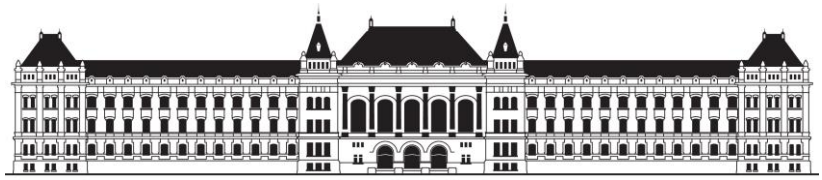


BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
DEPARTMENT OF TELECOMMUNICATIONS AND MEDIA INFORMATICS

ROBOT POSITION ESTIMATION USING DEEP NEURAL NETWORK WITH IMU DATA

Deep learning in practice based on Python and LUA
(BMEVITMAV45)
Homework

MIRA Team:

BALÁZS NAGY – O668S4
MÁRK FEKULA – VUW9RS
GERGELY HUNYADY – FPZXIL

Budapest, 2018. 12. 09.

TABLE OF CONTENT

1. PROJECT DESCRIPTION

The aim of this study is to examine if it is possible to estimate a robot position from IMU (Inertial Measurement Unit) data. During our research we are using a Phidget Spatial sensor to collect data.

Our research efforts contain the following steps that are documented in the further chapters. You can read a short review on our survey connected to the relevant literature in Chapter 2. As a next step we collected accurate position and orientation data using a special motion capture system presented in Chapter 3. Working with this data we tried to apply different supervised machine learning techniques (namely Long short-term memory networks - LSTM and 1D Convolutional Neural Networks - CNN) to estimate the absolute or relative position and the orientation of the robot. Our findings and experimentations on these methods can be found in Chapter 4. Finally, in Chapter 5 we discuss the achieved results and the further development potential.

2. LITERATURE OVERVIEW

In a previous study two orientation estimation algorithm have been implemented on the same Phidget Spatial sensor. The trivial algorithm to use for this problem could be Kalman filter, however this method requires the accurate system model and is computationally too expensive to use in an embedded system. The first method in our scope is based on the concept of complementary filters [1]. The algorithm combines the gyroscope and accelerometer data according to fix weights, and the magneto data is used to compensate the offset error in the gyroscope data. **Error! Reference source not found.** shows a performance comparison between the complementary filter approach and other algorithms based on a single sensor.

Another algorithm is a Qvaternion based method which was presented by Valenti et al. [2]., where every rotation in a 3-dimensional space can be described by an axis and a rotation around it. After implementation, the algorithms have been tested in the MoCap system and the errors achieved by them are shown in Table Table 1.

Table 1: Error comparison

	Min err[°]	**Max err[°]**	**Avg error[°]**	**Dev [°]**
Complementary	0.01	20.36	9.38	7.28
Qvaternion	0.01	21.27	9.44	7.27
Magneto	0.02	21.35	9.51	7.29

Based on the previous study we assume that the orientation can be estimated with explicit numeric methods. Yet the position estimation with numeric methods without using an external observer is an unsolved task. Due to the white noise disturbance on the measured signal after two integration stage, a drift will appear in the signal. If there is an explicit numeric solution to a problem, a neural network could learn it. But deep neural networks can also provide solutions to problems which has not an explicit solution so far.

As it has been noticed before double integration of acceleration data won't yield adequate results: "This isn't because the accelerometers themselves are poor, but because the orientation of the sensor must be known with a high degree of accuracy so that gravity measurements can be distinguished from the physical acceleration of the sensor. Even small errors in

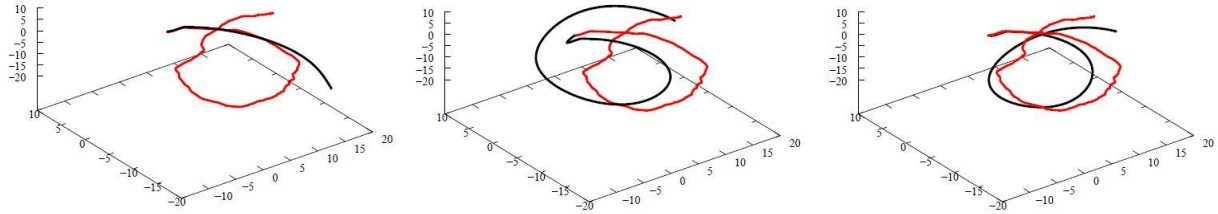
the orientation estimate will produce extremely high errors in the measured acceleration, which translate into even larger errors in the velocity and position estimates." [3].

