Technische Universität Berlin

Institut fü Softwaretechnik und Theoretische Informatik
Quality and Usability Lab

Fakultaet IV
Franklinstrasse 28-29
10587 Berlin

Master Thesis

Predicting tap locations on touch screens in the field using accelerometer and gyroscope sensor readings

Emanuel Schmitt

Matriculation Number: 333772

Examined by:
Prof. Dr.-Ing. Sebastian Möller
Prof. Dr.-Ing. Axel Kuüper

Supervised by:
Dr.-Ing. Jan-Niklas Antons

Thanks to all dem brothas

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, 01.01.2050

. . . . . .

(Signature [your name])

Abstract

This template is intended to give an introduction of how to write diploma and master thesis at the chair 'Architektur der Vermittlungsknoten' of the Technische Universitï¿½t Berlin. Please don't use the term 'Technical University' in your thesis because this is a proper name.

On the one hand this PDF should give a guidance to people who will soon start to write their thesis. The overall structure is explained by examples. On the other hand this text is provided as a collection of LaTeX files that can be used as a template for a new thesis. Feel free to edit the design.

It is highly recommended to write your thesis with LaTeX. I prefer to use Miktex in combination with TeXnicCenter (both freeware) but you can use any other LaTeX software as well. For managing the references I use the open-source tool jabref. For diagrams and graphs I tend to use MS Visio with PDF plugin. Images look much better when saved as vector images. For logos and 'external' images use JPG or PNG. In your thesis you should try to explain as much as possible with the help of images.

The abstract is the most important part of your thesis. Take your time to write it as good as possible. Abstract should have no more than one page. It is normal to rewrite the abstract again and again, so probaly you won't write the final abstract before the last week of due-date. Before submitting your thesis you should give at least the abstract, the introduction and the conclusion to a native english speaker. It is likely that almost no one will read your thesis as a whole but most people will read the abstract, the introduction and the conclusion.

Start with some introductionary lines, followed by some words why your topic is relevant and why your solution is needed concluding with 'what I have done'. Don't use too many buzzwords. The abstract may also be read by people who are not familiar with your topic.

Zusammenfassung

Contents

List of Figures

List of Tables

1 Introduction

This chapter should have about 4-8 pages and at least one image, describing your topic and your concept. Usually the introduction chapter is separated into subsections like 'motivation', 'objective', 'scope' and 'outline'.

1.1 Motivation

Start describing the situation as it is today or as it has been during the last years. 'Over the last few years there has been a tendency... In recent years...'. The introduction should make people aware of the problem that you are trying to solve with your concept, respectively implementation. Don't start with 'In my thesis I will implement X'.

1.2 Objective

1.3 Outline

The 'structure' or 'outline' section gives a brief introduction into the main chapters of your work. Write 2-5 lines about each chapter. Usually diploma thesis are separated into 6-8 main chapters.

This example thesis is separated into 7 chapters.

Chapter 2 is usually termed 'Related Work', 'State of the Art' or 'Fundamentals'. Here you will describe relevant technologies and standards related to your topic. What did other scientists propose regarding your topic? This chapter makes about 20-30 percent of the complete thesis.

Chapter 4 analyzes the requirements for your component. This chapter will have 5-10 pages.

Chapter 5 is usually termed 'Concept', 'Design' or 'Model'. Here you describe your approach, give a high-level description to the architectural structure and to the single components that your solution consists of. Use structured images and UML diagrams for explanation. This chapter will have a volume of 20-30 percent of your thesis.

Chapter 6 describes the implementation part of your work. Don't explain every code detail but emphasize important aspects of your implementation. This chapter will have

a volume of 15-20 percent of your thesis.

Chapter 7 is usually termed 'Evaluation' or 'Validation'. How did you test it? In which environment? How does it scale? Measurements, tests, screenshots. This chapter will have a volume of 10-15 percent of your thesis.

Chapter 8 summarizes the thesis, describes the problems that occurred and gives an outlook about future work. Should have about 4-6 pages.

2 Related Work

The focus of this thesis lies on the practice of utilizing motion information in order to reconstruct user interactions. As this practice is a form of eavesdropping, this chapter will shed light into similar approaches of side-channel attacks that have been revealed by researchers in the past. The first section deals with different device emanations that have been utilized in various forms to obtain confidential information. The second section covers the foundations of this work as it discusses previous similar attempts predict tap locations on smartphone screens.

2.1 Eavesdropping on Emanations

Eavesdropping is defined as the the practice of secretly listening to private conversations of others without their consent [7]. As this definition originally refers to conversations between humans, eavesdropping can also be seen in terms of human-computer-interaction. In this context, the computer is seen as one conversational partner whereas the user interacting with the device is seen as the other. As the channel of communication in a human conversation is the acoustic channel, the channel in which human-computer interaction takes place has various forms. Typically, a human may enter information on a peripheral device while the computer gives feedback through an image representation. However, speech interfaces and other forms of interaction are also possible. In order to spy on the human-computer conversation, a third party with malicious intent must apply different techniques in order to spy on these interactions. A subset of these techniques which involve the utilization of device emanations will be discussed in this chapter.

In this context, a frequently discussed practice throughout academia involves the use of leaking emanations for eavesdropping purposes. These emanations can be monitored in order to carefully reconstruct the contained information. As these emanations occur in various formats, a categorization has been done based on the sensory channel they transpire. Therefore, the following sections will cover eavesdropping techniques based on acoustic, optical, electromagnetic and motion emanations.

2.1.1 Acoustic Emanations

One way of spying on electrical devices is by utilizing the acoustic channel [4, 1, 46]. Many electronic devices deploy tiny mechanics that generate sounds as a byproduct during interactions or during operation. These distinct sounds can differ in their characteristics making them adequate to identify the original information currently being

processed by the machine. In these scenarios an eavesdropper targets a microphone in near proximity of the target device to capture the audio signals the device is exposing. A learning algorithm is then applied to the audio signals to reconstruct information.

In 2010, Backes et al. examined the problem of acoustic emanations of dot matrix printers, which where, at that time, still commonly used in banks and medical offices. By using a simple consumer-grade microphone, the researchers were able to recover whole sentences the printer was printing based. Backes et al. processed the audio samples in order to extracted frequency-domain features. These features worked as input for a hidden markov model, a technology commonly used in audio speech recognition. As a result, the recognition system was able to reconstruct individual characters based on the sound inputs. To demonstrate a potential attack, the researchers deployed the system in a medical office being able to obtain up to 72% of the sentences being printed on medical subscriptions [4].

Being inspired by the findings concerning the dot matrix printer, Asonov and Agrawal investigated acoustic emanations produced by hitting keystrokes on a desktop and a notebook keyboard. Following their hypothesis claiming that each keystroke has a macroscopic difference in it's construction mechanics, as well as a distinct reverberation caused by the position in the board, individual keystrokes were recorded [1]. Researchers then extracted frequency domain features from the audio signals and passed them into a neural network. In an experiment performing 300 keystrokes, 79% of the characters could be correctly recognized [1]. As this technique required substantial training before recognition, other studies have reached similar accuracies using an unsupervised approach [46] on the one hand and by using acoustic dictionaries [6] on the other.

2.1.2 Optical Emanation

Besides acoustic emanations, optical emanations can also pose a valuable source of information for a potential eavesdropper. Most electronic devices, such as notebook, smartphones and tablet computers, provide graphical user interfaces through their own built-in screens. Even though these screens are meant to target the human eye, they can reflect off other surfaces. The reflections can be caught by high resolution camera sensors, which can then display the image revealing secret information.

One example of the use of optical emanations has been developed by Kuhn aiming to eavesdrop on cathode-ray-tube(CRT) monitors at distance. The researcher has shown that the information displayed on the monitor can be reconstructed from its distorted or even diffusely reflected light. In an experiment, Kuhn targeted a screen displaying an image against a wall while the reflections of the screen were captured using a photomultiplier. The experiment showed that enough high-frequency content remained in the emitted light for a computer to reconstruct the original image [21]. A similar approach that comprises reflections has been shown by Backes et al., however focusing on LCD displays. In this experiment, the researchers caught reflections in various objects that are

commonly to be found in close proximity to a computer screen. Such objects included eyeglasses, tea pots, spoons and even plastic bottles. This work was later extended to additionally capture screens based on the reflections on the human eye's cornea [3].

