ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie

Institut für Neuroinformatik

MotionLogger - Investigating the inference of tap locations using smartphone motion data

Project neural networks, Ulm University

Vorgelegt von:
Sebastian Vendt
Sebastian.Vendt@uni-ulm.de
752219

Gutachter:
PD Dr. Friedhelm Schwenker

2019

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

This work investigates the correlation between motion data from smartphone sensors and user input on the touch screen. Therefore a data acquisition app is developed and a convolutional neural network is trained on the generated dataset. The performance of the network is tested on a number pad with 12 buttons.

Table of contents

1 Introduction

Today's smartphones play with their high computing capacity and various sensors an important role in our daily life. They are an essential part in Communication, banking or sporting to name a few applications. Thus they combine a huge amount of highly private information on a single end.

This work targets the user input on the touch screen as one of the most sensitive data. It exploits several motion sensors, which are highly accurate nowadays and investigates the correlation between user input and the movement of the smartphone. When tapping on the screen to input data (e.g. on a soft keyboard or number pad) the phone slightly tilts every time a finger touches the screen. This rotation depends on the position of the tap. A convolutional neural network is trained and evaluated if the motion data from tapping on different locations can be separated well enough to infer the user input with significant accuracy.

2 Related work

Only few paper about inferring tap locations from motion data were published so far. They all have in common that they use hand-crafted features which were fed into a classifier (e.g. SVM, Random Forest, MLP) [1, 2, 5, 6, 8, 9]. They also directly predict a mapped button rather than general coordinates. Owusu et al. only uses data from the accelerometer, Cai and Chen only uses the device orientation data and Xu, Bai, and Zhu as well as Shen et al. combine the accelerometer and the orientation data. Since the device orientation sensor became deprecated in Android 2.2 (API level 8)[10] the gyroscope together with the accelerometer was used in this work. Although gyroscopes are more suitable for high frequency rotations and accelerometers for low frequency rotations, both contain information about the current movement. So both sensors were included as in the publication of Miluzzo et al. This project introduces the use of deep convolutional neural networks which hasn't been investigated before. It also approaches the idea of inferring keystrokes from motion data in a more general way by predicting screen coordinates. In a second step these coordinates can be mapped to any interface of your choice. The process of manually selecting features was skipped as Chen and Xue did it in their research. The paper is closely related and estimates human activities from motion data[3]. To reduce complexity and since most of the user input happens on soft keyboards this project's investigation is limited to the lower half of the screen.

3 Technical Background and feasibility

This chapter describes the technical background making this approach possible. Secondly the initial feasibility analyze of this project is documented.

Both sensors measure in a three-dimensional coordinate system that is defined relative to the device's screen. The horizontal X axis points to the right, the vertical Y axis points upwards while the Z axis points toward the outside of the screen face[10]. This is depicted in figure 3.1. Three distinct movements of the phone exist. Those have three corresponding screen positions (A, B and C in fig. 3.1). All other movements from different locations are a (attenuated) combination of those movements. The Graphs in Figure 3.2 show the sensor readings from tapping on the right and left side of the screen (A and B). The traces were averaged across 10 taps. Comparing the two plots it can be seen that they are well distinguishable from each other. Especially the trace of the Y axis reflects the inverted movement of the phone between tapping on the left or the right side. Similar distinguishable properties can be found in the traces of the accelerometer which are in the appendix together with the remaining plot of the gyroscope from position C.

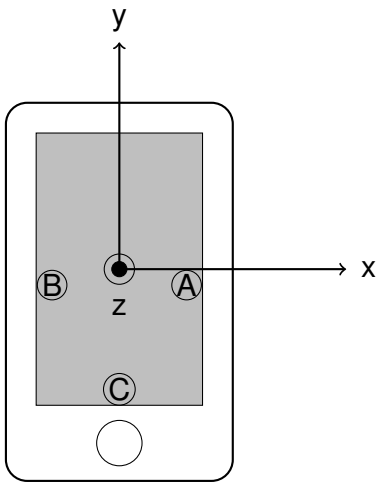
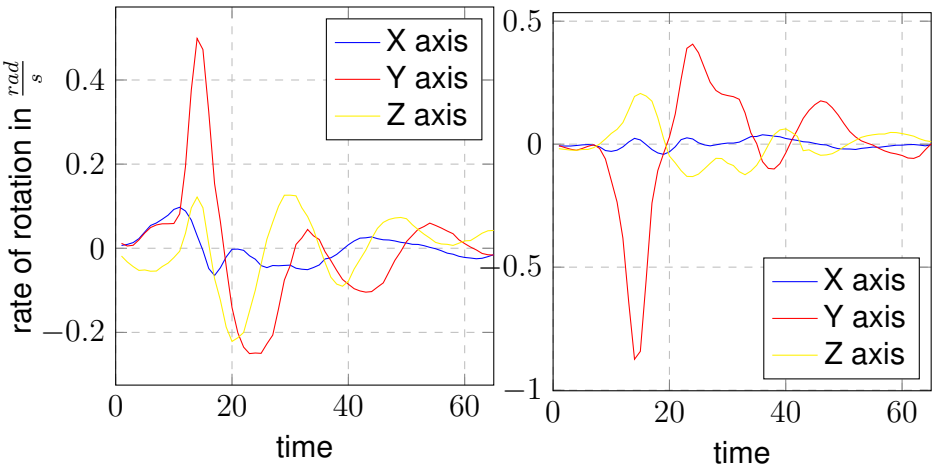


Figure 3.1: Coordinate system relative to the device's screen



(a) gyroscope trace pos. A

(b) gyroscope trace pos. B

Figure 3.2: Averaged motion stream of the gyroscope from tapping on two positions A and B

4 Data acquisition

This chapters covers the process of acquiring the necessary data for training and evaluating the neural network. While many public image datasets are available, only few datasets of motion sensors like accelerometers and gyroscopes exist. Most of these datasets are used in human activity recognition research and cannot be used for the investigation at hand. This required the generation of a custom tailored dataset specific for this study. Therefore an android application, in the following called app, was developed to record data from the sensors as well as the tap locations on the screen. The saved data from the application is then preprocessed by a Matlab script for further use.