In the paper of Shih et al. [4] the main focus is determining whether double integral or machine learning produces smaller position error, using only accelerometer data. With a constant sampling time - Δt, the integration simplifies to a linear combination.

$$Pos = \iint_1^n a\, dt = \Delta t^2 \sum_{i=1}^{n} \sum_{j=1}^{i} a_j = \Delta t^2 (n a_1 + (n-1) a_2 + \cdots + a_n)$$

Thus, they tested if using machine learning is even a viable solution for this problem or not. On 1000 training and 1000 test generated samples, linear combination had an error of 6.5%, while linear regression resulted only 2.5%. After that, they tried using neural networks with Scaled Conjugate Gradient, and Bayesian Regularization to get a robust model. For this test the data was collected from a smartwatch. After fine tuning the models the average error was 13%. This was accomplished with 5 hidden layers, which implicates the complexity level of the problem with this few input parameters.

Hermann Mayer et al. [5] worked on a network, that learns a 3D trajectory based on a few samples. They used LSTM networks with Evolino, which is an evolutionary method, therefore it is better for an error surface with many local minima. Using an LSTM network with 10 memory cells, and evolving Evolino networks for 60 generations, a smooth and accurate trajectory was created from the training data.



1. Figure 3D plots of trajectory generated after every 20 generations [5]

Leung et al. [6] developed a neural network for smartphones, which classifies human activity based on accelerometer and gyroscope data. It is crucial, that while adequate accuracy is maintained, the network has to be lightweight and battery friendly as well. To meet these requirements, they tested LSTM and CNN based solutions. As the task mainly depends on classification, CNN proved to be more accurate, but LSTM was faster and more power efficient, while not lagging behind so much. The results of this research project are shown in Figure 2.

	LSTM	CNN
Walking	87.3%	97.2%
Walking Upstairs	93.4%	93.0%
Walking Downstairs	97.4%	97.4%
Sitting	81.7%	87.6%
Standing	84.6%	84.0%
Laying	95.0%	97.0%
TOTAL ACCURACY	89.7%	92.6%

2. Figure Network accuracy for different classes [6]

3. DATA UNDERSTANDING

3.1. Data collection method

In order to collect measurement data we built up our own measurement setup which consist of three main parts. Our client program - realized in LabVIEW - can connect to the Motion Capture system to track and log the position and orientation of the sensor mounted on a robot. It also connects to the robot via wifi to control its movement. And the program connects to an Inertial Measurement Unit (IMU) via USB to collect and log data from it. The sematic of MIRA (MoCap, IMU and Robotino Analyser) can be seen on Figure 3. A Phidget Spatial sensor had been used during our study, which provides acceleration [g], gyroscope speed [°/s] and magnetic field strength [G] data along 3 axes. The exact parameters of the sensor can be found on the official website of the sensor [7]. The Robotino - shown on the middle of Figure 3 - is a small robot with omnidirectional driving system made by FESTO.