2.1.3 Electro-magnetic Emanation

As electric currents flow through computer components, they emit electromagnetic waves to their near surrounding. These electromagnetic radiations can be picked up as a side channel using sensitive equipment in order to retrieve data. Electromagnetic emanations have been a research topic that has been present for several decades while the first research conducted reaches far in time.

Back in 1943, a research group under the codename TEMPEST, a subdivision of the NSA[1], were able to infer information from the infamous Bell Telephone model 131-B2, a teletype terminal which was used for encrypting wartime communication [36]. Using an oscilloscope, researchers could capture leaking electromagnetic signals from the device and by carefully examining the peaks of the recorded signals, the plain message the device was currently processing could be reconstructed [36]. This technique was later advanced and used in the the Vietnam war. Through similar electric emanation the US military could detect approaching Viet Cong trucks giving them an immense competitive advantage [33]. Today, TEMPEST is a security standard for electronic devices ensuring that certified devices do not accidently emanate confidential information [34].

A second prominent finding of eavesdropping on electromagnetic emanations was done by the researcher van Eck, who discovered that cathode-ray-tube monitors could be spied upon from a distance [44]. By using general market equipment, such as antennas van Eck and standard receivers, signals emitted from the cable connecting the computer to the monitor could be received. Since these cables only transmit the video signal for visualization, the researcher could display the visual output of the target monitor revealing a full screen cast of the original image. This attack is referred to in literature as *Van Eck Phreaking* [16, 20].

Furthermore, research has shown that side-band electromagnetic emanations are present in keyboards [45], computer screens [44, 22], printers [39], computer interfaces, such as USB 2 [35] and the parallel port [43] and in Smart Cardes[40].

2.1.4 Motion Emanation

In the past decade modern devices are increasingly equipeed with highly responsive sensors, such as the gyroscope and accelerometer enabling the devices to sense rich interactions with their environment. As user interactions, such as typing the keyboard

[1] National Security Agency

or tapping on touchscreens, require the user to apply a certain force while entering information, this motion can be captured by motion sensors in order to be used for a side-channel attack [30, 37, 9].

Marquardt et al. conducted an experiment where an Apple iPhone that captures accelerometer motion was placed next to a dekstop keyboard. Subjects then had to enter sentences while the application was monitoring the user's motion. The researchers could decode the accelerometer signals by mapping the certain vibration caused the typing motion to their keystrokes. The decoded characters were then matched based on a dictionary containing a frequency distribution of commonly used words. As a result, words could be successfully obtained with an accuracy of up to 80% [28].

As motion emanations are highly relevant for the work in this thesis, the next section will be dedicated to further work regarding this topic.

2.2 Eavesdropping on touch screen user interactions

As we have seen in the previous section, user interactions with peripheral devices, such as the keyboard or PIN pads, can be obtained by either the acoustic channel, by electromagnetical leakage or by capturing the motion of the user. However, with the rise in soft keyboard usage, the same discussed methods that extract information from keyboard do not apply to tap interactions on a touchscreen surface. Since a touchscreen does not embody fine mechanics producing sounds nor does it have emanating cables, the research community has developed nouvelle methods to spy on user inputs based on the smartphone embodied motion sensors.

The general idea behind the three approaches that are going to be discuss in the following is that a tap, or to be more precise the magnitude of the force of a tap, on a specific touch screen location creates an identifiable pattern on the motion sensors that can be sufficient to infer the initial tap location. This is particularly interesting since motion sensors are not considered as being privacy-sensitive and therefore lack access restrictions by the operating systems of the devices.

Touchlogger

The first paper regarding this security threat was published by Cai and Chen. In their proof-of-concept study they created an Android[2] application which displays a 10-digit PIN-pad. During interactions with the PIN-pad, the accelerometer signals were monitored and used for later data analysis. Having observed that a tap movement affects the rotation angle of the screen, the researchers handcrafted features based on the path

[2]Android operating system for smartphones by Google Inc.

of the *pitch*[3] and the *roll*[4] angles of the accelerometer. These were intersected to find a dominating edge on where the tap had presumably taken place. By using a probability density function for a Gaussian distribution the researchers were able to achieve an average accuracy of 70% for interred PIN-pad digits. The training set size involved 449 pin strokes [9]. Even though *Touchlogger* was a promising first step, due to it's low granularity of only 10 distinguishable large screen areas, it remained unclear if the attack can be carried over to a full software keyboard. Furthermore, since the inference was performed on only a single smartphone model, the question is left open whether other smartphones or tablet computers are similarly vulnerable.

ACCessory

In order to show the feasibility for a full software keyboard, Owusu et al. performed a second attempt to the problem by creating *ACCessory*. *ACCessory* is an Android application with functionalities similar to the previously mentioned *Touchlogger*. However, the application significantly differ in it's tap area granularity providing two separate modes for tap inputs: *area mode* and *character mode*. *area mode* consists of tap areas arranged in a 60-cell grid, whereas a QWERTY keyboard within landscape orientation was displayed in *character mode*. Having extracted features mainly from the time-domain, a classification using the Random Forests algorithm reached an accuracy of 24.5% for the 60-cell grid. Here, the corresponding dataset consisted of 1300 keystrokes collected from 4 participants. As the *area mode* experiment focused on recognizing individual keystrokes, the *character mode* experiment focused on cracking passwords. By combining the keystrokes into a sequence and assuming recognition errors in individual characters, the researchers could create a ranked list of candidate passwords by running a maximum likelihood search for the most probable classification errors for an obtained password. Here, 6 out of 99 password could be inferred under 4.5 median trials given that one trails refers to traversing down one item of the candidate list. Furthermore, the majority of 59 out of 99 passwords could be inferred within 2^{15} median trials. As general result, even though the overall accuracy of the learning system scored low, the researchers could significantly reduce the search space for reconstructing a password indicating that accelerometer readings can in deed yield confidential information.

TapPrints

The most comprehensive study to date regarding the topic was conducted by Miluzzo et al. and differs from *ACCessory* and *Touchlogger* in many important ways. While both previous studies are both evaluated on the Android smartphones, *TapPrints* investigates the tap inference on both iOS and Android operating systems including tablets and smartphones alike. Another important point of differentiation is the used learning system. In order to raise the level of entropy, *TapPrints* combines readings from the

[3]The pitch-angle corresponds to the x-axis of the accelerometer.
[4]The roll angle corresponds to the y-axis of the accelerometer.

accelerometer and gyroscope for a more sophisticated feature extraction. Here, time-domain and frequency-domain features, as well as the correlation and angles between individual sensor components are considered. For classification purposes, the researchers use an ensemble method combining decision tress, support vector machines, k-nearest neighbors and multinomial logistic regression in a winner-takes-it-all[5] voting fashion. The dataset collected in this experiments contains over 40.000 individual taps collected from 10 different users. In addition, the researchers also requested user to use different input modalities while typing including the usage of the index finger and thumb.

The *TapPrints* undertaking consists of two separate experiments: The first is a icon tapping experiment where icons are arranged in a 20 cell grid and the second one being a letter tapping experiment involving the standard software keyboard offered by the operating system. In the first experiment, an average accuracy of 78% was achieved for the iPhone whereas 67% of icon taps could be correctly inferred on the Android device. In the letter tapping experiment users were asked to enter pangram[6] sentences on the OS soft-keyboard. Results showed that on both iPhone and Android an average of 43% of the letters could be correctly classified. Even though the average accuracy for individual letters were not particularly high, Miluzzo et al. could show that when pangrams were repeatedly entered, a majority vote could be applies to individual character recognitions allowing to recover the whole pangram in approx. 15 trials. To conclude, *TapPrints* could demonstrate that motion sensor can be used to obtain passwords on multiple platforms and formats and with different input modalities.

