Hochschule Wismar
University of Applied Sciences
Technology, Business and Design
Fakultät für Ingenieurwissenschaften, Bereich EuI

Nondisruptive user re-identification

Development of an Identification Factor to Keep Running Sessions Alive without Active Participation of the User, Referring to a Mobile Departure Control System Application

Master-Thesis

September 25, 2020
Moritz Herbert
born on 20.05.1992
in Franfurt a. M.

Supervisor: Mr. Prof. Dr. Matthias Kreuseler
2nd Supervisor: Mrs. Prof. Dr. Antje Raab-Düsterhöft

Aufgabenstellung

Das Ziel der Master-Thesis ist die Entwicklung eines Konzeptes zur kontinuierlichen Überprüfung einer etablierten Nutzersession und deren Gültigkeit.

Dabei ist das Nutzerverhalten durch in mobilen Endgeräten verbauten Sensoren (Touchscreen, Gyroskop, Beschleunigungssensor, ...) aufzunehmen, zu analysieren und anhand einer prototypischen Umsetzung zu evaluieren.

Die Studie ist dabei auf die Auswertungen bzw. Entscheidungsfindungen mit einfachen Heuristiken zu beschränken, welche sich auf die Erhebung von Daten und der sich darauf basierenden Erstellung von Nutzerprofilen beziehen.

Das entwickelte Konzept ist mit der Verwendung von Bluetooth Beacons als besitzbasierter Authentifizierungsfaktor und Gesichtserkennung zur Überprüfung der Nutzerauthentizität zu vergleichen und zu bewerten.

Abstract

Departure Control System (DCS) applications are mostly time-critical. They require that the user interacts with it at any time to ensure that operations in the airport can run smoothly. This concerns the ultra-high availability of the applications and connected APIs and the application allowing the user to interact with it at any time.

o enable the user to use the app freely after the initial login, without endangering the confidentiality of the mass of customer data inside, a concept of re-identifying the user in a non-disruptive way is evaluated within this master thesis.

The concept contains the selection of measurements taken by various sensors of an *iPad*, on which the DCS application is running. The measurements are used to create a user's biometric profile, which is matched against the measurements taken after a certain point in time. It is evaluated whether measurements are sufficient to decide if users in front of the *iPad* changed after login.

Furthermore, this concept will be compared to the usage of iBeacons to continuously check the proximity to the mobile device and using facial recognition as an alternative biometric approach of verification.

All three approaches aim for the same goal: Decide as to whether the application may still be used. The session would only be terminated in the case of an anomaly so ensure the legitimate user is not disrupted during the regular work.

Abstrakt

DCS Applikationen sind meist zeitkritisch. Sie setzen voraus, dass der Benutzer jederzeit mit ihr interagieren kann, damit der Betrieb auf dem Flughafen reibungslos ablaufen kann. Dies betrifft sowohl die Ultrahochverfügbarkeit der Anwendungen und der damit verbundenen APIs als auch die Anwendung, mit der der Benutzer im Prinzip jederzeit interagieren können muss.

Dem Benutzer die Möglichkeit zu geben, die App nach der ersten Anmeldung frei zu nutzen, ohne die Vertraulichkeit der darin enthaltenen Masse an Kundendaten zu gefährden, wird im Rahmen dieser Masterarbeit ein Konzept zur unterbrechungsfreien Reidentifikation des Benutzers evaluiert.

Das Konzept beinhaltet die Auswahl von Messungen, die von den verschiedenen Sensoren eines *iPads*, auf dem die DCS-Anwendung läuft, durchgeführt werden können. Die Messungen dienen dazu, ein biometrisches Profil des Benutzers zu erstellen, das mit den Messungen nach einem bestimmten Zeitpunkt abgeglichen wird. Es wird evaluiert, ob die Messungen ausreichen, um zu entscheiden, ob der Benutzer des *iPad* nach der Anmeldung gewechselt hat.

Dieses Konzept wird darüber hinaus sowohl mit der Verwendung von iBeacons verglichen, die kontinuierlich die Nähe zum mobilen Endgerät überprüfen, als auch dem Ansatz der Gesichtserkennung als alternatives biometrisches Verfahren.

