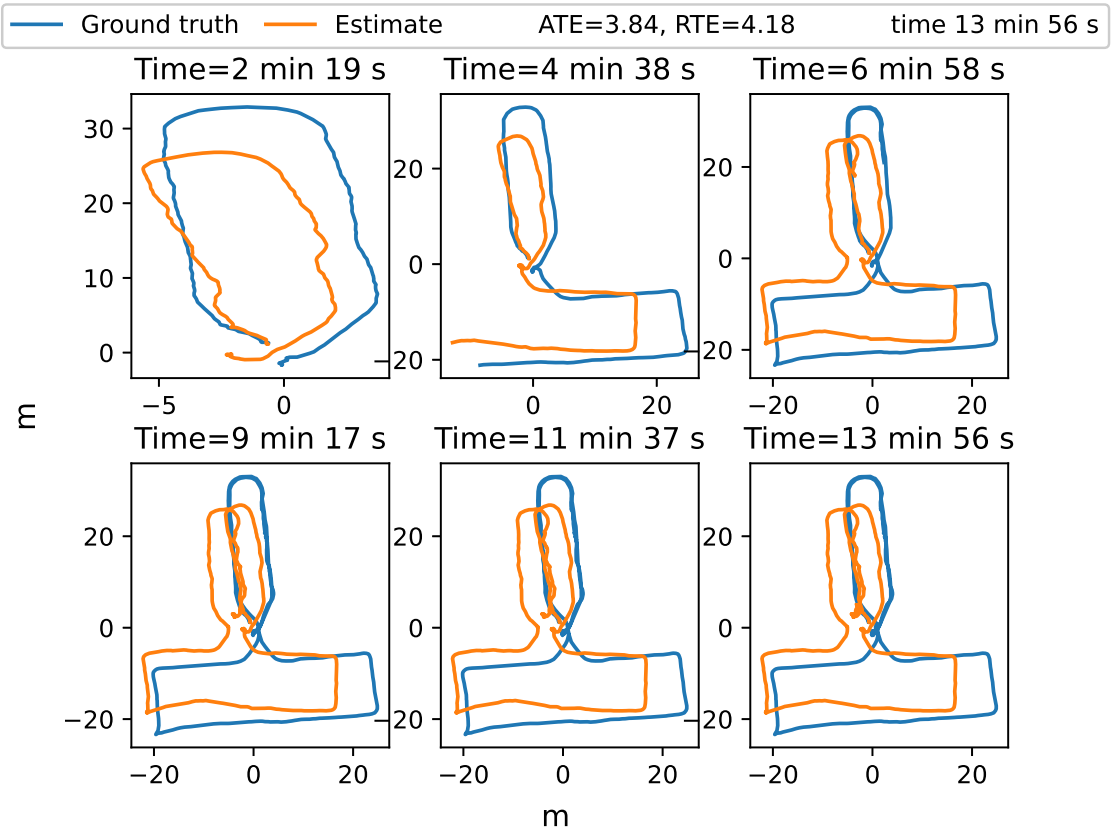

Figure 4.3. *Scenario 0's estimated trajectory with the closest ATE and RTE to mean over time.*

Distribution of the errors per scenario was on most scenarios less than 5 m (Figure 4.4) for both ATE and RTE with scenario lengths in time being between 5 minutes 40 seconds to 15 minutes and 3 seconds averaging to 5 minutes and 29 seconds (Figure 4.5).