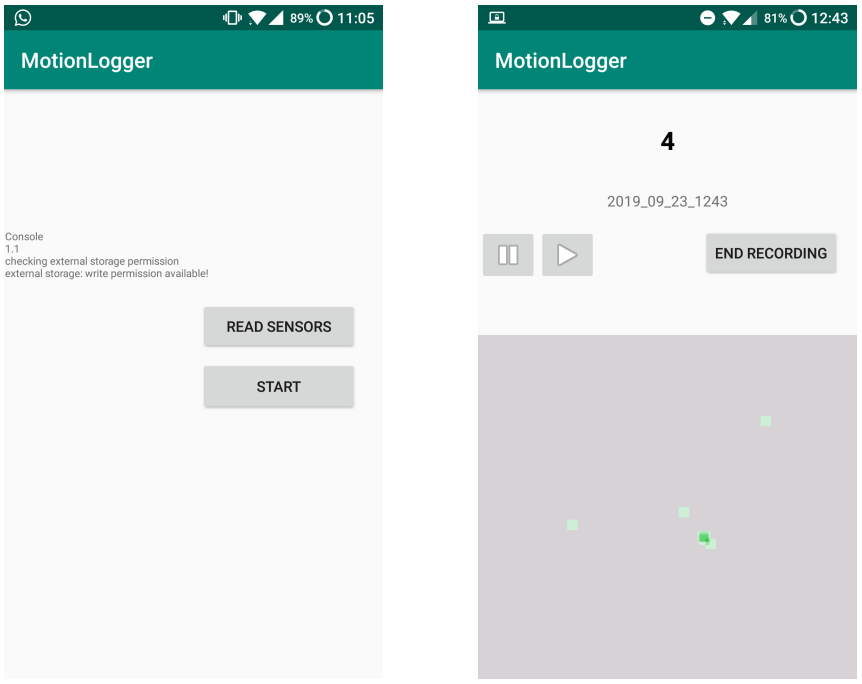
To characterize the dataset one needs to be able to make some statements according to the speed and the strength of the tapping. Thus histograms of the distribution of the absolute maximum of each frame of each sensor and each axis are generated. The distribution of the duration of touch events, describing the speed of tapping, is plotted together with the distribution of the time between two touch events. One will find negative values in the latter one which is the result of overlapping touch events (e.g. while the first finger hasn't been lifted a second one touches the screen). If further datasets will be generated these are the criteria to compare those.

4.1 Android application development

The general purpose and the requirements for the app are described in the following. The main task is recording the data from both motion sensors and the corresponding tap location. Furthermore it should provide feedback of already tapped locations in some form of a heat map. This is necessary so the taps are evenly distributed across the whole area. It should provide the number of already recorded taps as well as the possibility to pause and resume the data acquisition. Timestamps of the taps should be recorded so the additional information as described in chapter 4 can be extracted from the dataset.

The requirements described above are implemented in the app called *motionLogger*. It consists of two interfaces shown in 4.1. It was developed using android

studio[1] from Google Inc. The app saves the recorded data to a CSV[2]-file with



(a) main interface (b) recording interface with heat map

Figure 4.1: User interface of motionLogger

13 columns. (A list and description of all columns can be found in the appendix in Fig. A.1) For measuring rotation of the device the sensor *TYPE_GYROSCOPE* was used. It is a three-dimensional vector representing the rate of rotation around the every axis in rad/s. For measuring acceleration the sensor *TYPE_LINEAR_ACCELERATION* was used. It is a three-dimensional vector as well, representing acceleration along each device axis, excluding gravity.

4.2 Matlab preprocessing

The files generated from the android app are further processed in Matlab. The continuous stream of sensor events is separated into windows with fixed length. Each window is located around the beginning of a tap event (pointer_down event) and one window includes 60 samples of each sensor and each axis. 28 samples lie before and 31 after the pointer_down event. These values were chosen by hand

[1] https://developer.android.com/studio
[2] Comma separated values

using plots of the sensor data. The window size is large enough to cover short as well as long tap events. It is also short enough to only fully include the movement of a single tap. The frames can overlap if two touch events were lying close enough together.
Since the raw data had to be generated by hand, a form of data augmentation was used to enlarge the generated dataset. When slicing the sensor stream into windows the anchor of the window was shifted by ten samples in both directions, resulting in 21 frames for one touch event. This also makes the network more invariant to the exact location of the touch event within the window. This is an important aspect since during the actual attack the ground truth of a pointer down event is unknown and needs to be estimated. The feasibility of detecting tap events was proposed in [8, 9] and will not be part of this work.
The dataset is separated into a training- (30 %), validation- (20 %) and test-set (10 %). This separation is done before augmentation so the network has never seen any of the validation and test data during training. Every set is saved as a .mat file and has two fields: data and labels. *Data* is a three-dimensional matrix, the first two define one frame (60x6) while the last denotes the total number of frames. *Labels* is a two-dimensional matrix. The coordinates x and y are stored in the first while the second dimension again denotes the total number of frames. The origin of the coordinate system is shown in figure 4.2. The lower left corner has the coordinates (1080, 980)