Alle drei Ansätze verfolgen das gleiche Ziel: Eine Entscheidung darüber zu treffen, ob die Applikation noch verwendet werden darf. Die Sitzung würde nur im Falle einer Anomalie abgebrochen, um sicherzustellen, dass der rechtmäßige Benutzer während der regulären Arbeit nicht gestört wird.

Contents

1 Introduction

The chapter clarifies the necessity and initial situation of the project as well as essential circumstances and previous history leading to the master thesis founding.

1.1 Outline

The following list summarizes the different chapters of this thesis.

Chapter one describes the environmental circumstances, the motivation to write the master thesis, and the necessity of the project.

Chapter two gives a comprehensive introduction to biometric systems and, in this regard, information specific to the prototype of this thesis.

Chapter three clarifies the technical environment in terms of used technologies and design patterns.

Chapter four describes measurements and metrics that have been chosen to be put in place as part of the prototype.

Chapter five deals with the architecture of the practical elaboration of this thesis.

Chapter six treats the implementation of the practical elaboration of this thesis.

Chapter seven gives insights into the analysis of the usefulness of the chosen measurements and the chosen approach in itself.

Chapter eight gives an outlook into possible future development.

Chapter nine explain the usage of beacons and its pros and cons.

Chapter ten introduces the last theoretical approach discussed within the thesis.

Chapter eleven targets the final comparison of the approaches.

Chapter twelve considers the progress of the thesis and a conclusion.

1.2 Amadeus IT Group

Amadeus IT Group, from now on referred to as *Amadeus*, is one of the most extensive Global Distribution System (GDS) and travel Information Technology (IT) providers worldwide. The GDS has initially been founded in 1987 by four major European airlines with its headquarters located in Madrid, Spain, and became an independent company years after.[1]

1.3 Amadeus Airport Companion App

The *Amadeus Airport Companion Application* is a mobile Business to Business (B2B) application, based on *iPadOS*, which has been under development since 2018. As a DCS application, mainly focusing on the customer management within airports, it is used by airline ground personnel, for example, to serve customers in case of disruptions or give general guidance.

The application can only be used by logging in using a personalized account. The backends are secured with several layers of defense. A user session needs to last for hours to ensure that the airline agents can adequately work with it. This gives attackers the possibility to abuse the application for multiple hours if they manage to obtain an *iPad* after login and keep the session alive.

1.4 Motivation and Initial Situation

Various data is and needs to be collected when traveling internationally. The data is required to enable the traveler to reach a target destination both safely and reliably. Apart from that, the data is used to protect affected countries by making sure the traveler can be identified uniquely. For these purposes, the Personally Identifiable Information (PII) processed by the travel industry applications include traveler name, country of residence, payment data, and even information about the medical condition.

Most of the time, airline agents working in airports process the data using stationary devices located behind counters or kiosks. Therefore any software-related security measures, such as role-based user authentication, are supported by this physical hurdle.

Assuming that this hurdle could be removed, an attack that requires a malicious user's physical presence would be much easier to execute after a legitimate user has started a session. This is the case when agents switch to mobile devices, such as tablets, to serve customers right where they need support or service.[2, chapter 2.7.1]

In scenarios where planes arrive with customers on board who were already informed that their connecting flight had been canceled, agents do not have the time to log in again after a short period of inactivity or even use Multi Factor Authentication (MFA) to resume sessions.

The purpose of this thesis is to find a way of ensuring the legitimate usage of a mobile B2B application without restricting the device in regards to User Experience (UX). Therefore the interaction between the user and the device and all its sensors (touch screen, gyroscope, accelerometer, etc.) is recorded to create a biometric profile. The profile is matched against the interaction that happens after enrolling in the biometric profile to spot deviations from the original profile.

The recorded measurements' usefulness is mainly analyzed, and ideas of matching the behavioral fingerprints are presented. This approach is compared with two alternatives: The usage of a physical authentication factor in the form of an iBeacon, which could be detected to prove the proximity between the mobile device and the user, and face recognition to verify users' authenticity periodically.

2 Biometric Systems

This chapter will describe the creation of a Biometric system. Firstly, the characteristics of a biometric system will be defined, and the components are listed. Secondly, the operation modes are explained. Afterward, it will be explained how the quality of a running biometric system can be quantified. These steps are aligned with the chapter 'INTRODUCTION' of the book *Introduction to Biometrics* by *Anil K. Jain, Aron A. Ross, and Karthik Nandakumar*, which gives well-structured information about the characteristics of biometric systems [3]. Additionally, the already built-in biometrics of *iPadOS* are introduced. Finally, 'heterogenous feature-level fusion' will be briefly touched, as it is especially relevant for the thesis' central approach.