Comparison to similar studies

Since the data used in *TapPrints* and in the other related work was collected in a controlled environment [37, 9, 30], it is not possible to tell if the feasibility of tap inference will also apply to data collected in a field environment. As this has not been investigated, this will form the central research question in this thesis. Additional questions will be discussed in the hypothesis section.

To draw the border between this study and the previous mentioned ones, this study will differ or relate as follows:

- **User interface**: For data acquisition, the user interface will cover tap area grids, as we have seen in *ACCessory* and *Touchlogger*, with 4, 12 and 20 distinguishable classes.

- **Devices**: Unlike *Touchlogger* and *ACCessory*, the devices used for this study are on the iOS Platform. Apple iPhone 6, 6s and 7 will be considered due to their mutual screen size.

[5] This implies that all classifiers classify separately and the classifier with the highest prediction score wins.

[6] A pangram is a sentence using every letter of a given alphabet at least once.

- **Motion sensors**: *TapPrints* has shown that the gyroscope yields more information than the accelerometer [30], therefore both sensors will be monitored. Furthermore, sensors will be read at a frequency of 100Hz, since this had been proven to deliver best results [30, 37].
- **Dataset**: In comparison to related studies, a dataset containing data points collected in a laboratory environment and the field environment will be acquired from a total of 27 users. This data will contain the index finder and thumb as input modalities and standing and sitting as body postures while input.
- **Feature extraction**: *TapPrints* shows a deliberate list of features [30] that will be partially adopted. The features extracted will be discussed in section TODO.
- **Classification**: A feed-forward neural network, as well as a support vector machine with radial kernel will be used for classification. More will be covered in the classifcation section.

3 Machine Learning Fundamentals

As machine learning techniques will be used for the classification of individual tap locations on a smartphone touchscreen, the following chapter will give a brief overview of the fundamental concepts evolving around statistical learning. Individual categories of learning algorithms will be discussed followed by two supervised-learning algorithms, namly Support Vector Machines and Neural Networks.

3.1 Overview and Definition

Ever since computers were invented, there has been a desire to enable them to learn [42]. This desire has grown into the field of machine learning which seeks to answer questions on how to build build systems that automatically improve with experience, and what the fundamental laws of learning processes are. Today, state-of-the-art ML covers a large set of methods and algorithms designed to accomplish tasks where conventional hard-coded routines have brought insufficient results. From speech recognition to email spam detection or recommendation systems, ML methods find broad usage in a variety of problem domains.

In order to understand what the principle of machine learning is, we will start with a definition by Samuel [42]:

> *Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed.*

In this definition, special emphasis is to be put on the last part of this definition. A computer is only then able to learn when he can perform a task without being explicitly instructed. Thus, in order to learn, the computer must somehow be able to instruct itself without the influence of an outer . As this definition lacks a more detailed view on what computer learning is, we will dive into a definition by Tom Mitchell [31]:

> *A learning system is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

The example that Mitchell notes, is one from the games of checkers [31]. In this case checkers is the task T that the computer is aiming to learn. In order for the computer to learn, information on previously played matches is required. Since the computer does not know how to evaluate is a particular match was either good or bad, we set the performance to be defined based on how many matches were actually won. If a computer program can raise the amount of games won *(performance measure P)* with the help of the experience from previously placed matches *(experience E)* then it can learn to play checkers *(task T)*.

To break this down into a more practical perspective, the challenge lies in finding an appropriate model in order to learn from data, which is the most common format to represent past experience. By learning, the computer adjusts parameters on the model based on the data that we feed the system with. Once the model has been adjusted, it can perform tasks with new incoming experience.

3.2 Categorization of Methods

As machine learning algorithms and methods differ from their approach to learning and underlying concepts, it is common practice to separate these into the following categories [11, 29] : Supervised learning, unsupervised learning, reinforcement learning and evolutionary learning. In the following sections I will briefly outline these.

Supervised learning, which is also named learning from example, is presumably the most prominent category of ML algorithms. The algorithm is given a training set of examples $\{x_0, \ldots, x_n\}$, which are also known as *features* and the correct target values $\{y_0, \ldots, y_n\}$ mapped to each set of features, which is the answer that the algorithm should produce. The algorithm then generalizes based on the training set in order to respond with sensible outputs on all possible input values. Outputs, if they are discrete labels, correspond to a classification task whereas outputs on a continuous scale refer to a regression task (see [29]).

An example for supervised learning is the classification of malignant or benign tumors as seen in cancer diagnosis. Let's assume we have a dataset with different properties of a tumor, such as the size or the color of the cells. These properties form our features x. Each set of features is mapped to an output label y stating if the tumor is malignant or benign. The first step is to use the pairs (x, y) of the training set to teach the algorithm the correct mapping of the problem space. As x is linked to the output y in the training set, learning the conjunction of these two values is done under supervision since the output label y is given. Once learned, the algorithm is generalized to map unseen inputs to the correct output label.

Practical applications are for example digit and handwriting recognition [23], spam filtering [14] for e-mails or network anomaly detection [25].

Presumably the most widely known machine learning techniques belong to this category, such as Support Vector Machines (SVMs), Artificial Neural Networks, Bayesian Statistics, Random Forests and Decision Trees [11].

Unsupervised learning is the task of learning structures from input values that are not explicitly labeled. In comparison to supervised learning, where correct output values are provided for each input, unsupervised algorithms learn to identify similarities in the input data and can therefore group these [11]. These grouping problems are referred to as *clustering*. The underlying idea here, is that humans learn by not explicitly being told what the right answer should be [29]. If a human sees different species of snakes, for instance, he or she is able to identify them all as snakes. Hence, the human is aware that there are differences in each specific type of snake without specifically knowing a correct label.

A prominent example where unsupervised learning is heavily used, is in recommender systems for online retail shops. Amazon.com, for instance, uses a technique called *collaborative filtering*, which measures similarity in customers based that they have previously bought [26]. Having identified similar customers utilizing the cosine similarity, the algorithm can then recommend items that similar users have bought. This technique is also used for music recommendations [38] or social network recommendations [19].

The field of unsupervised learning is closely related to density estimation in statistics, as with the density of inputs, we are able to group them. The K-means algorithm is the most prominent in this field [29].

Reinforcement learning falls in between supervised and unsupervised learning methods. Whereas supervised learning tries to bridge the gap between input and corresponding output values and unsupervised methods detect groupings in incoming data, reinforcement learning is based on learning with a *critic* [29]. The algorithm tries different solution strategies to a problem and is told weather or not the answer provided was correct. An important fact here, is that the algorithm is not told how to correct itself. This practice of "trying-out" is based on the concept of *trail-and-error learning* which is known as the *Law-of-effect* [29]. A good example is a child that tries to stand up and learn walking. The child tries out many different strategies for staying upright and receives feedback from the field based on how long it can stand without falling down again. The method that previously worked best is then repeated in order to find the optimal solution resulting in the child learning to walk [29].

In more mathematical terms, the reinforcement learning problem is formalized with an agent and his environment. The environment in which the agent is set provides a set of *states* on which the agent can perform *actions* to maximize a certain *reward*. By performing actions the state changes and a new reward is

calculated. The reward then tells the agent if the action was a good choice. Goal of the algorithm is to maximize the reward [29].

Reinforcement learning is a practical computational tool for constructing autonomous systems that improve themselves with experience. These applications have ranged from robotics, to industrial manufacturing, to combinatorial search problems such as computer game playing [18]. Prominent methods of this category are Q-learning, Monte Carlo methods and Hidden Markov Models [29].

Evolutionary learning is inspired by strongly inspired by nature. As biological evolution improves the survival of a species, the strategy of adaptation to improve survival rates and the chance of offspring has inspired researchers to craft genetic algorithms (GA) [29].