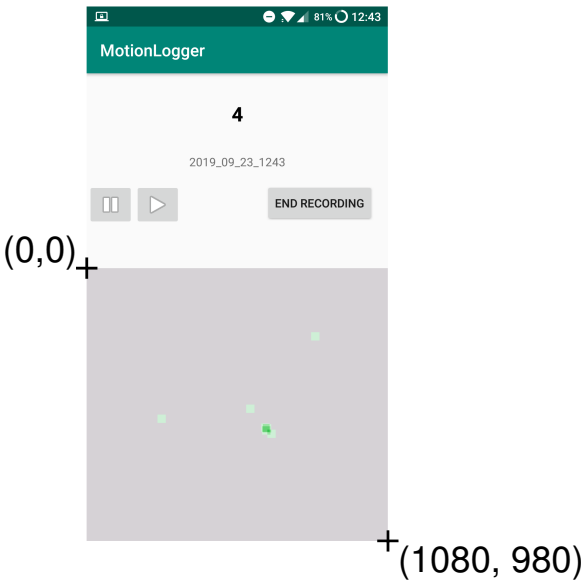


Figure 4.2: coordinate system origin of the labels

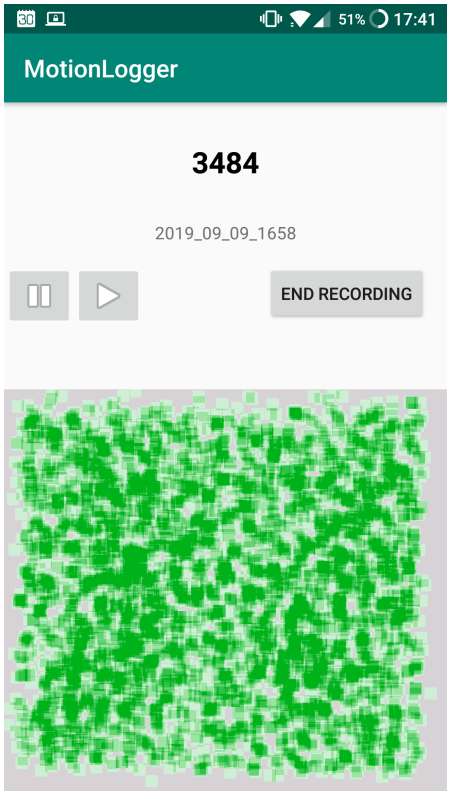


Figure 4.3: Distribution of the tap locations

4.3 Dataset creation

Finally one dataset was generated using the app. 3484 tap events were recorded in six sessions of three minutes each, followed by a three minute break. After three sessions the break was extend to six minutes. The phone, a OnePlus 2, was held in portrait mode while sitting on a chair having both elbows placed on the arm rests. Both thumbs were used to tap on the screen and time between to taps was kept short. After a few a longer break was introduced. This should simulate typing words on a soft keyboard. The final distribution of the tap locations is shown in figure 4.3. The highest available sampling rate was used (200 Hz). Sensor events (e.g when the value changes) are reported asynchronously from the system but since the time between two events only differed in a few nanoseconds no further processing was done here. After augmentation the dataset contains 73101 samples. Detailed statistics of this dataset can be found in the appendix. Randomly picked samples from the dataset were compared with the curves discussed in chapter 3. Similar patterns could be found.

5 Neural network design

This chapter covers the procedure of design and optimization of the neural network. It estimates normalized screen coordinates from motion data vectors with fixed length.

Initially a general structure was defined and subsequently its hyperparameter were evaluated in a random search. In a second step the best performing network was selected and its learning rate and number of epochs were fine-tuned.

The network was implemented in *julia*[1] using its neural network framework *Flux*[2]. All networks were trained on a NVIDIA Tesla K40c GPU.

5.1 Basic architecture

To incorporate that motion data is highly correlated in the temporal domain several convolutional layers were used to extract low- and high level features [4]. Every axis is convolved using a 1D Convolution kernel with shape (*conv1*, 1) and (*conv2*, 1) as in [7]. Low level features like edges and flat areas should be extracted first before those get merged in the last convolutional layer. This has a 2D Kernel with shape (*conv3*, 6) covering all 6 axis so the network learns dependencies among the X,Y and Z dimensions of both sensors. The first two convolutions are zero-padded, followed by a Maxpooling layer (*maxpool1* and *maxpool2*) similar to the architecture proposed in [3, 7]. No Maxpooling was implemented after the third convolutional layer because I suspected loss of information. The dimension after the third convolution is roughly 13x1 (depending on the used kernel sizes).
The last convolution is followed by three dense layers (*dense1*, *dense2* and *dense3*) in a pyramid structure. The second dense layer has 600 inputs, and the final layer has 300 inputs and two outputs with sigmoid non-linearity for both normalized coordinates. For all other layers RELU was used. Dropout is incorporated in the fully-connected layers to prevent over-fitting. The described architecture is shown in figure 5.1.

[1] https://julialang.org/
[2] https://fluxml.ai/Flux.jl/stable/

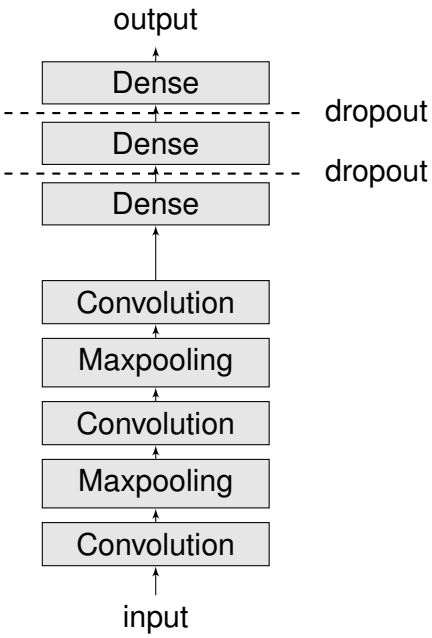


Figure 5.1: basic network architecture

I used the mean-square-error to calculate the loss of the network:

$$[h]E(\hat{y}, y) = \frac{1}{N}\sum_{n=1}^{N}\sqrt{(\hat{y}_{n,x} - y_{n,x})^2 + (\hat{y}_{n,y} - y_{n,y})^2} \tag{5.1}$$

L1-regularization is included with $\lambda = 0.0005$.

$$L(\hat{y}, y) = E(\hat{y}, y) + \lambda \sum_{p=0}^{P} |w| \tag{5.2}$$

w denotes the parameter vectors of the network, P the total number of parameters and N the batch size.
Accuracy is calculated according to the future use case. The examined screen is divided into 12 buttons, similar to a number pad. The networks predicted coordinates are counted as a hit when they land within the same button as the ground truth.