2.1 Purpose and Definitions

The ability to identify individuals has played a significant role in human society for centuries. As populations speed up in growing more and more, it has become increasingly essential to associate human attributes to distinct persons. During the transition from smaller tribes, where everyone knows each other, to modern society arranged by countries and even host of countries, it is crucial that affiliations can be identified. Democracy, health care, unemployment benefits, safe travel, to name only a few examples, would not be possible without ways of identifying people uniquely. Elections can only take place if one can ensure that people vote only once. Criminal offenders wanted by the local police could easily cross borders without being recognized.

Traditionally, identification can be achieved by probing into the knowledge of a secret, possession of an individual item, like an ID Card or a Passport. A biometric system offers its operators the possibility to enroll a (sometimes large) set of profiles, each consisting of a person's single or multiple equatable characteristics, such

as fingerprints, iris, face, or posture features. These physical features have one significant advantage over the traditional approaches of identification or verification: The characteristics are ideally unique compared to all other human beings. Since they are attached to the user, they cannot be lost, forgotten, or replicated easily.

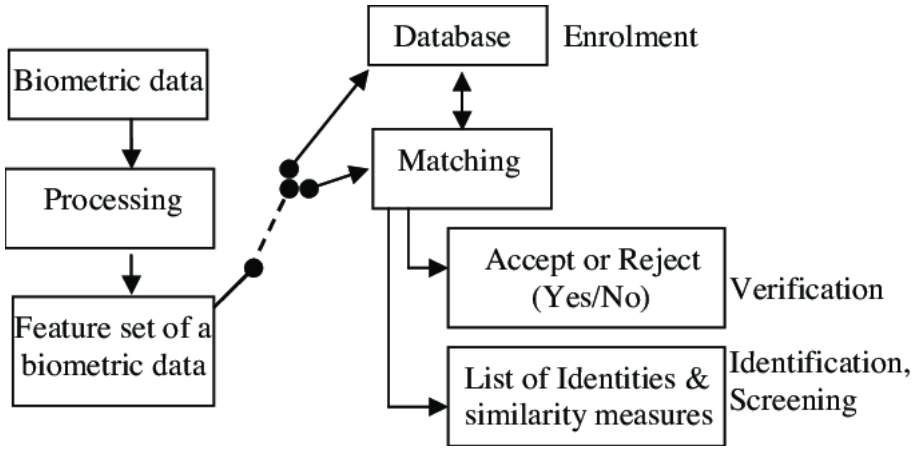


Figure 2.1: A generic biometric system [4, page 185]

The components of a biometric system will be introduced in the following sections. They all reference to Figure 2.1 showing the main steps that are performed when running a biometric system.

2.1.1 Biometric Data

Biometric data is present to the system in its raw form and translated from the physical characteristic to the data. The data is, depending on the kind of biometric system, read by various types of sensors. For facial or posture recognition, it could be a camera, for palm- or fingerprint, e.g., a capacitive sensor. This raw form of data contains not only the features that would be used for further analysis but also irrelevant data or even interfering factors like motion blur, background noise, or insufficient light.

2.1.2 Processing

As described in the previous section, raw data is more than what is needed for further processing. The biometric system is most certainly not used under perfect

conditions, and extraneous circumstances can impact the performance. The selection of the appropriate biometric system is one factor in optimizing the accuracy and, thus, usefulness. For example, the constant lack of light might lead to the decision that face recognition would not be the optimal way of deciding whether a worker is allowed to access a construction site. In any case, before the feature extraction (next section) is performed, the optimization of raw data is likely to be part of the process. After optimization, which can come in different shapes, concerning the specific biometric system, it is checked whether the data is suitable enough to be used for feature extraction. If not, the capture would be dropped, and if active interaction between the subject and the biometric system interface is part of the system, the user might be prompted for another attempt. The system might also trigger an exception or could try to acquire another capture automatically. Also, part of the preprocessing is the segmentation of data, including cropping an image to the bounds of a face or isolating a voice. In most cases, it is the removal of background noise.