Genetic algorithms are a family of adaptive search procedures which have derived their name from the fact that they are based on models of genetic change in a population of individuals. These models have their foundation in three basic ideas: (1) Each evolutionary state of a population can be evaluated on a *fitness* scale. This is done since biological evolution has a natural bias towards animals that are f̈itter ẗhan others. These animals tend to live longer, are more attractive and generate healthier and happier offspring, an idea which was originated in Charles Darwin's T̈he Origin of Species.̈ (2) Each population can be mated to generate offspring using a *mating operator*. (3) The third component are *genetic operator*, such as *crossover* and *mutation*, which determine how the offspring solution is composed of the genetic material of the parents [10].

Evolutionary learning is often considered when other methods fail to find a reasonable answer. Algorithms find applications in search and mathematical optimization, but also in arts and simulation [29].

In this section we have seen several different problems that we can solved with the help of algorithmic learning. In this thesis, the focus is put on supervised learning as mapping the sensory data to the locations of tap event falls into this category. For predictions, two learning algorithms will be used which will be introduced in the following.

3.3 Support Vector Machines

A SVM is a non-linear kernel based extension of the so-called maximum margin classifier. Originating from binary classification problems, were $y \in \{1, -1\}$, the general idea of a maximum margin classifier is to find a separating hyperplane in the p-dimensional feature space.

This hyperplane separates the training examples leading to a maximum distance between the observations of the two classes [17]. This distance is referred to as margin M mea-

suring the smallest distance of a training observation towards the defined hyperplane. In order to find a suitable hyperplane only a subset of the training examples are sufficient.

Despite all that, linear separability is unlikely to be applicable to most real world problems. However, when a problem is linear separable, the constraint to the support vectors induce an undesirable sensitivity to individual observations, which can lead to high variance of the classifier. For this reason, the maximum margin classifier can be extended to it's more robust successor, the support vector classifier. Still being a linear classifier, the SVM allows for violations in fitting the margin.

Mathematically, the support vector classifier can be described via following optimization problem [17]:

$$\max_{\beta,\epsilon} M \tag{3.1}$$

$$\text{subject to} \sum_{s=1}^{p} \beta_s^2 = 1 \tag{3.2}$$

$$g(x_i) = y_i(\beta_0, \beta_1 x_1, \ldots \beta_p x_p) \geq M(1-\epsilon) \tag{3.3}$$

$$\epsilon \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \tag{3.4}$$

The objective of this optimization problem is to maximize the margin M while choosing appropriate vector parameters β and ϵ. In this context, the parameter vector β contains the coefficients of the hyperplane and the vector ϵ includes so-called slack variables that account for instances which are located on the wrong side of the margin and the hyperplane. These can be expressed as follows assuming that M is positive [17]:

$$\epsilon_i = \begin{cases} 0 & g(x_i) \geq M \\ > 0 & M < g(x_i) < 0 \\ > 1 & g(x_i) > 0 \end{cases} \tag{3.5}$$

The hyperparameter C is a non-negative variable, which can be seen as a budget variable that allows for a certain sum of ϵ_i observations to be on the wrong side of the margin or hyperplane, respectively [17]. C manages the bias-variance tradeoff, since a low C tries to find a maximum margin hyperplane that almost exactly separates the two classes, resulting in low bias classifier for the available data set but in a high variance classifier for test data. Concurrently, allowing high budget C results in a high bias classifier that widens the margin, introducing more violations ϵ_i while reducing the variance of the classifier. Thereby C also controls the number of considered support vectors in dependence of the margin width.

Extending the support vector classifier to non-linear decision boundaries brings us to the SVM. Instead of extending the predictor space using higher order polynomials and interactions, SVM uses the so called "kernel trick" [12]. In order to apply the "kernel trick" , i.a. Efron and Hastie show that the optimization problem above can be rewritten as

$$\hat{f}(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle \tag{3.6}$$

where $i \in S$ defines the subset of support vectors and $\langle x, x_i \rangle$ is the dot product of all pairs in the support vector. Thus, the parameters β_0 and $\sum_{i \in S}$ can be estimated with the help of least squares via simply computing the inner products of each pair in the support vector [12]. The Expression in $\langle x, x_i \rangle$ can be generalized by a kernel function

$$K(x_i, x_{i'}) = \sum_{s=1}^{p} x_{is} x_{i's} \tag{3.7}$$

indicating the linear kernel that quantifies the distance between each pair in the data set [17]. Accordingly, the equation above can be rewritten as

$$\hat{f}(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i) \tag{3.8}$$

but in spite of restricting $K(\cdot, \cdot)$ to (3.7), we can now choose an arbitrary kernel function that maps our data into a high dimensional space where it is linearly separable. With this "kernel trick" the SVM can work in an implicitly enlarged predictor space, just by computing $\binom{n}{2}$ kernel functions $K(\cdot, \cdot)$, as opposed to an explicitly augmented predictor space, which is in fact computationally intractable [17]. In this work, we will be using a radial kernel function:

$$K(x_i, x_{i'}) = \exp(-\gamma(\sum_{s=1}^{p} x_{is} x_{i's})^2), \tag{3.9}$$

where γ is a tuning parameter. After the parameters are learned on the basis of the training set, a new observation with the feature vector x_0 is classified via the following decision rule

$$\hat{f}(x) = sign(\beta_0 + \sum_{i \in S} \alpha_i K(x_i, x_{i'})). \tag{3.10}$$

In view of the task at hand, an extension to multi-class classification of the SVM is utilized via one-versus-one classification. Given C classes, $\binom{C}{2}$ binary classifiers are learned in such a fashion that the C-th class is coded as +1 and the C0-th class as 1. Consequently, a new instance is assigned to the class to which it was classified most frequently.

3.4 Neural Networks

In this section, I will introduce neural networks which are interconnected systems used for supervised learning tasks. Besides the SVM, neural networks will be used in this thesis to predict the tap locations based on the collected motion data. Artificial neural networks are computing systems inspired by the biological neural networks found in animal brains. As these systems consist of several components, the first section will cover the neural units forming the fundamental processing unit of a network. After this, individual activation functions of these networks are discussed. In the last section the backpropagation algorithm which is used to train ANNs is explained.

3.4.1 Neurons

A neuron is fundamental processing unit of an artificial neural network. The diagram shows a model [15] of a neuron which consists of following components:

- A set of *connecting links* or *synapses* which have a certain weight defined as the vector $\vec{w}$. The signal, represented as $\vec{x}$, flows through the *synapse* and is mutipied by it's weight w_i.
- A *adder* for summing the input signals and weights of the incoming synapses. These operations constitute a linear combiner.
- An *activation function* φ for limiting the amplitude of the output signal. This function is often referred to as the *squashing function* since it squashing the possible output range [15].
- An externally applied *bias* b which has the ability to lower or rise the net input to the activation function depending if the bias is negative or positive.

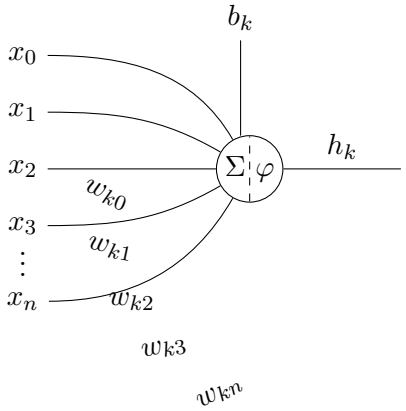

Figure 3.1: The diagram shows an artificial neuron's model with it's individual components.

Mathematically, a neuron k can be expressed with the following equation [15]

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{3.11}$$

$$h_k = \varphi(u_k + b_k) \tag{3.12}$$

where $x_1, \ldots, x_n$ are the input signals; $w_{k1}, \ldots, w_{kn}$ refer to the synaptic weights of the neuron; u_k is the linear combiner output of the summation on which the bias b is added. The output of the neuron is expressed as h_k.