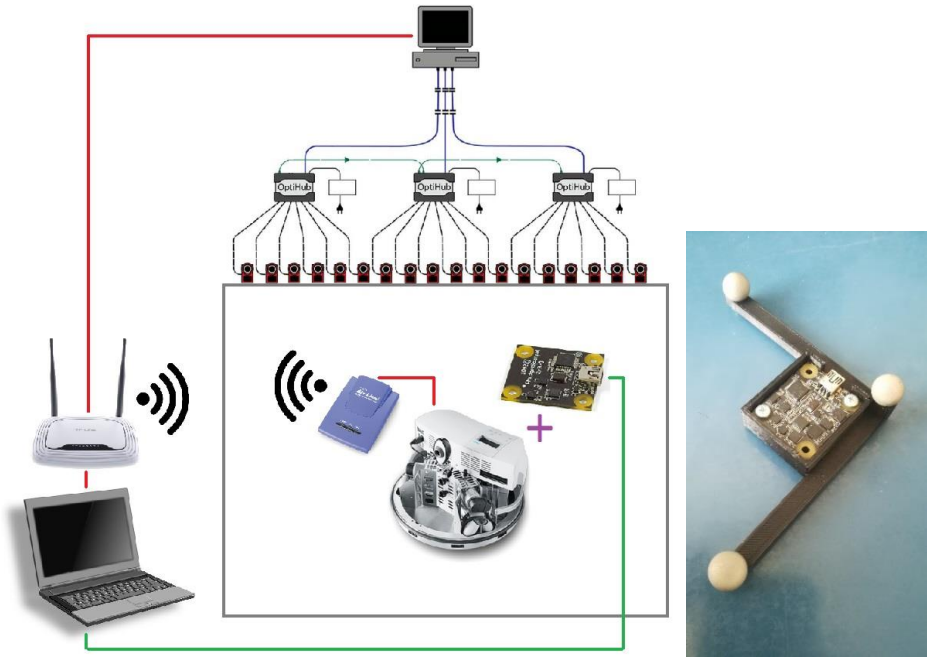


Figure 3: Sematic of MIRA and a markered sensor

The OptiTrack MoCap system is capable of tracking retroreflective markers. The system contains 18 cameras mounted in a room. Markers, shown on the right side of Figure 3, were attached to the sensor so the MoCap can track the position and orientation of the IMU. As a next step, we mounted the sensor on the top of a Robotino. After the successful setup and system calibration the robot can move in the measurement space shown on Figure 4.

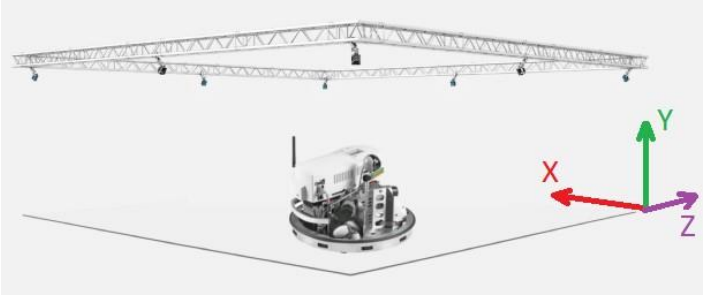


Figure 4: MoCap illustration with axis to better understand the data stream

3.2. Data visualization

A logging session during a measurement produces three files. MIRA outputs a file including the IMU data, a file including the MoCap data and a file including the robot data. We needed multiple attempts to finally record error-free data, because synchronizing the IMU and the MoCap turned out to be a difficult task.

We recorded special measurements for calibration and later data pre-processing reasons. We also recorded complex routes for training, validating and testing our models, two of these measurements MoCap output are shown on Figure 5.