5.2 Hyperparameter tuning

Even though the general structure described in section 5.1 is determined, many hyperparameter exist which are subject of further evaluation. In the following all optimized parameters with their possible configurations are listed. Due to the large dimension of the parameter space random search was chosen. In the first round randomly picked configurations (excluding decay step) were trained for 40 epochs. The best performing architecture, measured by accuracy, was then trained in a second round for 100 epochs with three decay steps and three learning rates.

1. **training epochs**
2. **momentum rate**
   0.9, 0.92, 0.94, 0.96, 0.98, 0.99
3. **features in each convolution layer**
   32|64|128, 32|32|32, 64|64|64, 96|192|192[7]
4. **dropout rate**
   0.1, 0.3, 0.4, 0.6, 0.8
5. **1D convolution kernel *conv1* and *conv2***
   (3,1) (3,1), (5,1) (5,1), (7,1) (7,1), (7,1) (5,1), (5,1) (3,1)
6. **2D convolution kernel *conv3***
   (2,6), (3,6)
7. **pooling dimension *maxpool1* and *maxpool2***
   (2,1) (2,1), (3,1) (3,1)
8. **learning rate**
   1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001
9. **decay step**[3]
   20, 40, 60

[3] used formula in the appendix

6 Results

In the random search 40 configurations were trained. I stopped the search quite early although only a very small part of the parameter space was evaluated considering the limited time and since most networks converged to a similar accuracy.

Five networks scored less than $50\,\%$, 17 networks between $50\,\%$ and $60\,\%$ and 15 networks achieved more than $60\,\%$. The best performing network reached $65.79\,\%$ (on the validation set). The following table 6.1 lists the five best networks and their configurations. One can notice that the best scoring networks have highly various architectures. The best performing network was then trained in a second round to optimize the decay step. Also a local search around the given architecture for the best learn rate was again conducted. The network was trained with learn rates 0.03, 0.01 and 0.003 and decay steps 20, 40 and 60 for 100 epochs. The number of epochs were chosen by hand from several runs and is approximately the threshold at which all networks converge. The final best accuracy is $68.22\,\%$ on the validation set and $60.16\,\%$ on the test set. The best performing network was trained with learn rate 0.03 and decay step 60. The learning progress is shown in figure 6.1.

acc.	ρ	features	dropout	conv kernel	pooling	η
$65.79\,\%$	0.99	32|64|128	0.3	(5,1)(5,1)(2,6)	(3,1)(3,1)	0.003
$65.66\,\%$	0.96	32|64|128	0.8	(3,1)(3,1)(3,6)	(3,1)(3,1)	0.03
$64.96\,\%$	0.99	64|64|64	0.3	(7,1)(7,1)(3,6)	(2,1)(2,1)	0.01
$63.67\,\%$	0.92	32|32|32	0.1	(5,1)(5,1)(2,6)	(3,1)(3,1)	0.1
$63.40\,\%$	0.9	96|192|192	0.6	(3,1)(3,1)(3,6)	(2,1)(2,1)	0.3

Table 6.1: configurations of the best scoring networks

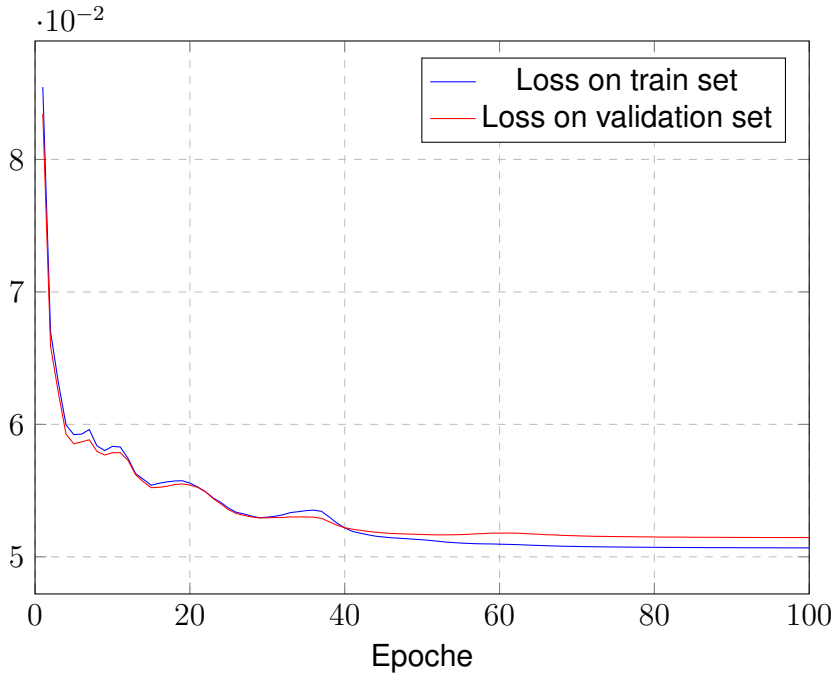


Figure 6.1: Loss plotted over the epochs in second round of optimization

7 Conclusion

The presented results should be handled with care. It shows that this side channel attack using motion sensors is in principle possible but considerably more work is necessary. Roughly $60\,\%$ accuracy on the number pad is already far beyond the chances of a random or brute-force search. The probability of correctly inferring a button by a random guess is, on average, only $\frac{1}{12} = 8.3\%$. Also consider that this is the accuracy for a correct prediction. Further work could include neighboring buttons increasing the inference precision even more. The achieved accuracy of this work falls within the same range as the results from the other publications. Miluzzo et al. reaches $67.1\,\%$ but uses the whole screen and far more training data (40.000 taps). Cai and Chen with their TouchLogger reach $71.5\,\%$ with 16 buttons in landscape mode. The best results among the cited paper were achieved by Shen et al. with $83.9\,\%$ on a number pad covering half the screen. They also had a very large training set with 97.000 taps.

More time should to be invested using more datasets acquired from different people. The network should also be tested on motion data from user input on real interfaces rather than the app. In this work the presented way of acquiring data was chosen because it offered a fast and easy way to implement the requirements. But it is also a compromise because it abstracts the real use case and introduce a possible source for misleading results. Also many more steps like the tap event detection needs to be implemented before a full attack can be established. They all will come with their own problems.