3.4.2 Activation Functions

The activation function $\varphi(u_k)$ denotes the output of the neuron k and forms the junction between the neuron's input x_k and output h_k. In order for the network to learn any complex non-linear function, each neuron in the networks requires a non-linear activation function [13]. In this context, one commonly used function is the s-shaped *sigmoid function* of which the *logistic function* [15] is an example:

$$\varphi(v) = \frac{1}{1 + \exp(-v)} \tag{3.13}$$

The logic function, as in 3.2, is well suited for classification as it is a non-linear function and it transforms the output values to either side of the curve. This results in a clear distinction between classes. However, the function has a compact domain range, meaning that the logistic function squashes output values into ranges $(0, 1)$. Consequently, when the inputs of a neuron become large, the function saturates at 0 or 1 with a derivative in these points being close to 0. As the derivatives are used for training the network[1], the network trains slower if the weights or biases are high. This is referred to in literature as the *vanishing gradient problem* [32].

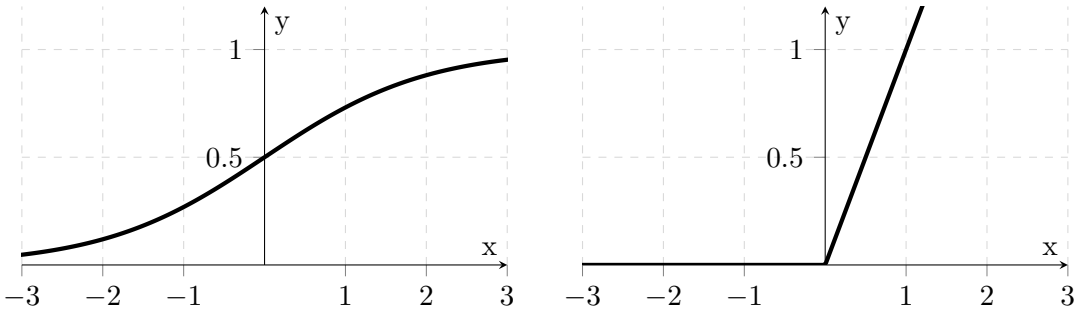

Figure 3.2: The figure shows a plot of the hyperbolic tangent activation function on the left and a plot of the ReLU activation function on the right.

[1]The training is performed via the backpropagation algorithm which is explained in section 3.4.4

In order to avoid saturation problems, a commonly used non-linear activation function is the Rectified Linear Unit (ReLU) function [32], which can be seen in figure 3.2:

$$\varphi(v) = \max(0, v) \tag{3.14}$$

ReLUs are non-saturating which results in a neuron always learning if the input is positive. Networks with ReLUs train several times faster and have become, as of 2015, the standard activation function for deep neural networks [24].

3.4.3 Feedforward neural networks

A feedforward neural network, or multilayer perceptron (MLP), is an ANN which consists of layers L of neural units. The goal of a feedforward network is to approximate some function f^* [13]. In terms of a classifier, $y = f^*(x)$ maps an input x to a label y. Consequently, a neural network with m input nodes and 1 output node serves as a function with m inputs and 1 output.

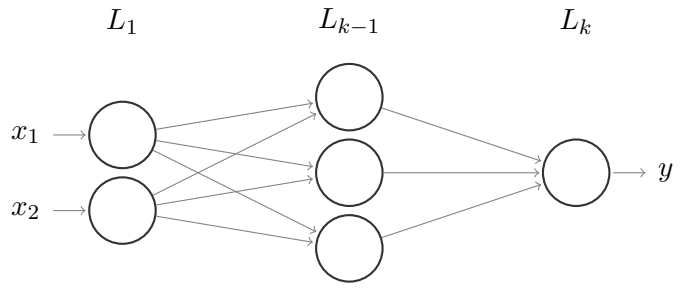

Figure 3.3: The diagram shows the structure of a typical feedforward network. The network has two input units in the input layer L_1 and one output unit in the output layer L_k. Layers $L_2 \dots L_{k-1}$ the the so-called hidden layers.

The network is named feedforward as it the information flowing through the network passes with the outermost input layer and and ends at the output unit of the output layer [13]. All units in a layer are fully connected to the succeeding layer, however, there is no interconnection between units in the same layer. Figure 3.3 shows a typical feedforward architecture with three layers including a single hidden layer L_{k-1}.

Considering equation (3.11) from the neuron section, we can now compute the input to the output node, given by

$$\sum_{k=1}^{n} \alpha_k h_k \tag{3.15}$$

where h_k is the output of the kth hidden node and α_k being the weight from the kth

hidden to the output node. The output unit's activation function is then applied to this value, transforming the output to the given equation

$$y = \varphi\left(\sum_{k=1}^{n} \alpha_k \varphi\left(\sum_{i=1}^{m} w_{ik} x_i + b\right)\right). \tag{3.16}$$

.

3.4.4 Backpropagation

The purpose of the backpropagation algorithm [41] is to adjust the synaptic weights of the network to approximate the function mapping between the inputs x of the training data with the corresponding output labels y. The algorithm describes the process of training. The result of this algorithm is a neural network configured to minimize the error when solving a supervised learning task.

As a first step, the weights of the network are initialized which is usally done in a randomly manner or based on a certain heuristic. Then, each input pattern p, with features $p = \{x_0, x_1, x_2, \ldots, x_n\}$ and label y, is then sequentially processed, layer by layer, by the network in two phases. In the first phase, the *forward phase*, the ouput of the network is computed. The error function for the output nodes j are then calculated as follows, where $\hat{y}_j$ denotes the output node's generated output and y_j is the desired ouput:

$$E = \frac{1}{2} \sum_{j} (\hat{y}_j - y_j)^2. \tag{3.17}$$

Continuing in the *backward phase*, the network measures how much each neuron in the output layer L_k has contributed to each output neuron's error. Furthermore, as to measure the error contributions coming from each neuron in the previous layer, this step is repeated until the input layer is reached and all error contributions, the gradients, are computed. To put this in other words, the *backwards phase* measures the error gradients across all connection weights in the network by propagating the error gradients back into the network.

In this iterative process, the error gradients of the error function are calculated based on the partial derivative with respect to each connecting weight. If we define o_j as output of a neural unit with

$$o_j = \varphi(net_j) \tag{3.18}$$

and the units input as

$$net_j = \sum_{j=1}^{n} x_i w_{ij}, \tag{3.19}$$

the chain rule can be applied in order to compute the partial derivatives as follows:

$$\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta o_j} \frac{\delta o_j}{\delta net_j} \frac{\delta net_j}{\delta w_{ij}} \tag{3.20}$$

Given the partial derivative in respect to the weight w_{ij}, the change of the weight Δw_{ij} can be determined. Here, the weight update equation (3.22) is computed depending on two cases. Either if the node j is in the hidden layer or in the output layer:

$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}} = -\eta \delta_j o_i \tag{3.21}$$

$$\delta_j = \begin{cases} \varphi'(net_j)(o_j - \hat{y}_j) & \text{node } j \in L_k \\ \varphi'(net_j) \sum_k \delta_k w_{jk} & \text{node } j \notin L_k \end{cases} \tag{3.22}$$

In equation (3.23), η denotes the *learning rate*, which defines to which amount the reverse gradients are applied to the weight update; k refers to a node in the successor layer of the node j.

Once the weights are updated, the global error of the network is calculated and the forward and the backward phase are repeated for the rest of the training patterns. One pass through all of the training patterns is called a *training epoch*. Several strategies exist to stopping the described training process. One condition is to stop after to a fixed number of training epoches, a second can be a stop once the change in the weights reaches a certain low-end threshhold. After the process, the final values of the weights are saved and can be used for prediting new incoming patterns.

4 Data Acquisition System

In this chapter, I would like to introduce TapSensing. TapSensing is a labeled data acquisition system designed to collect user generated taps with corresponding sensor readings. The system comprises of an iOS Swift and a Python Django server-side application. The chapter begins with a broader view on the system by explaining the overall system architecture and will then dive into it's individual components and implementation.