2.1.3 Templates

Templates result from the extraction of decisive information from the - most likely previously enhanced - raw data. The feature set or template is comprehensive enough to represent the raw data digitally but, on the other hand, concise enough to be storage efficient and to not slow down matching templates. For a fingerprint, the template would be a set of minutiae points, meaning the points where the ridges of the fingerprint end, or a bifurcation of ridges. In addition to that, the matching is enhanced by segmenting the fingerprints based on specific criteria.

2.1.4 Storage

Biometric systems mostly contain two different workflows: The enrolment and the matching. During enrolment, the system stores a template in its database. During the matching, the acquired data is directly pushed to the matcher (after preprocessing). In this mode, the storage is used to read previously enrolled templates and provide them to the matcher as well.

2.1.5 Matcher

Due to the previously explained external factors like weather or light, it is nearly impossible to acquire the same data as from the sample, which was stored during enrollment. Matcher can, therefore, work in two different ways. They either calculate the degree of similarity or the degree of dissimilarity between the template and sample and create a score out of that.

2.1.6 Decision Module

The decision-making module, which can be a part of the matcher, decides whether the match is accepted or rejected. One important step during the biometric system setup is the calibration of the threshold of the decision-making module. When comparing the degree of similarity, a high threshold would bring greater security to the system and increase the chance that true matches are not recognized as such. Hence, there is always a tradeoff between security and usability when setting thresholds, depending on the area of use. It might be acceptable to retry when opening a bank vault, but unlocking a phone should typically work after the first attempt.

2.2 Operation Modes

Biometric systems can be used to either **identify**, or **verify** an identity. The main difference is the number of comparisons to be performed before concluding. Verification refers to an identity claiming to be someone specific, enrolled in the system, and therefore only matching the probe against one template is required. The identification mode would, in the worst case, match against all entries of the database. Identification can happen for two main reasons:

- During enrolment, to prevent the user from enrolling in the system multiple times.
- During recognition, when testing, whether the user is known to the system, without claiming a specific identity.

Verification means that the similarity between the presented feature and the claimed identity's template is supposed to exceed a pre-defined threshold to generate a match.

A biometric system can produce a match in identification mode by comparing a sample to the database entries until the threshold is exceeded. It could also return a list of the top X most similar findings.

2.3 Errors Rates and Other Quality Measures

Traditional biometric systems are based on the premise of relying on unique and continuous characteristics to be sufficiently robust. Reasons for errors shall not be discussed any further in this thesis if they cannot be directly associated with the performance prototype.

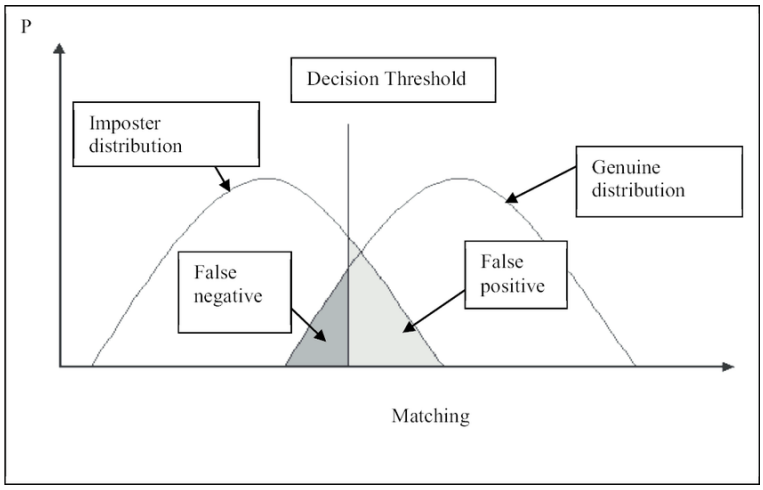


Figure 2.2: Correlation between error rates and decision threshold [5, page 483]

However, there are several metrics that can be correlated to the chosen decision threshold, as illustrated in 2.2. The resulting rates will be listed below, along with their description.

FTE *Failure To Enrol Rate:* Number of failures to create and save a template to the database divided by all enrollments

FTA *Failure To Acquire Rate:* Number of failures to acquire a sample divided by all acquired samples

FAR *False Acceptance Rate:* Imposter matches that were mistakenly accepted as genuine divided by all matches

FRR *False Rejection Rate*: Genuine matches that were mistakenly rejected divided by all matches

GAR *Genuine Acceptance Rate:* All genuine matches, that were recognized as such. The rest $(1 - FRR)$, compared to **FRR**.