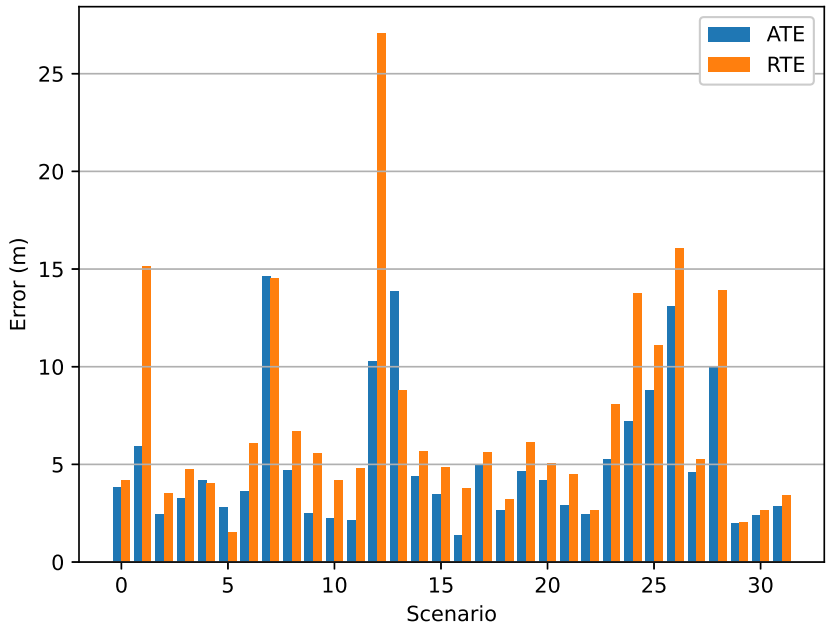

Figure 4.4. *ATE and RTE errors for all scenarios*

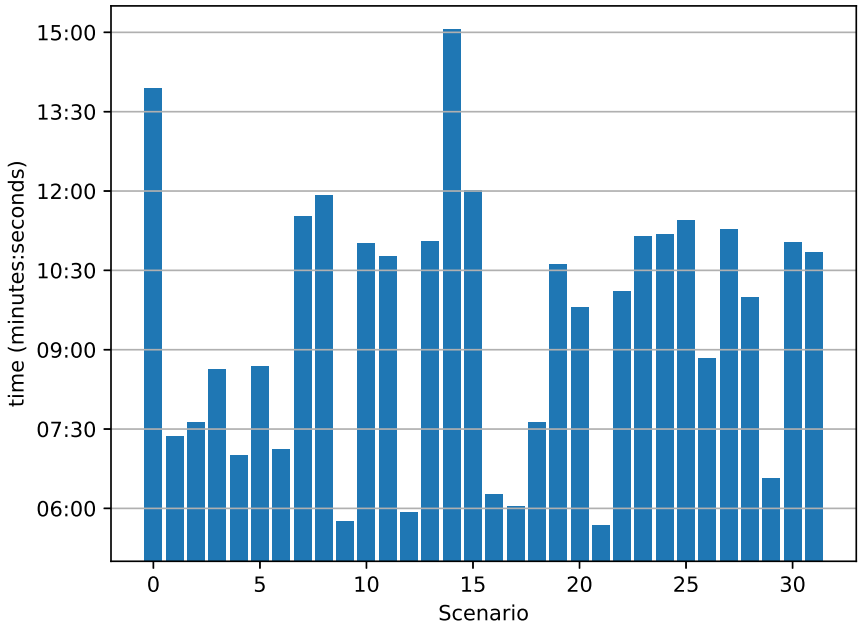

Figure 4.5. *length of scenarios*

The two worst cases with either largest ATE or largest RTE were scenarios 7 and 12 where scenario 7 had the largest ATE of 14.60 m and RTE of 14.50 m and scenario 12 had the largest RTE of 27.08 m and ATE of 10.26 m. The two scenarios are illustrated in Figures 4.6 and 4.7.