4.1 Overall System Architecture

The TapSensing application consists of two main components: the mobile and the server-side application. In brief, the mobile client provides the ability for a user to generate tap information including the motion sensor recordings. For the data to be stored in a centralized manner, the server-side application provides HTTP endpoints as a gateway to the database. The architecture, as illustrated in figure 4.1, consists of various components which are outlined in the following.

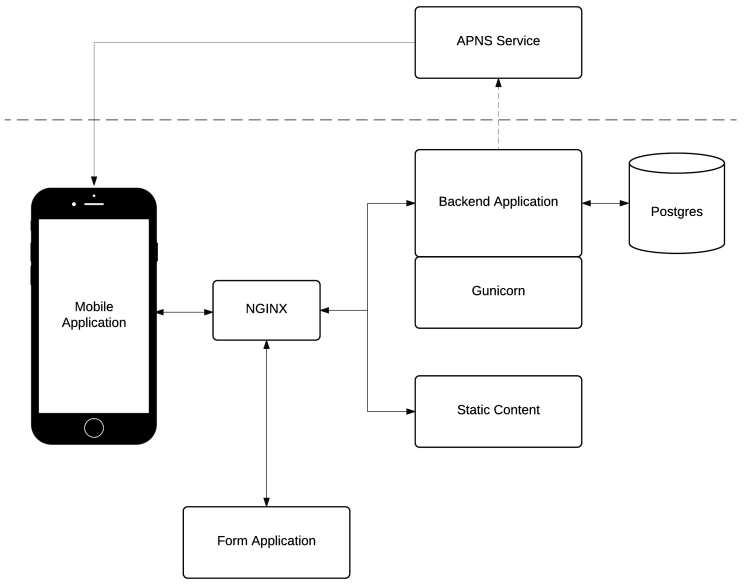

Figure 4.1: The diagram shows the overall architecture of TapSensing.

- **Mobile application**: The iOS application provides interfaces for the user to generate taps with corresponding sensor information. Furthermore, the application is capable of sending the acquired data to the backend application. More features of the mobile application are presented in section 4.2.
- **NGINX**: For accepting and routing incoming HTTP requests, an NGINX reverse proxy/load balancer is used. The reverse proxy forwards the incoming requests to the server-side application and is capable of serving static content, such as images, HTML, CSS and JavaScript files of the form application.
- **Gunicorn**: Gunicorn is a Python Web Server Gateway Interface (WSGI) HTTP server which runs the source code of the backend application.
- **Backend Application**: The backend application provides HTTP API endpoints which inherit the server-side logic. The backend provides authentication and persistence functionalities. More information on the backend application is to be found in section 4.3
- **PostgreSQL**: TapSensing uses a PostgreSQL[1] database for storing the application state, user related information, the survey data and the retrieved tap and sensor information.
- **Form application**: The form application provides a web user interface where study participants can answer survey questions.

4.2 Mobile Application

4.2.1 User Interface

Login

When the application is opened for the first time, a login screen appears. As in standard login screens, the interface asks for credentials including username and password. Authenticating users, has the advantage, that the generated data can be mapped to individual users.

Start Screen

During the trial the user is asked to generate taps once a day. In order to indicate if the user is eligible to perform a tap generation trial, the start screen shows a button that is either active or inactive. This switch depends on 4 distinct conditions:

When data is collected in the laboratory environment, the app is set to *lab mode*. In *lab mode* the button is always active and trials can be performed. When a user has not performed a trial today, the button is inactive. In contrast, when a user has performed

[1] PostgreSQL is a general purpose and object-relational database management system.

Figure 4.2: This figure shows the start screen in different configurations. On the left hand-side screenshot, the user has not performed a trial while the the middle image shows the screen where a trial has been performed. The right-hand side image shows the *lab mode* where trails can permanently be performed.

a trial today, the button is inactive and a further trial can be performed the next day. Once all field trials are performed, the app confirms that all data is collected and the button remains inactive.

Tap Input

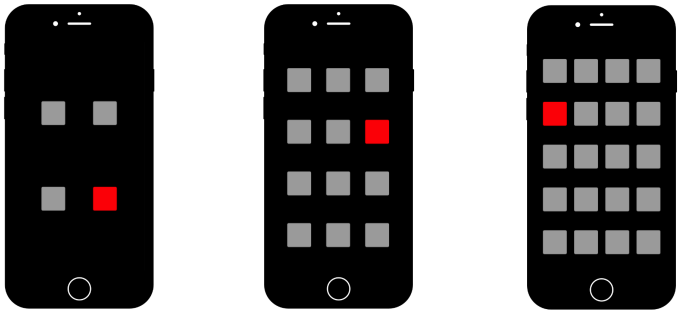

Figure 4.3: The figure shows the tap input user interface with buttons aligned in a grid shape structure. The leftmost structure offers 4 buttons, the middle offers 12 buttons whereas displays 20 distinguishable buttons.

To acquire individual user taps the mobile application offers a user interface where buttons are aligned in a grid shape structure. The structure is calculated based on a specific configuration set where the amount the vertical and horizontal buttons in the interface can be set. Figure 4.3 shows the interface, where 4, 12 and 20 distinguishable buttons are configured.

For the user to tap on every location of the screen exactly once, a red button indicating the next button to tap is highlighted guiding the user through the interaction process. While the user is tapping the grid, the gyroscope and accelerometer information is recorded. After all buttons gave been tapped, either a new grid is loaded or the tap acquisition phase ends proceeding with the question interface.

Questions

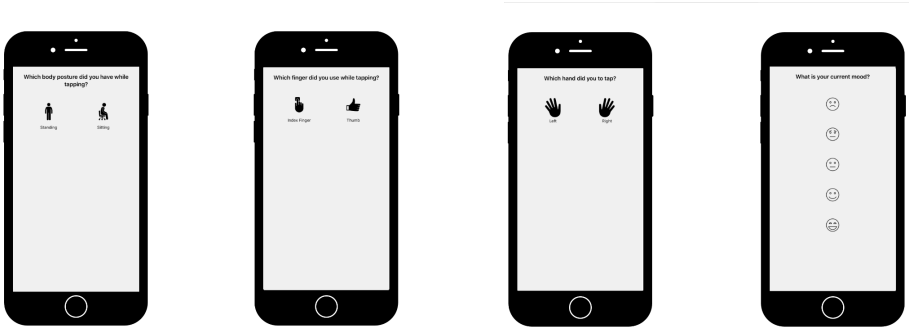

Figure 4.4: The figure displays question views with icons as answer possibilities.

To furthermore label the data acquired in the tap input interface, the application provides screens for the user to answer several questions. These question regard the various condition in which the user has generated taps. In the table below the asked questions with corresponding answer choices are to be found.

Question	**Answer Choices**
Which body posture was used during the interaction ?	Standing, Sitting
Which finger did you use while tapping?	Index Finer, Thumb
Which hand did you use to tap?	Left, Right
What is your current mood?	1 - 5

To enable fast interactions, each answer possibility to a question is represented in the interface with an icon. Once an icon has been pressed, the application transits to the next question until all questions have been answered.

Upload

After all taps and questions are gathered, the acquired data is sent to the server. The interface at this points displays a spinning wheel for the user to acknowledge that the mobile phone is processing the data. In case an upload fails, the application provides a manual upload screen, where past sessions can be uploaded.

4.2.2 Implementation Notes

Obtaining Sensor Information

In order to access the gyroscope and the accelerometer, Apple provides a high-level API[2] for accessing the device's sensors: *Core Motion*. Core Motion provides motion and environmental related data from sensors including accelerometers, gyroscopes, pedometers, magnetometers, and barometers in easy to use manner.

Sensor values can either be accessed as proceeded including aggregations of the values or as raw version. For TapSensing, raw values are recorded to avoid any form of bias. The update interval can be configures at ranges from 10Hz - 100Hz. Higher update-rates are possible but are not ensured to be processed in real-time by the device. For TapSensing, the update rate is configured with the highest (safe) value possible. This ensures that tap patterns are captured with high resolution in order to make a classification easier.