TAR *True Acceptance Rate:* Synonym for **GAR**

Even if a trait does not belong to the owner of the template, it might have similarities. Comparing two samples of the same trait should lead to a significantly higher score. However, the perfect match might indicate fraud because it would require the acquisition's internal and external conditions to be identical compared to before. As well, the trait presented on/in front of the sensor would need to be in the same condition and presented in the same way. This is why the decision threshold is so critical. Setting it too low might make the system weak; setting it too high might make it unusable.

The performance of a biometric system can be visualized by plotting the GAR against FAR. The result is called Receiver Operating Characteristics (ROC) [6]. There would be no overlap between true rejections and true matches at a certain threshold, or even a range of thresholds, in an optimal biometric system; thus, the system would not produce any errors. In this scenario, the Area Under the Curve (AUC) is equal to 1 and would take the diagram's entire space. A biometric system performs the worst when the AUC equals to 0.5, which means a total overlap of genuine matches and true negative matches. The system was then not able to distinguish between the two states.

Figure 2.3 shows an exemplary diagram that compares the different ROC curves of fingerprint sensors paired with different algorithms.

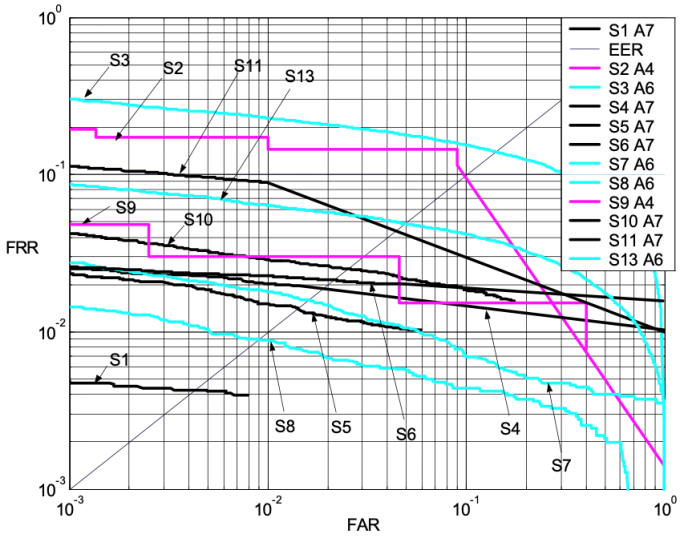


Figure 2.3: ROC curves in comparison [7, page 66]

2.4 iPadOS Built-in Biometrics

Depending on the different *iPad* models, biometric authentication is available in two different ways on newer devices (through fingerprint matching since 2015 and face recognition since 2018)[8, 9]. The fingerprint and face recognition, branded by *Apple Inc.* as *Touch ID* and *Face ID*, are services embedded into the operating system. However, the services do not provide APIs to retrieve the biometric templates, but only to trigger the matching and to return the result of the operations. At any time, the process of using the user's biometric trait for authentication is transparent to the user, which means it cannot be run in the background without blocking the User Interface (UI). This makes it useless for re-authenticating the user from time to time since it would introduce interruptions of the regular *iPad* usage and, therefore, greatly impact usability. In addition to that, *Face ID* and *Touch ID* are not built to support multiple users. A user can enroll one alternative template in *Face ID* or a few fingerprints (depending on the device) for *Touch ID*. While the primary, scenario described in this thesis, showcases an application used by several agents across their shifts, the built-in biometrics become mostly unsuitable.

2.5 Heterogenous Feature-level Fusion

There are several ways of fusion to enhance the performance of a biometric system. Fusions can happen at the sensor, feature, store, or decision step [10, Chapter 7: Biometric Fusion]. Since this thesis's prototype is going to implement the heterogenous feature-level fusion, only this type will be explained. To build a robust set of metrics to create a metaphorical fingerprint of a user's way of utilizing an *iPad*, several sensors, like the accelerometer and the touch screen sensors, are readout. This makes it a heterogeneous fusion. The values are transformed into enums or numerical metrics, which are heterogeneous per se, even if extracted from the same sensor. Still, the number of metrics should be manageable for the matcher. The metrics are stored together as one user profile; thus, the fusion happens on the feature level. The matcher should not be concerned by the origin of the single values, but compare the degree of similarity.