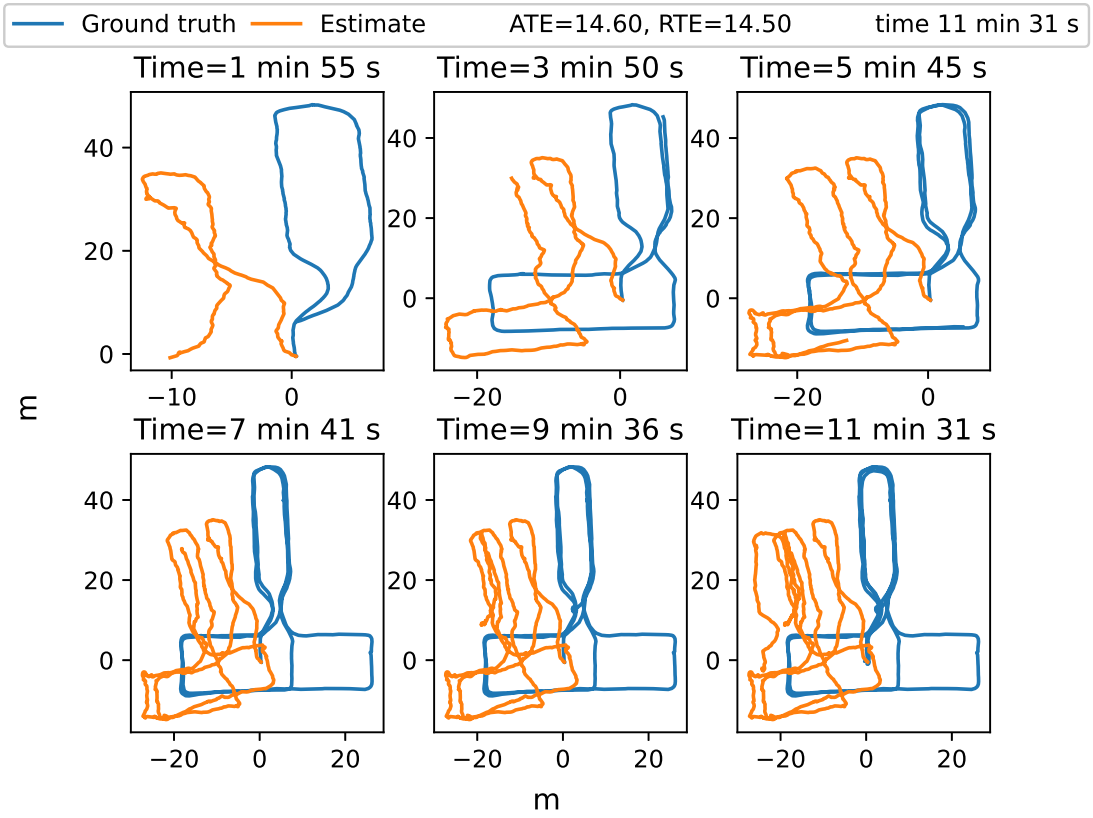

Figure 4.6. *Scenario 6's estimated trajectory with the worst ATE over time*

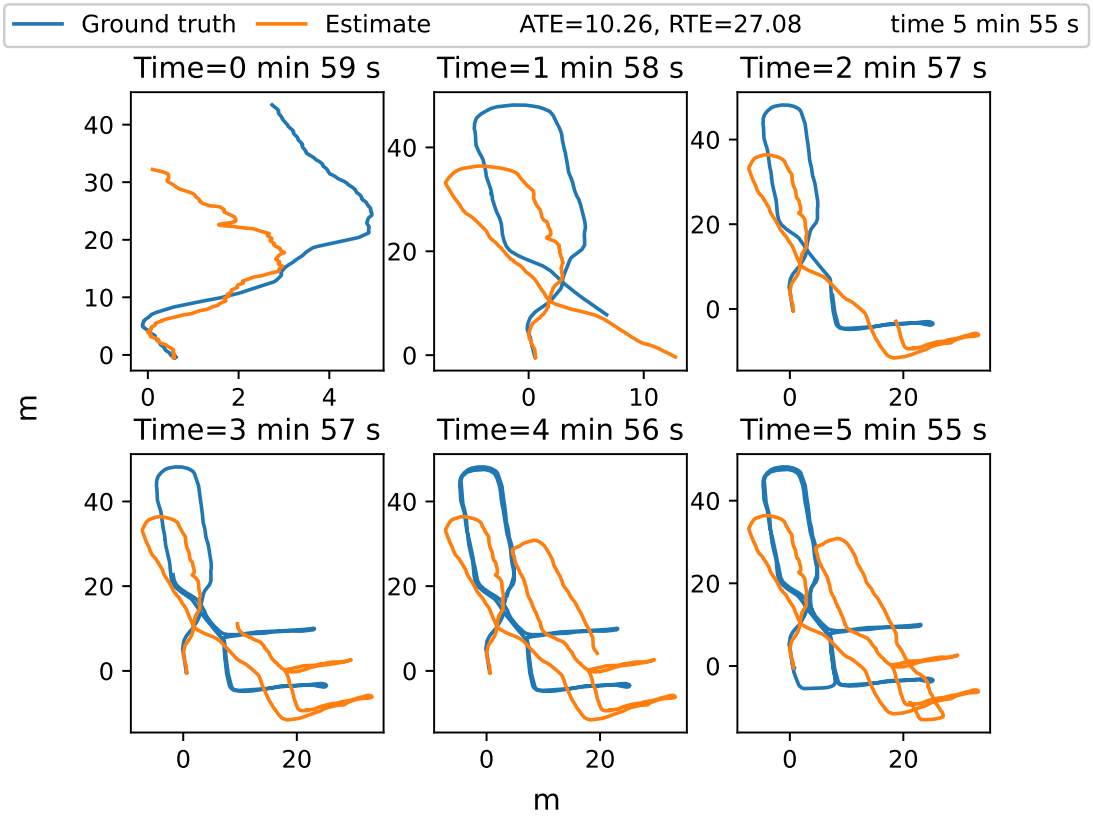

Figure 4.7. *Scenario 12's estimated trajectory with the worst RTE over time*

From Figures 4.6 and 4.7 can be seen that the overall shape of the estimate is approximately correct but due to some sort of drift they have some translation and mostly rotation errors. Translation error can be caused by incorrect bias removal described in 3.1 or just CNN not having enough training samples. A more dominant feature is visible in both images as a rotational error. This makes the whole trajectory drift in rotation slowly over time. This is visible especially in Figure 4.7.