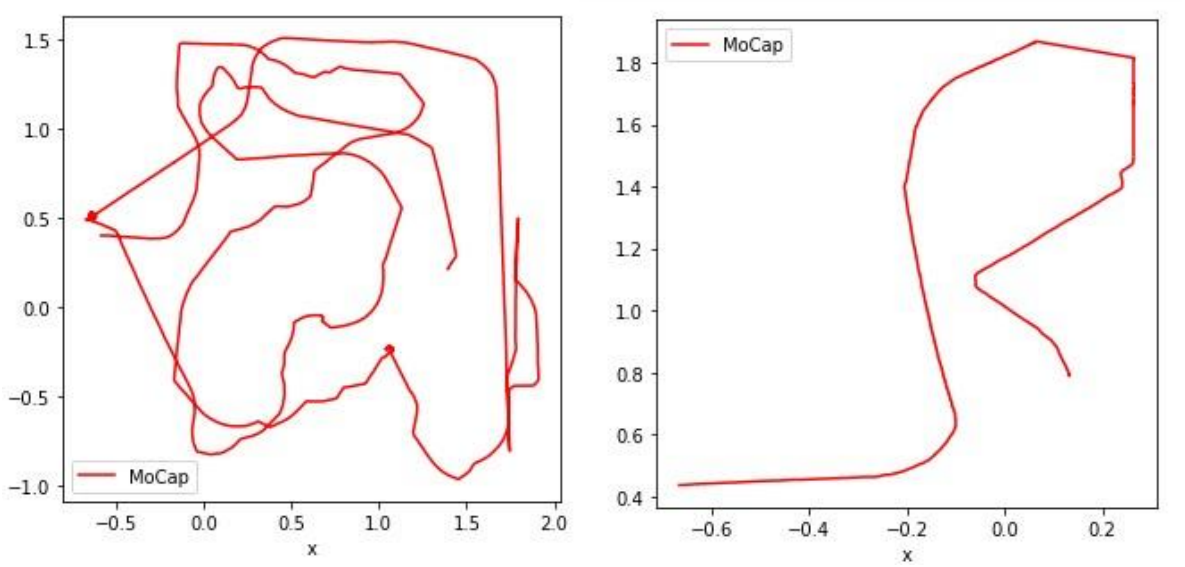


Figure 5: Recorded routes on the x-z plane for training and testing

The raw data collected from the MoCap system is visualized on Figure 6, and the acquired datastream from the IMU sensor is shown on Figure 7.