The optimization showed that the general network architecture independently of its parameter performs well on the dataset. More research should be done in finding a more suitable network structure to further increase inference accuracy. Several ways of data normalization could be tested. Padding for the third convolution was resigned since it would contradict with the idea that one kernel establishing a relation between the six axis but one could try padding only in the temporal domain.

A Appendix

A.1 Formulas

Equation A.1 shows the formula for the learning rate decay over the epochs. η denotes the initial learning rate, e the epoch, ε the new learning rate, d the decay step and δ the decay rate.

$$\varepsilon = \eta\delta^{\frac{e}{d}} \tag{A.1}$$

A.2 Usage of the software

motionLogger The app has two interfaces shown in Figure 4.1. In the main interface one can read the available sensors and start a new recording session with the button *Start*. This opens a second interface for the recording. It has a tap counter at the top followed by the name of the file, the data is written to. On the left side two buttons are available to start and pause the recording. With *End recording* the user finishes the data acquisition and returns to the main interface where some information of the last recording activity is displayed in the console. Figure 4.1 also shows the heat map. The sensitive area is separated in light Gray from the rest of the screen. Tapping at a certain location results in a light green rectangle. This is added to the already existing rectangles, so the more taps are recorded at one location the darker the color of the heat map.

The table A.1 list all columns of the CSV-File together with a description.

julia scripts The script net.jl can be run from the command line with several options. A list with all available options and additional help text can be printed with

```
julia net.jl --help
```

column name	description
GYRO_DIFF	Time between two gyroscope events in ns
GYRO_X	Rate of rotation around the x axis in rad/s
GYRO_Y	Rate of rotation around the y axis in rad/s
GYRO_Z	Rate of rotation around the x axis in rad/s
ACC_DIFF	Time between two accelerometer events in ns
ACC_X	Acceleration force along the x axis (excluding gravity) in m/s^2
ACC_Y	Acceleration force along the y axis (excluding gravity) in m/s^2
ACC_Z	Acceleration force along the z axis (excluding gravity) in m/s^2
TOUCH_DOWN	marks the beginning of a touch event (pointer down event)
TOUCH_X	x coordinate of the tap location
TOUCH_Y	y coordinate of the tap location
ID1	Tracking ID of the first pointer
ID2	Tracking ID of the second pointer (in case of overlapping touch events)

Table A.1: Description of the columns in the CSV-File

The script logs many important values (e.g. current loss, accuracy) to a log file for easily monitoring the training progress. The default configuration (–runD) is the configuration of the best performing network from chapter 6. If this flag is not set the random search, described in 5.2, is conducted.