3 Applied Technologies

3.1 Swift

Swift is a programming language announced by *Apple Inc.* in 2014. It was designed to replace C-based languages, focusing on safety (e.g., no undefined behavior), speed, and an easy to learn and easy to master syntax. Most of its parts were open-sourced in December 2015 under the Apache 2.0 license. Parts of the runtime library remain closed-source. *Swift* projects can currently be run on *Apple Inc.* devices and on Linux. This includes mobile devices like tablets and smartphones, but also applications running on a computer.[11]

3.2 iBeacons

iBeacon is a proprietary protocol presented by *Apple Inc.* in 2013[12]. Hardware accessories that implement the protocol can use the Bluetooth Low Energy (BLE) technology to transmit information to devices with compatible software nearby.[13] *Apple Inc.'s* Core Location APIs, which are available in the *Swift* language (described in the previous section), can read the information sent by the iBeacon transmitter. The approximate proximity to the iBeacon can read out as three categories: **Immediate** (a few centimeters), **Near** (a few meters) and **Far** (more than around 10 meters). By design, the signal of the iBeacon is transmitted unsecured, which means that the message can be read by anyone and will always be the same as long as the data held by the beacon does not change.[14]

3.3 Splunk

Splunk Inc.'s about page states the following:

„Splunk is the world's first Data-to-Everything Platform designed to remove the barriers between data and action so that everyone thrives in the Data Age.“[15]

Splunk is meant to transform machine data into operational insights. *Splunk Enterprise* can be clustered into five main components: **Index Data** is the component that takes raw data of any format and tries to categorize the contents into known source types and normalizes the data to be stored on the searchable indexes. **Search & Investigate** offers the possibility to create search queries to retrieve filtered data and run analysis on top of that data. **Add Knowledge** can be used to tag data for further use. It provides the interface to enrich the data with additional descriptions. **Monitor & Alert** would automatically survey data in real-time that is pushed from the different sources and could notify operators if needed. **Report & Analyze** is used to dashboard the data in a comprehensive visual overview.

Splunk can take data from all sorts of sources that might be structured or unstructured, streamline this data and transform it into events stored on searchable indexes. The data that is fed to *Splunk* can be used to automate log analysis and trigger alerts whenever irregularities have been detected, or to simply dashboard and analyze data manually.[16]

Splunk is going to be used to analyze the raw event logs generated as part of the thesis. Therefore, the logs are uploaded to the *Splunk* platform, and different queries are used to create tables and dashboards to understand and visualize the logged data.

4 Measurements and Metrics

The prototype is designed to work as a library that can be hooked into any application built using *Swift* version 5 or above. This limits the capabilities to measure touch gestures sightly since it is harder to access the original components and properties than if the prototype was natively integrated into an app. There are still various measurements to be analyzed.

The measurements used to create a profile describing the user's behavior will be listed and explained in the following. Measurements are recorded using the touch screen sensors, the accelerometer, and already built-in indicators, which combine accelerometer data with other electronic parts that can sense motion.

4.1 Touch screen data

The following measurements are recorded using touch screen sensors. The data is provided by *Swift*'s *UIGestureRecognizer* class. The measurements are partially raw data, but some need to be computed by the prototype.

4.1.1 TouchForce

The physical force with which the user has performed the screen interactions. It is represented by a floating-point number, where 1.0 is the average non-user specific force set by the system.

4.1.2 TouchRadius

The touched surface radius represented as a floating-point number without considering the tolerance, which could also be accessed.

4.1.3 Deflection

This metric is calculated using the initial touch-point of a gesture and where the touch has ended. The coordinates of both points are used to determine the deflection, represented by the eight cardinal points (north, northeast, ...) of the touch. Depending on the proximity of a gesture's start and endpoint, this measurement is supposed to target different goals. It is supposed to show in which direction a tap deviated from its original touch point for close proximity. For larger gestures, it is supposed to show the direction in which the gesture was executed.

4.1.4 LinearStrokeDeviance

Assuming that a swipe gesture was supposed to represent a linear finger movement, this metric shows how straight the gesture was performed. The floating-point number is gained by measuring the distance between the imaginary line segment if the line was straight, and the point in the actual line that deviated the most.