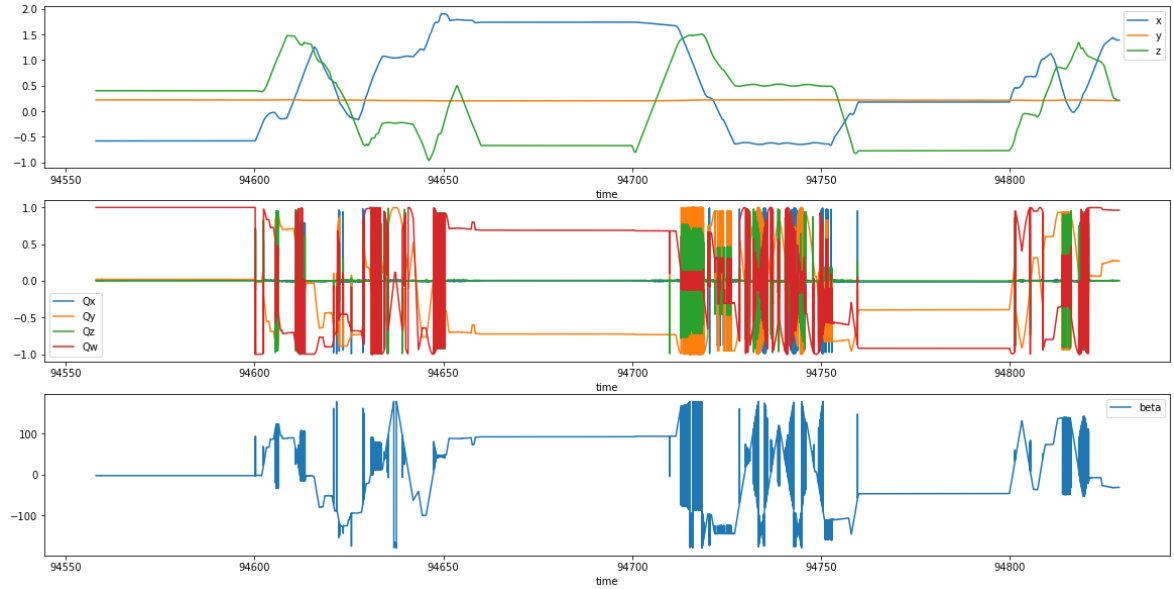


Figure 6: Raw data from the MoCap system

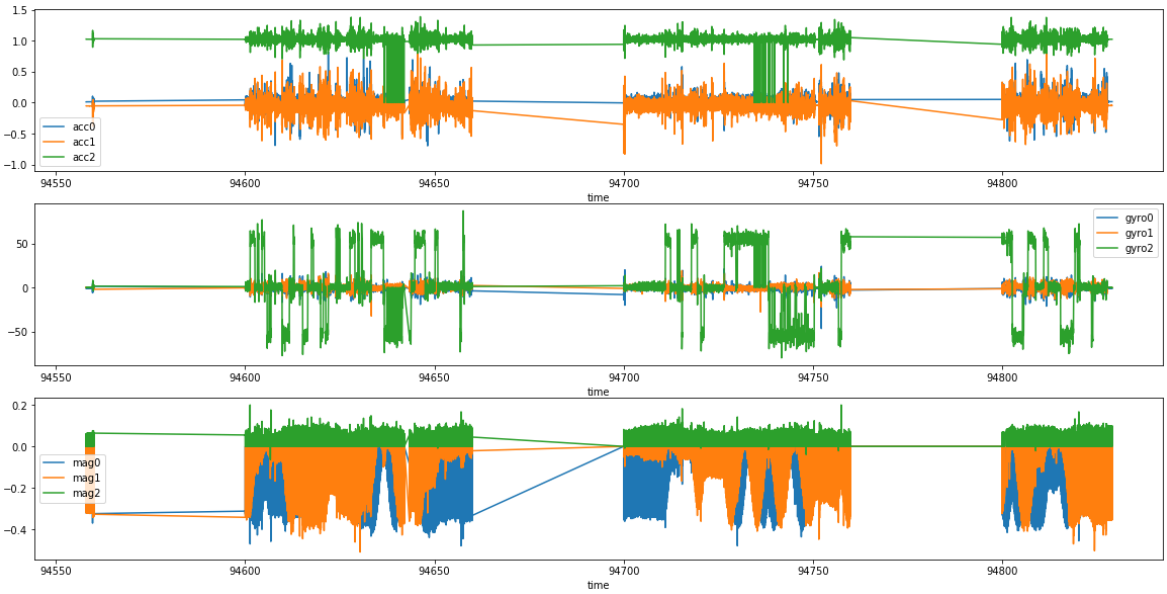


Figure 7: Raw data from the IMU (accelero, gyroscope, magnetometer)

3.3. Data preparation for deep learning models

Data preparation is an essential step in deep learning to feed the net with consistent data flow. Figure 7. shows that the data from the IMU sensor is not ideal and we need to deal with these issues, and to do so apply calibration and normalization steps.

First, during the measurement the robot lost wifi connection. In this case the communication was blocked between the laptop and the robot, so the robot stopped. It caused some gap which can be seen on the timeseries. We simply dropped out these false measurement points from the pool hence the robot did not move during these steps.

Before data normalization, which includes sensor calibration as well, some ground rules were made. The algorithm must be reproducible before every measurement session. The task is challenging because the sensor is mounted on a robot.

To calibrate the accelerometer a separate measurement file was recorded in which the robot stands still. With the help of these file we can eliminate the effect of the gravity vector and mean normalize the accelerometer.

To calibrate the magnetometer the usually used method is to rotate the sensor around each axis and use feature scaling and mean normalization on the data. Using a mounted sensor on a robot we made a separate measurement file in which the robot rotates around the z axis. We are not capable of a full calibration, but it is enough to calibrate the necessary axis. Using this file, we can calibrate the offset and scaling of each axis of the magnetometers. The magnetomer sensor is the slowest sensor and if the data request is out of synchronization the provided data is zero. We ignore these data rows and drop out them from the pool.

The gyroscope calibration is tricky. At this point we looked at the training data and calculated some “magic” constants to scale the dataflow. These constants are hard coded and can be used on the test data. Further development potential to use datasheet data to use as constant but for now this solution is enough.

On the other side some additional value was calculated from the reference data. Using delta time and delta positions made the algorithm robust and we can keep the data normalized. For example, a robot can move a long distance but due to the mechanical part it has limita-

tions so in a short delta time it can only move a short distance. The conclusion is that using relative values instead of absolute values is more sustainable.

Finally, we normalized the orientation values as well from [-180, 180] to [-1, 1] interval. The preprocessed IMU datas can be seen in Figure 8.