A.3 Plots

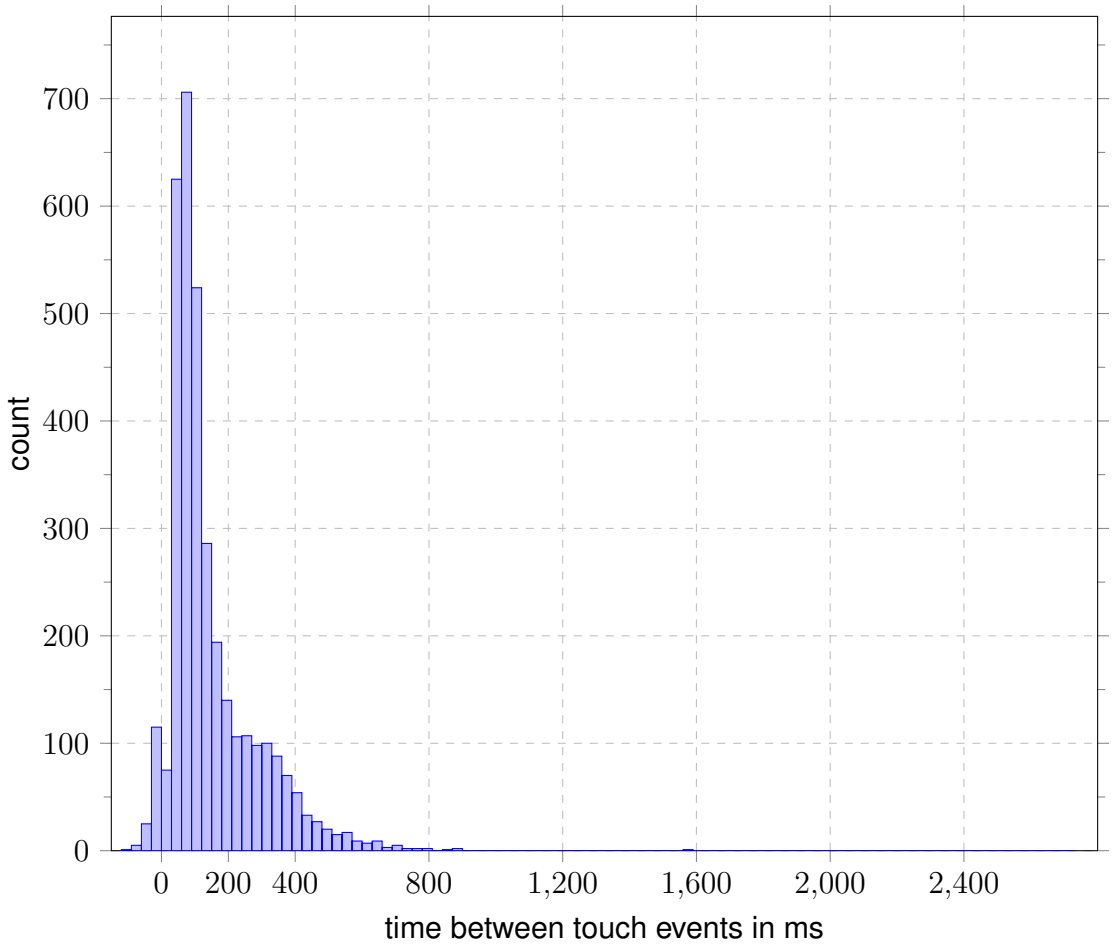


Figure A.1: Histogram of the time between touch events

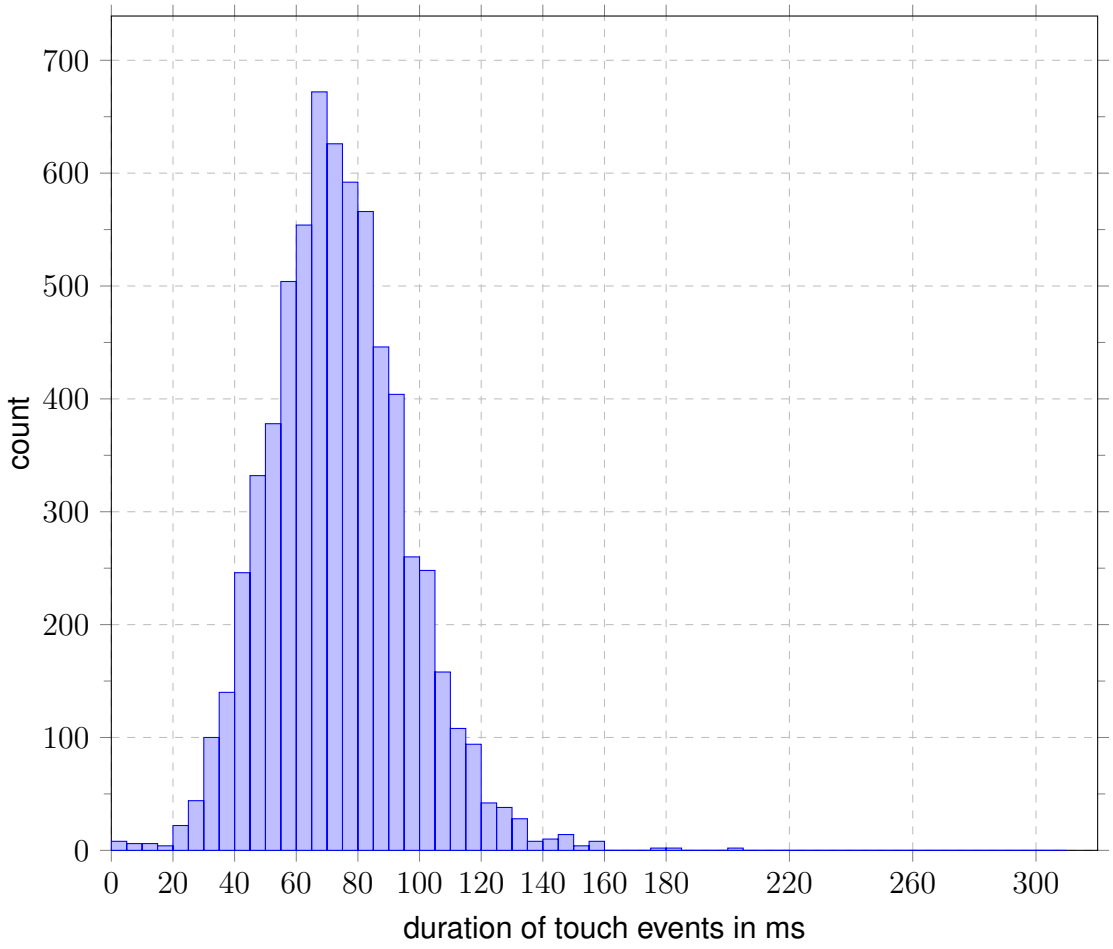
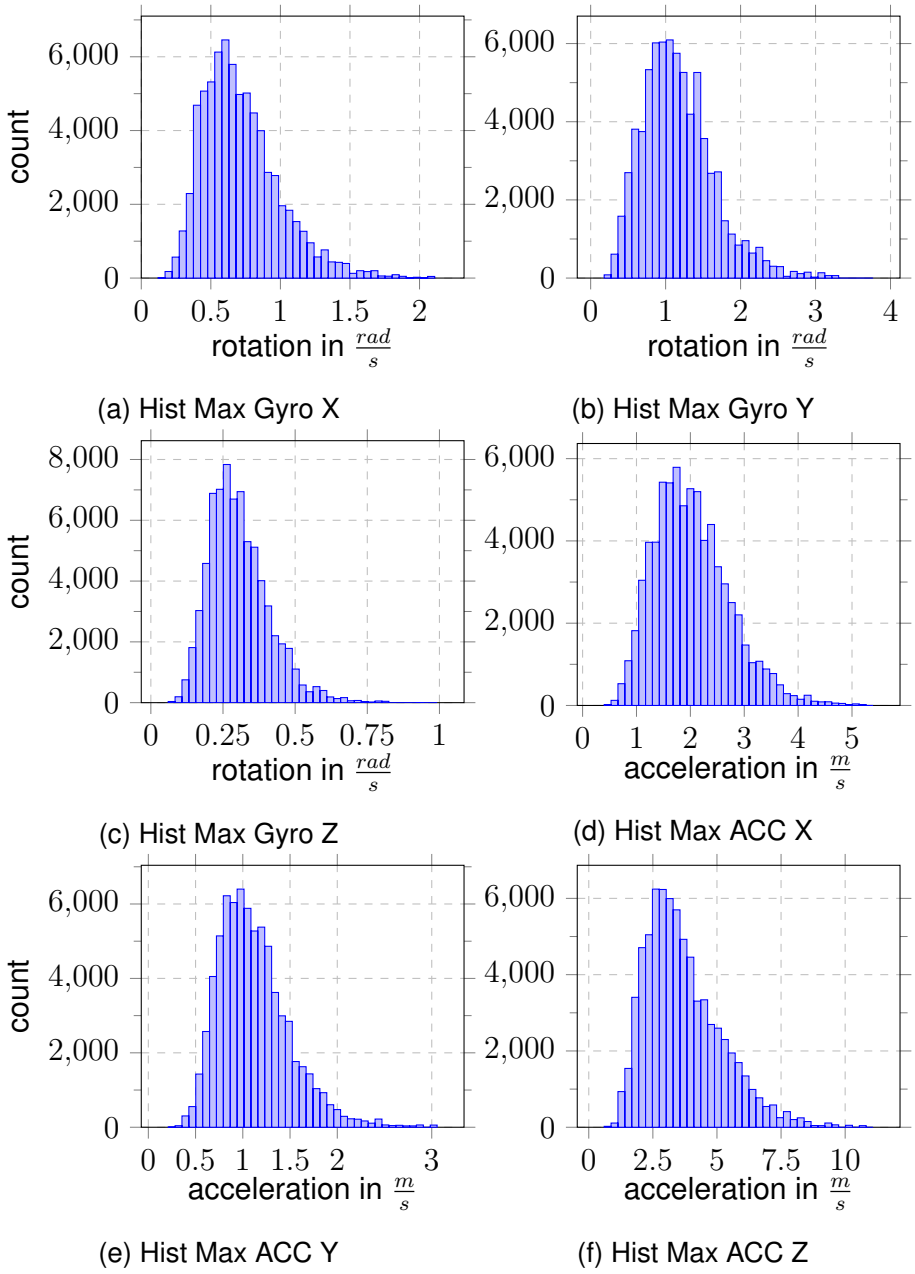