Local Persistence

It is possible that a session upload fails due to lack of internet access or another reason. Due to this, the application stores all session information and sensory data in it's local SQLite database. For local persistence Apple provides it's own framework called *Core Data* which is extensively used in TapSensing. Once the data has been successfully received by the backend, the data is deleted from the local database.

Ensuring data consistency

Sending all collected data in a single HTTP-request results in the sent package being too large for the server-side system to process. For this reason, the data is split up into packages of 300 objects per request. To send these requests in an asynchronous manner, the *Promise*[3][27] library *Hydra*[4] is used.

During an upload phase, packages are sent in a sliding window approach. Packets are sent three at a time until all packages have been acknowledged by the server-side application. For each transmitted package, the server responds with the amount of data objects contained within the request. The mobile client can therefore track the amount of packages transmitted to ensure that all data has been transmitted successfully. If one packet fails to arrive, the package is resent with an exponential back-off.

[2] An Application Program Interface is a set of rules and subroutines provided by an application system for the developer to use. The following link leads to the Core Motion API documentation: `https://developer.apple.com/documentation/coremotion/`

[3] Promises are a software abstracting for dealing with asynchronous computation. Promises are objects that may produce a value at some point in the future.

[4] `https://github.com/malcommac/Hydra`

Push Notifications

TapSensing is registered with the Apple Push Notification Service allowing the application to receive push notifications. Notifications are used to remind the user during the field study, that he has to take part in the study.

Distribution

Installing iOS applications that are not uploaded to the Apple App Store, requires each device to be registered in the Apple Developer Portal with the smartphone's serial number. To avoid this, the application has been uploaded to the App Store enabling an easy distribution.

4.3 Backend application

4.3.1 HTTP Endpoints

For the purpose of interoperability the network communication from the mobile application and the forms application to the server-side application is done via HTTP Requests in the JSON[5] format. The endpoints listed below represent tiny logic components that can be called from an external client.

HTTP Method	**URL**	**Description**
POST	/login	Provides login functionalities for the mobile client. This method returns an authentication, that is used for further requests to authenticate the user.
POST	/session	Provides upload functionality for a session data objects.
POST	/touchevent	Provides upload functionality for touchevent data objects.
POST	/sensordata	Provides upload functionality for a sensordata data objects.
POST	/apns	Retrieves the Device's APNS token. This is required to send push notifications to the users.
GET	/trial-settings	Provides the configuration for the tap input view. Here, the amount of buttons and the amount of grid repetitions that are to be performed in a single session can be defined.
POST	/survey	Provides upload functionalities for the survey form application.

[5]JSON (JavaScript Object Notation)[8] is a lightweight data-interchange format.

4.3.2 Data Model

TapSensing's data model reflect the schemas that are used in the PostgreSQL database. As seen in figure 4.5, the data model consists of 4 data objects that I will describe in this section:

- **User**: The user model is inherited by the Django's user model[6]. The user model is used for authentication and persisting the authentication token. The other models described in this section are connected to the user model through a foreign key relation to identify the user associated with the data object.
- **Session**: The session data object holds all information associated with the user's trial. This includes the data collecting in the "labeling part"[7] of the application such as the hand used, the body posture and the typing modality. In addition, information is stored such as device infos and if the session took place in a laboratory or field environment.
- **Touchevent**: The touchevent data object stores information regarding the "ground truth" of the user generated tap including the exact x and y coordinates, timestamp and an identifier of the specific grid rectangle tapped in the tap input view. Furthermore, it is noted if the user hit a specific rectangle and if the event is a touch-down or touch-up event.
- **Sensordata**: The sensordata data object captures all information obtained by the gyroscope and accelerometer of the mobile device. To differentiate between accelerometer and gyroscope values, the data object includes a type field. In addition, the model captures the timestamp and the 3-sensor components (x, y, z) of the individual sensors.

4.3.3 Implementation Notes

Authentication

The authentication mechanism implemented in TapSensing is based on a standard token authentication scheme. This allows the association of an incoming request with a set of identifying credentials, which is - in this context - the user the request came from. In order to obtain an authentication token, the mobile application sends the login credentials of a user including username and password to the login endpoint[8]. When the credentials have been checked for validity, the endpoints returns an authentication token in the following format:

[6] For more information on the Django user model, the following URL leads to the model's documentation: `https://docs.djangoproject.com/en/1.11/ref/contrib/auth/`

[7] The "labeling part refers to the question views, where additional information on the data acquired is collected"'

[8] The login endpoint is described in section 4.3.1.

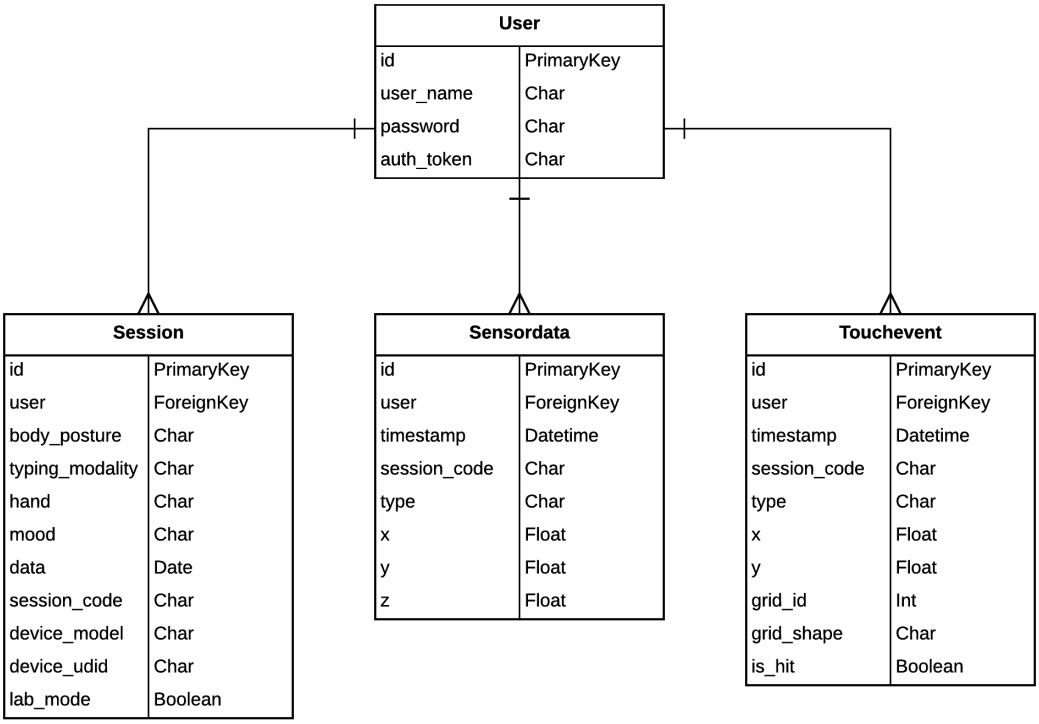

Figure 4.5: The diagram shows TapSensing's data model with the user, session, sensordata and touchevent data object.

```
{
  "token": "Token 3acc6c2a58723e2f1579d4526add2511f6a0a525"
}
```

The token is then added to the HTTP-request in the "Authorization" header to authenticate the user.

Push Notifications

As a previous study has shown that push notifications greatly enhance user participation [5] during a mobile field study, notifications are sent on a daily basis reminding the participants to take action. In the same study[5], passive notifications - notifications without sound - have been seen to futhermore impact the participation. Following these assumptions, a push notification strategy has been designed combining notifications with and without sound. Figure 4.6 shows this implemented strategy with corresponding trigger times.