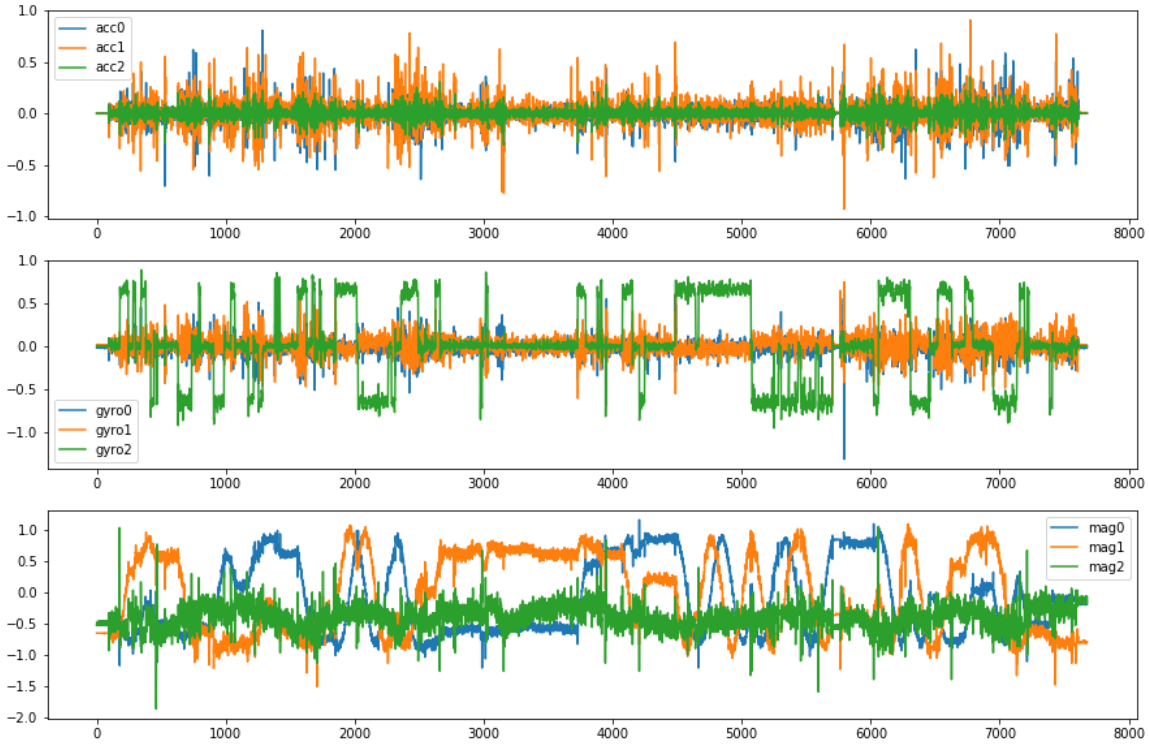


Figure 8: Preprocessed data from the IMU (accelero, gyroscope, magnetometer)

4. EXPERIMENTS WITH DIFFERENT DNN MODELLS

We tried out two different approaches, which are frequently used for predicting from timeseries. In this chapter we discuss the tried and tested model architectures and results.

4.1. LSTM

An LSTM [8] model with 50 cells was built to predict orientation first. This model is designed to predict only one output signal. Based on our experiment results this method is unable to predict the needed labels, but we assume our implementation also suffers from some bug, which we could not completely rule out unfortunately. We did not manage to reach a breakthrough in this direction, so we concentrated to our more promising networks.

4.2. 1D CNN

Using 1D Convolutional Neural Networks, first we tried to predict the absolute position values. This approach first seemed to be nearly perfect when the network had the previous positions from the MoCap data as input (shown on the left side of Figure 9), but the network completely fails if the previous position inputs are the previous predicted outputs from the model itself (shown on the right side of Figure 9). This behaviour is easily understandable if

the delta-t timestep is small enough, as the previous correct position, which in this case is also an input, is really close to the predictable position.

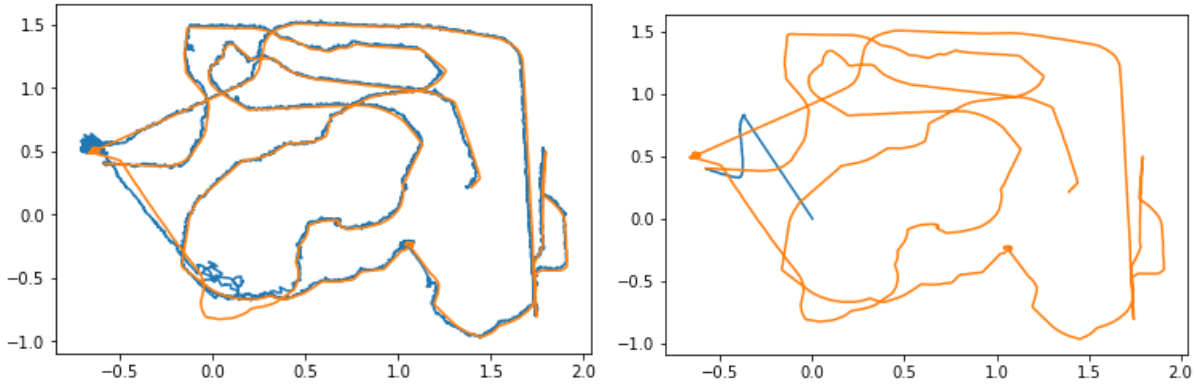


Figure 9: Predicting the absolute position

From this time on we mainly focused on predicting the small delta position change values, which resulted much better output signals after adding the small movement pieces together, as shown on Figure 10. Multiple variations (more layers, more neurons) were tested and the figure shows the most promising result. The measurement used for testing the model starts with rotation in the same place, and this might mess up the initial predicted moving direction, but the similarity between the original and the predicted tracks are easily discoverable.