Figure A.2: Histogram of the duration of touch events in ms



(a) Hist Max Gyro X (b) Hist Max Gyro Y

(c) Hist Max Gyro Z (d) Hist Max ACC X

(e) Hist Max ACC Y (f) Hist Max ACC Z

Figure A.3: Histograms of the max. abs. values of all frames (per sensor, per axis))

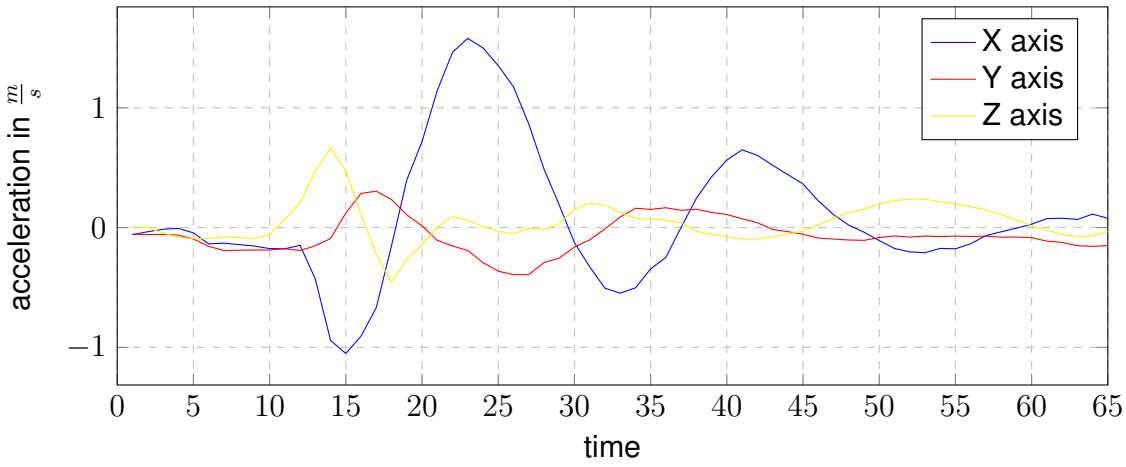


Figure A.4: Averaged motion stream of accelerometer from tapping on position A

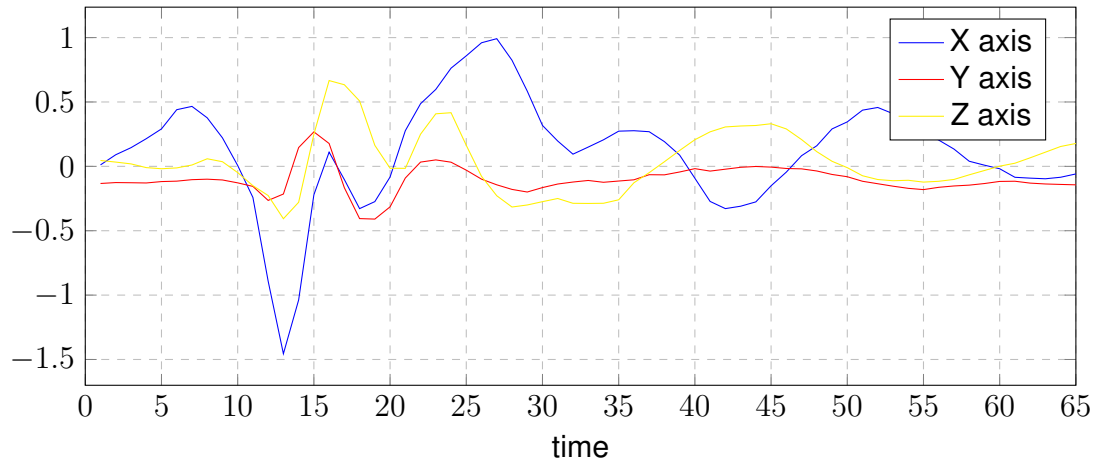


Figure A.5: Averaged motion stream of accelerometer from tapping on position B

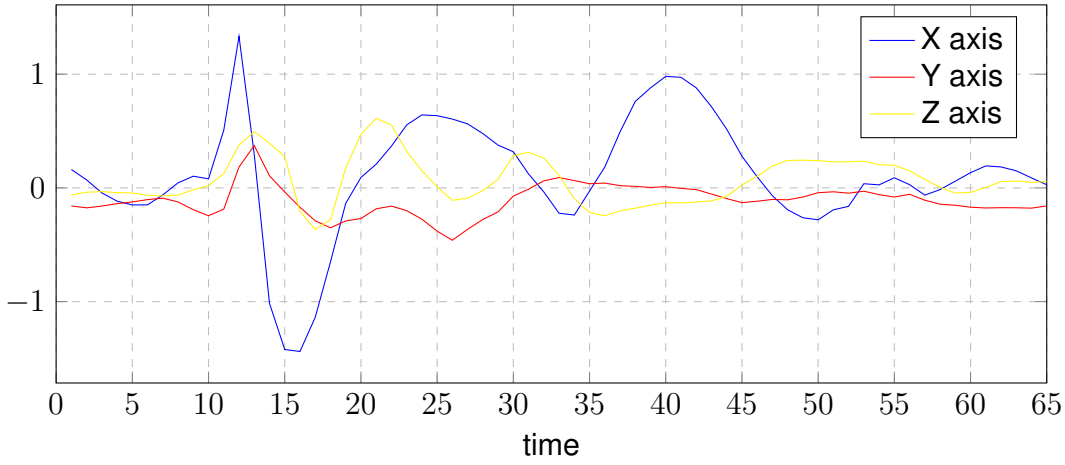


Figure A.6: Averaged motion stream of accelerometer from tapping on position C

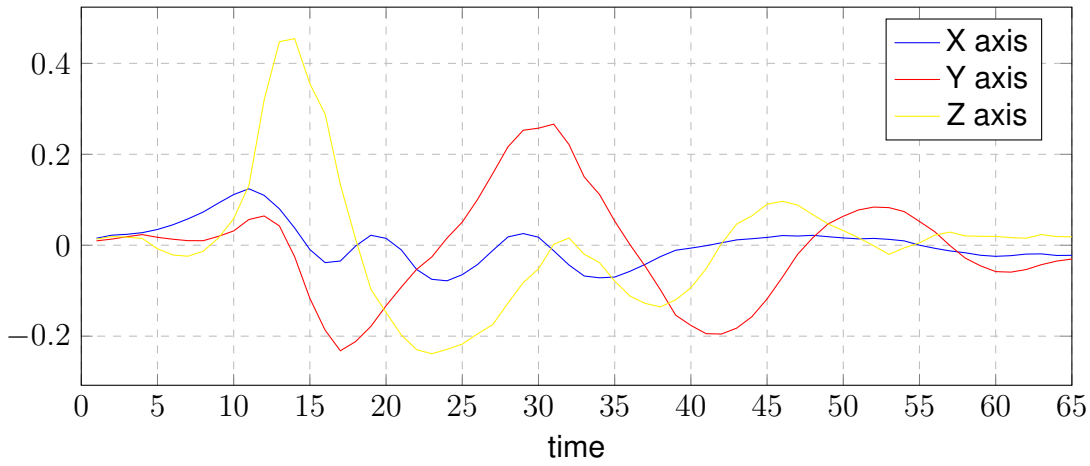


Figure A.7: Averaged motion stream of gyroscope from tapping on position C

Bibliography

[1] Liang Cai and Hao Chen. “On the Practicality of Motion Based Keystroke Inference Attack”. In: *Proceedings of the 5th International Conference on Trust and Trustworthy Computing*. TRUST’12. Vienna, Austria: Springer-Verlag, 2012, pp. 273–290. ISBN: 978-3-642-30920-5. DOI: 10.1007/978-3-642-30921-2_16. URL: http://dx.doi.org/10.1007/978-3-642-30921-2_16.

[2] Liang Cai and Hao Chen. “TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion”. In: *Proceedings of the 6th USENIX Conference on Hot Topics in Security*. HotSec’11. San Francisco, CA: USENIX Association, 2011, pp. 9–9. URL: http://dl.acm.org/citation.cfm?id=2028040.2028049.

[3] Y. Chen and Y. Xue. “A Deep Learning Approach to Human Activity Recognition Based on Single Accelerometer”. In: *2015 IEEE International Conference on Systems, Man, and Cybernetics*. 2015, pp. 1488–1492. DOI: 10.1109/SMC.2015.263.

[4] Yann LeCun and Yoshua Bengio. “The Handbook of Brain Theory and Neural Networks”. In: ed. by Michael A. Arbib. Cambridge, MA, USA: MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. ISBN: 0-262-51102-9. URL: http://dl.acm.org/citation.cfm?id=303568.303704.