4.2.2 EKF and pseudo-velocity

Unfortunately, even after reaching out and getting a reference implementation [18] for a similar application from the authors, this paper was not able to reproduce the odometry results calculated with EKF. In experiments, the EKF position exploded to large numbers almost immediately after starting the algorithm. This was observed with both the pre-trained model and the self-trained model (Figure 4.8).

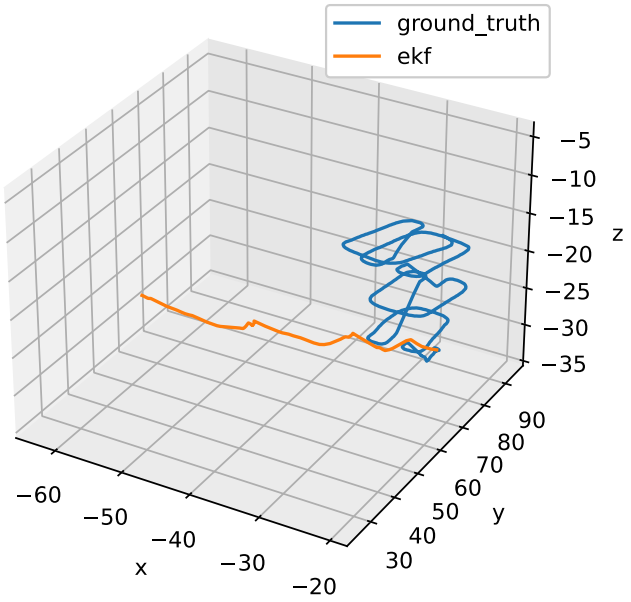

Figure 4.8. *Estimated trajectory for EKF based approach*

This can be caused by many things like a poor choice of covariances for the EKF. In their previous papers, they have not well defined the pseudo-speeds covariance and have only said that it should be large compared to the process noise to make it non-informative compared to the other information sources like EKF control input (acceleration and angular velocity) and of course, the problem may have been just a programming error.

5. DISCUSSION

Although using CNN to calculate odometry from IMU sensor data seems to produce good results there are a couple of drawbacks to this approach. Firstly, the process is highly dependent on the sampling frequency being the same as when the network was trained. This makes it not a good approach in cases where there is a variable sampling frequency or multiple different constant sampling frequencies. Secondly, estimating covariance for CNN output vector is hard, which makes it harder to use in cases that would require covariance like EKF though this can most likely be estimated experimentally. Thirdly, these approaches are meant to be used with pedestrian dead reckoning problems, and generalizing them to other use cases might not work well, especially without a large enough dataset.

Other problems might arise with wheeled vehicles which could have velocity over zero for a long amount of time. These neural networks would not be feasible to learn with any kind of dataset due to them not remembering anything beyond a couple of seconds.

Lastly, it is not certain how these models would behave if they were trained with data from one sensor model and tested with other. It might be that using these approaches won't work well without some external noise normalization which would be applied to sensor data before processing. This makes it possibly hard to scale across many different devices.

Processing requirements to run these systems in real-time are feasible with modern processors but it is unknown how well these would run on an average smartphone and especially, how much battery they would use. Battery usage has been one of the main reasons why to use IMU odometry over e.g. GPS so testing this would be important to understand how applicable these approaches are. Due to these problems, these methods may not be suitable for mobile applications.

6. CONCLUSION

Both the EKF-based and ResNet-based approaches have their positive and negative sides. Implementing the ResNet-based neural network is relatively easy but training it might be harder and more important than training the network which tries to guide EKF not to drift. The ResNet method seems to have a precision of meters even after multiple minutes and if augmented with other sensor data most likely would have good precision although being harder to integrate with other sensors than the EKF method. Unfortunately, reproducing the results of the EKF method was not successful, but most likely if there was a working example of it reproducing its results and integrating them with other sensors would be easy. The code used in this thesis is available at Github[1].

With current methods, we can reach meter-level accuracy on pedestrian dead reckoning and track the device over tens of minutes without any position fixes. Integrating new sensors to have even better accuracy in these systems is possible with algorithms like EKF.

Using neural networks to calculate odometry may not be the ultimate solution. Generalizing these methods to e.g. wheeled vehicles can be difficult and at least these models do not apply to them. Using the same models between different devices might not be possible and the battery usage of these neural networks might cause problems for mobile applications.