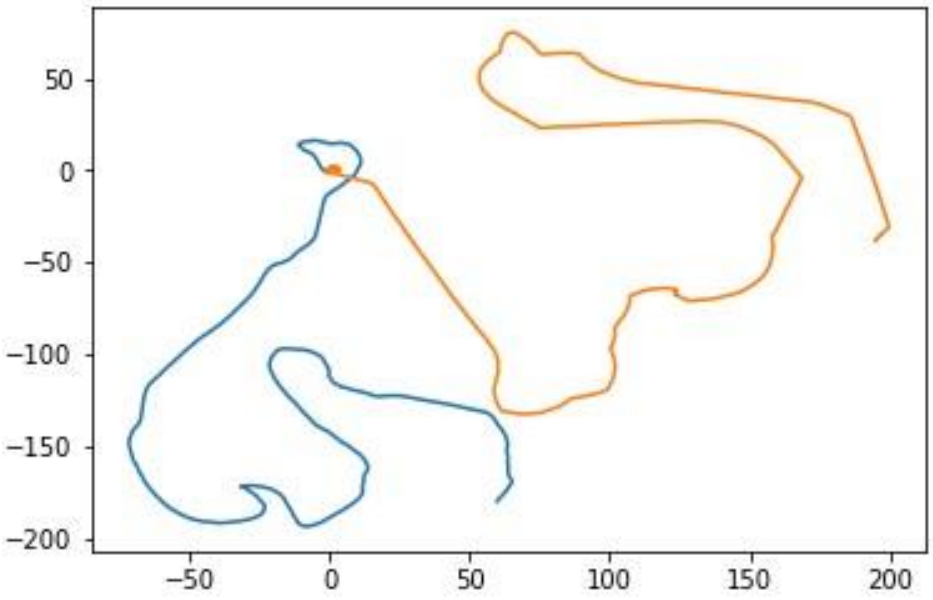


Figure 10: Predicting the relative delta values

5. SUMMARY

As we see, promising results have already been reached, but we need further research and investigation to get applicable models. Our model managed to learn and restore some of the required features to estimate the position and orientation from raw data from an IMU. According to our findings, recording and processing data to get proper training database for this special task takes more time than expected. However, we assume that significantly more data is needed to train more complex and deeper networks, which could grab more details from the input stream and produce more accurate outputs. We also plan to examine the ef-

fect of using regularization techniques in the future, which we didn't have time yet to try. Another important next step could be an automatic hyperparameter optimization, as we only used manual optimization.

As a summary, according to our results we assume that position and orientation estimation is achievable by the technique we used in this study, but more data and deeper network is needed for future work.

6. REFERENCES

[1] P.-J. V. d. Maele, "Pieter-Jan.com," 2013. [Online]. Available: http://www.pieter-jan.com/node/11. [Accessed 13 10 2018].

[2] V. RG., D. I. and X. J., "Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs," *Sensors*, pp. 19302-19330, 6. August 2015.

[3] "Using Accelerometers to Estimate Position and Velocity," [Online]. Available: http://www.chrobotics.com/library/accel-position-velocity. [Accessed 14. 10. 2018.].

[4] M. Shih and J.-T. Hsieh, "Predicting Short-Range Displacements From Sensor Data," 2015.

[5] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll és J. Schmidhuber, „A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks," in *International Conference on Intelligent Robotics and Systems*, 2006.

[6] B. Leung, „Optimising Human Activity Classification using Deep Learning for Smart Devices," [Online]. Available: http://www.cse.unsw.edu.au/~z5014700/docs/TOR.pdf. [Hozzáférés dátuma: 14. 10. 2018.].

[7] P. Inc., "Phidgets," 2017. [Online]. Available: https://www.phidgets.com/?&prodid=32. [Accessed 11 10 2018].

[8] S. a. S. J. Hochreiter, „Long Short-term Memory," *Neural computation*, %1. kötet9, pp. 1735-80, 1997.