The interaction with the APNS service has been implemented using the Django package

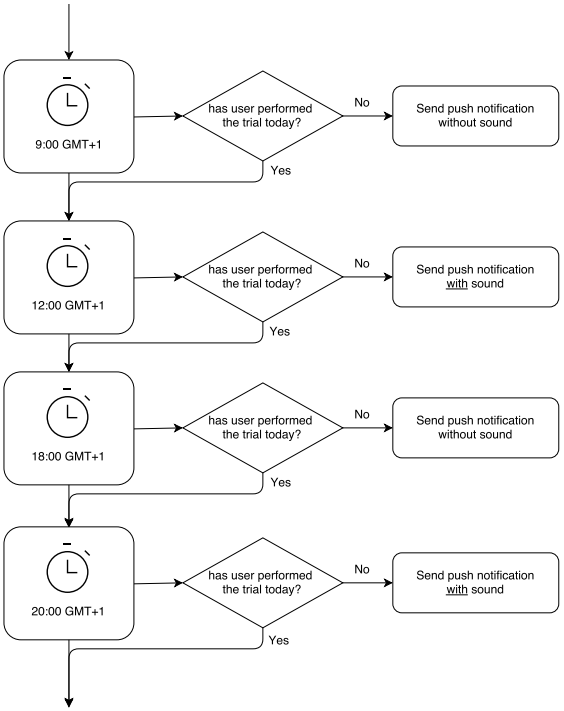

Figure 4.6: The diagram shows the implemented push notification strategy.

django-push-notifications[9]. It is used to provide mechanisms to register devices as well as to send notifications. To trigger the notifications at specific times unix cronjobs are executed.

Backups

To prevent data loss, the PostgreSQL database is backed up via a unix cronjob every evening. The database dumps are sent automatically to Amazon Web Services S3 Bucket.

Deployment

The server-side application with the NGINX reverse-proxy and a gunicorn WSGI application server has been deployed on a Ubuntu 14.04 virtual environment. The server comprises of 1GHz of shared CPU and 1 GB RAM.

[9] `https://github.com/jleclanche/django-push-notifications`

5 Data Preprocessing

5.1 Overview

1. Query 2. Split into Taps 3. Interpolate 4. Feature Extraction 5. CSV

5.2 Feature Extraction

- Explain column, matrix and sensor features.

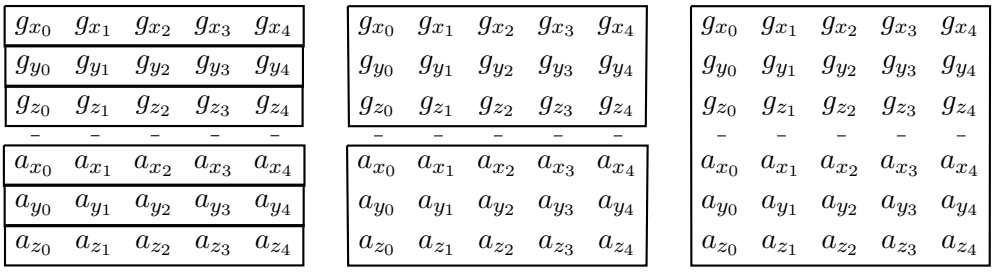

Figure 5.1: The figure shows how different features are extracted from the overall sensor matrices: Column features, sensor features and matrix features.

5.2.1 Column Features

Name	Description	Feature Type	Amount
peak	Amount of peaks in the time series	column	6
zero_crossing	Amount of zero crossings in the signal	column	6
energy	Energy of the signal	column	6
entropy	Entropy measure of the the signal	column	6
mad	Median absolute deviation of the signal	column	6
ir	Interquartile Range	column	6
rms	Root mean square of the signal	column	6
mean	Mean of the time series	column	6
std	Standard deviation of the time series	column	6
min	Minimum of the time series	column	6
median	Median of the time series	column	6
max	Maximal value of the time series	column	6
var	Variance of the time series	column	6
skew	Skewness of the time series	column	6
kurtosis	Kurtosis of the time series	column	6
sem	Standard error of the time series	column	6
moment	Moment in the time series	column	6
spline	Spline interpolation of the signal (12 steps)	column	6*12
fro_norm	Frobenius matrix norm	matrix	1
inf_norm	Infinity matrix norm	matrix	1
l2_norm	L2 matrix norm	matrix	1
cos_angle	Cosine Angle of sensor component pairs	sensor	3
pears_cor	Pearson Correlation of sensor component pairs	sensor	3

5.2.2 Matrix Features

Name	Description
fro_norm	-
inf_norm	-
l2_norm	-

5.2.3 Sensor Features

5.3 Feature Scaling

6 Method

6.1 Overview

6.2 Hypothesis

6.3 Experimental Setup

6.3.1 Subjects

6.3.2 Devices

6.3.3 Experiment Settings

Lab

Field

6.4 Analysis

7 Results

7.1 Hypothesis

7.2 Discussion

8 Conclusion

8.1 Further Outlook

List of Acronyms

3GPP	3rd Generation Partnership Project
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AS	Application Server
CSCF	Call Session Control Function
CSS	Cascading Stylesheets
DHTML	Dynamic HTML
DOM	Document Object Model
FOKUS	Fraunhofer Institut fuer offene Kommunikationssysteme
GUI	Graphical User Interface
GPS	Global Positioning System
GSM	Global System for Mobile Communication
HTML	Hypertext Markup Language
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
I-CSCF	Interrogating-Call Session Control Function
IETF	Internet Engineering Task Force
IM	Instant Messaging
IMS	IP Multimedia Subsystem
IP	Internet Protocol
J2ME	Java Micro Edition
JDK	Java Developer Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JSR	Java Specification Request
JVM	Java Virtual Machine
NGN	Next Generation Network
OMA	Open Mobile Alliance
P-CSCF	Proxy-Call Session Control Function
PDA	Personal Digital Assistant
PEEM	Policy Evaluation, Enforcement and Management
QoS	Quality of Service
S-CSCF	Serving-Call Session Control Function
SDK	Software Developer Kit
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMS	Short Message Service

SMSC	Short Message Service Center
SOAP	Simple Object Access Protocol
SWF	Shockwave Flash
SWT	Standard Widget Toolkit
TCP	Transmission Control Protocol
Telco API	Telecommunication API
TLS	Transport Layer Security
UMTS	Universal Mobile Telecommunication System
URI	Uniform Resource Identifier
VoIP	Voice over Internet Protocol
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XCAP	XML Configuration Access Protocol
XDMS	XML Document Management Server
XML	Extensible Markup Language

Annex

```
<?xml version="1.0" encoding="UTF-8"?>
<widget>
         <debug>off</debug>
         <window name="myWindow" title="Hello Widget" visible="true">
                 <height>120</height>
                 <width>320</width>
                 <image src="Resources/orangebg.png">
                        <name>orangebg</name>
                        <hOffset>0</hOffset>
                        <vOffset>0</vOffset>
                </image>
                 <text>
                         <name>myText</name>
                         <data>Hello Widget</data>
                         <color>#000000</color>
                         <size>20</size>
                         <vOffset>50</vOffset>
                         <hOffset>120</hOffset>
                 </text>
        </window>
</widget>
```

Listing 1: Sourcecode Listing

```
INVITE sip:bob@network.org SIP/2.0
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
Max-Forwards: 70
To: Bob <sip:bob@network.org>
From: Alice <sip:alice@ims-network.org>;tag=42
Call-ID: 10@100.101.102.103
CSeq: 1 INVITE
Subject: How are you?
Contact: <sip:xyz@network.org>
Content-Type: application/sdp
Content-Length: 159
v=0
o=alice 2890844526 2890844526 IN IP4 100.101.102.103
s=Phone Call
t=0 0
c=IN IP4 100.101.102.103
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.network.org:5060;branch=z9hG4bK83842.1
;received=100.101.102.105
Via: SIP/2.0/UDP 100.101.102.103:5060;branch=z9hG4bKmp17a
To: Bob <sip:bob@network.org>;tag=314159
From: Alice <sip:alice@network.org>;tag=42
Call-ID: 10@100.101.102.103
CSeq: 1 INVITE
Contact: <sip:foo@network.org>
Content-Type: application/sdp
Content-Length: 159
v=0
o=bob 2890844526 2890844526 IN IP4 200.201.202.203
s=Phone Call
c=IN IP4 200.201.202.203
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Listing 2: SIP request and response packet[**?**]