[5] Emiliano Miluzzo et al. “Tapprints: Your finger taps have fingerprints”. English (US). In: *MobiSys’12 - Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys’12 - Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services. Aug. 2012, pp. 323–336. ISBN: 9781450313018. DOI: 10.1145/2307636.2307666.

[6] Emmanuel Owusu et al. “ACCessory: Password Inference Using Accelerometers on Smartphones”. In: *Proceedings of the Twelfth Workshop on Mobile Computing Systems Applications*. HotMobile ’12. San Diego, California: ACM, 2012, 9:1–9:6. ISBN: 978-1-4503-1207-3. DOI: 10.1145/2162081.2162095. URL: http://doi.acm.org/10.1145/2162081.2162095.

[7] Charissa Ann Ronao and Sung-Bae Cho. “Human activity recognition with smartphone sensors using deep learning neural networks”. In: *Expert Systems with Applications* 59 (2016), pp. 235 –244. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.04.032. URL: http://www.sciencedirect.com/science/article/pii/S0957417416302056.

[8] Chao Shen et al. “Input extraction via motion-sensor behavior analysis on smartphones”. In: *Computers & Security* 53 (2015), pp. 143 –155. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2015.06.013. URL: http://www.sciencedirect.com/science/article/pii/S0167404815000991.

[9] Zhi Xu, Kun Bai, and Sencun Zhu. “TapLogger: Inferring User Inputs on Smartphone Touchscreens Using On-board Motion Sensors”. In: *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WISEC ’12. Tucson, Arizona, USA: ACM, 2012, pp. 113–124. ISBN: 978-1-4503-1265-3. DOI: 10.1145/2185448.2185465. URL: http://doi.acm.org/10.1145/2185448.2185465.

[10] unknown. *Position Sensors*. URL: https://developer.android.com/guide/topics/sensors/sensors_position (visited on 09/24/2019).

Name: Sebastian Vendt Matrikelnummer: 752219

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Sebastian Vendt