To summarize IMU-odometry has advanced greatly during the last couple of years due to advancements in neural networks and new non-classical methods. There are still a lot of open issues. Development of these methods should continue further to find more accurate and more efficient methods of calculating IMU-odometry.

[1] https://github.com/juha-ylikoski/imu-dead-reckoning

REFERENCES

[1] Seiskari, O., Rantalankila, P., Kannala, J., Ylilammi, J., Rahtu, E. and Solin, A. HybVIO: Pushing the Limits of Real-time Visual-inertial Odometry. eng. *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Piscataway: IEEE, 2022, pp. 287–296. ISBN: 1665409150.

[2] *Reconstructing the motion of a tossed iPhone*. [Online; accessed 26-May-2022]. 2022. URL: https://rotations.berkeley.edu/reconstructing-the-motion-of-a-tossed-iphone/.

[3] Solin, A., Cortés, S., Rahtu, E. and Kannala, J. Inertial Odometry on Handheld Smartphones. eng. *2018 21st International Conference on Information Fusion (FUSION)*. ISIF, 2018, pp. 1–5. ISBN: 0996452761.

[4] Foxlin, E. Pedestrian tracking with shoe-mounted inertial sensors. eng. *IEEE computer graphics and applications* 25.6 (2005), pp. 38–46. ISSN: 0272-1716.

[5] Galben, G. New Three-Dimensional Velocity Motion Model and Composite Odometry-Inertial Motion Model for Local Autonomous Navigation. eng. *IEEE transactions on vehicular technology* 60.3 (2011), pp. 771–781. ISSN: 0018-9545.

[6] Li, M.-G., Zhu, H., You, S.-Z. and Tang, C.-Q. UWB-Based Localization System Aided With Inertial Sensor for Underground Coal Mine Applications. eng. *IEEE sensors journal* 20.12 (2020), pp. 6652–6669. ISSN: 1530-437X.

[7] Simanek, J., Reinstein, M. and Kubelka, V. Evaluation of the EKF-Based Estimation Architectures for Data Fusion in Mobile Robots. eng. *IEEE/ASME transactions on mechatronics* 20.2 (2015), pp. 985–990. ISSN: 1083-4435.

[8] Chen, S., Chang, C.-W. and Wen, C.-Y. Perception in the dark—development of a ToF visual inertial odometry system. eng. *Sensors (Basel, Switzerland)* 20.5 (2020), pp. 1263–. ISSN: 1424-8220.

[9] Bloesch, M., Burri, M., Omari, S., Hutter, M. and Siegwart, R. Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback. eng. *The International journal of robotics research* 36.10 (2017), pp. 1053–1072. ISSN: 0278-3649.

[10] Wikipedia contributors. *Tango (platform) — Wikipedia, The Free Encyclopedia*. [Online; accessed 29-March-2022]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Tango_(platform)&oldid=1050773437.

[11] Cortés, S., Solin, A., Rahtu, E. and Kannala, J. ADVIO: An authentic dataset for visual-inertial odometry. eng. (2018).

[12] Herath, S., Yan, H. and Furukawa, Y. RoNIN: Robust Neural Inertial Navigation in the Wild: Benchmark, Evaluations, New Methods. eng. *Proceedings - IEEE International Conference on Robotics and Automation*. 2020, pp. 3146–3152. ISBN: 1728173957.

[13] Chen, C., Zhao, P., Lu, C. X., Wang, W., Markham, A. and Trigoni, N. OxIOD: The Dataset for Deep Inertial Odometry. eng. (2018).

[14] Cortés, S., Solin, A. and Kannala, J. DEEP LEARNING BASED SPEED ESTIMATION FOR CONSTRAINING STRAPDOWN INERTIAL NAVIGATION ON SMARTPHONES. eng. *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. Vol. 2018-. IEEE, 2018, pp. 1–6. ISBN: 1538654776.

[15] Yan, H., Shan, Q. and Furukawa, Y. RIDI: Robust IMU Double Integration. eng. (2017).

[16] *RONIN github*. `https://github.com/Sachini/ronin`. 2022.

[17] *Deep Learning Based Speed Estimation github*. `https://github.com/AaltoVision/deep-speed-constrained-ins`. 2022.

[18] *HybVIO github*. `https://github.com/SpectacularAI/HybVIO`. 2022.

[19] Sturm, J., Engelhard, N., Endres, F., Burgard, W. and Cremers, D. A benchmark for the evaluation of RGB-D SLAM systems. eng. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 573–580. ISBN: 1467317373.