4.1.5 LinearStrokeDevianceDirection

Similar to the *Deflection* measurement, this one is used to calculate the cardinal point to which the performed gesture was roughly pointing. For example, if a vertical scroll action was performed, and if the swipe was slightly bent to the left, either southwest for downwards or northwest for the upwards movement would be logged.

4.1.6 StrokeDistance

A floating-point number, showing the distance between the start and endpoint of a swipe gesture. The result is measured in points.

Note: To simplify the metric, the non-linear finger movement between the start and the endpoint was ignored. This measurement has been added after the first test series was recorded.

4.1.7 StrokeSpeed

A floating-point number, showing the speed of finger movement. Start and endpoints are used to calculate the gesture distance. It is then divided by the time it took to perform the gesture. The resulting unit of this measurement is $\frac{pt}{ms}$ (points per millisecond).

Note: Since it relies on *StrokeDistance*, the measurement is automatically reduced to the essential straight line distance between the gesture's start and endpoint.

4.1.8 TapDuration

The time in microseconds from touching the screen initially to when the finger has been lifted entirely. This metric is only calculated for tap gestures.

4.1.9 RelativeTapDevianceDirection

For smaller UI elements, like buttons, this metric is used for logging, where on the element, a tap was placed. Therefore the center of the element and the touch-point is taken into consideration to determine the cardinal point.

4.2 Tilt

The *Swift CMMotionManager* is used to capture this measurement. It logs the pitch of the device every 1 second. Figure 4.1 shows as a blue line around which axis the *iPad* turns when talking about the pitch.

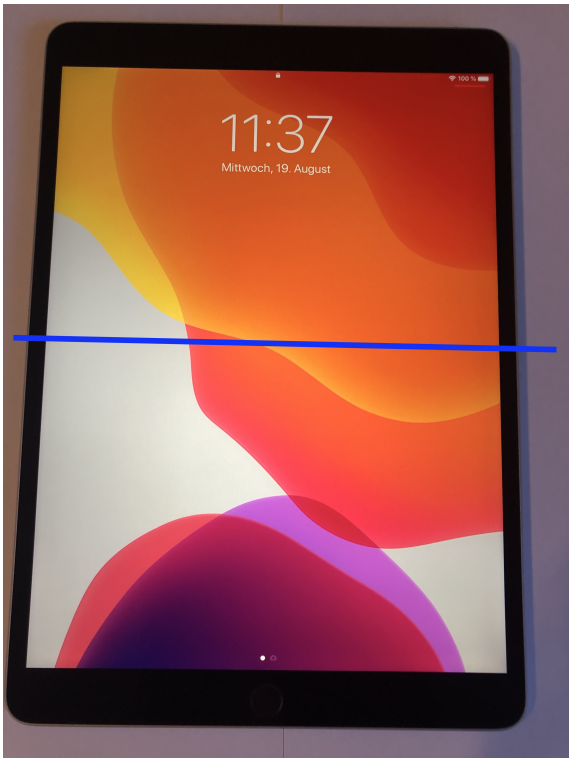


Figure 4.1: Pitch axis of the *iPad* visualized as blue line

The device attitude is available in two different formats in *Swift*. It can be read out as quaternions, but as only the pitch is needed for that measurement, and the value is available as one of the three Euler angles, it can be directly read from there. The floating-point number provided by the Application Programming Interface (API) is measured in radians, where one-quarter of a circle equals to $1.5707963268 = \pi/2$. Therefore, the scale goes from -1.5707963268 to 1.5707963268, which means that if the *iPad* stood upside down, the pitch would be -1.5707963268; if it would lay flat and leveled, the pitch would be 0 and 1.5707963268 in case its upside is up. The values in between are ambiguous since there is no distinction between tilting it towards the user so that the display is starting to face downwards or away from the user where the display would point upwards.

4.3 Motion

To create these indicators, several sensors of the device are readout. The *Swift CMMotionActivityManager* outputs multiple boolean flags to indicate whether the device is stationary, the user is walking or running, the device is inside a moving automotive, or on a bicycle. These values' confidence is provided in three categories (low, medium, high).

These Apple internal values are mapped to three states: *stationary*, *notStationary* and *inMotion*. The second value describes a state in which the device is neither stationary nor in actual motion. This is mainly when a user is lying, sitting, or standing while controlling the device without having it mounted on a stand or laid on a table.

5 Proof of Concept Architecture

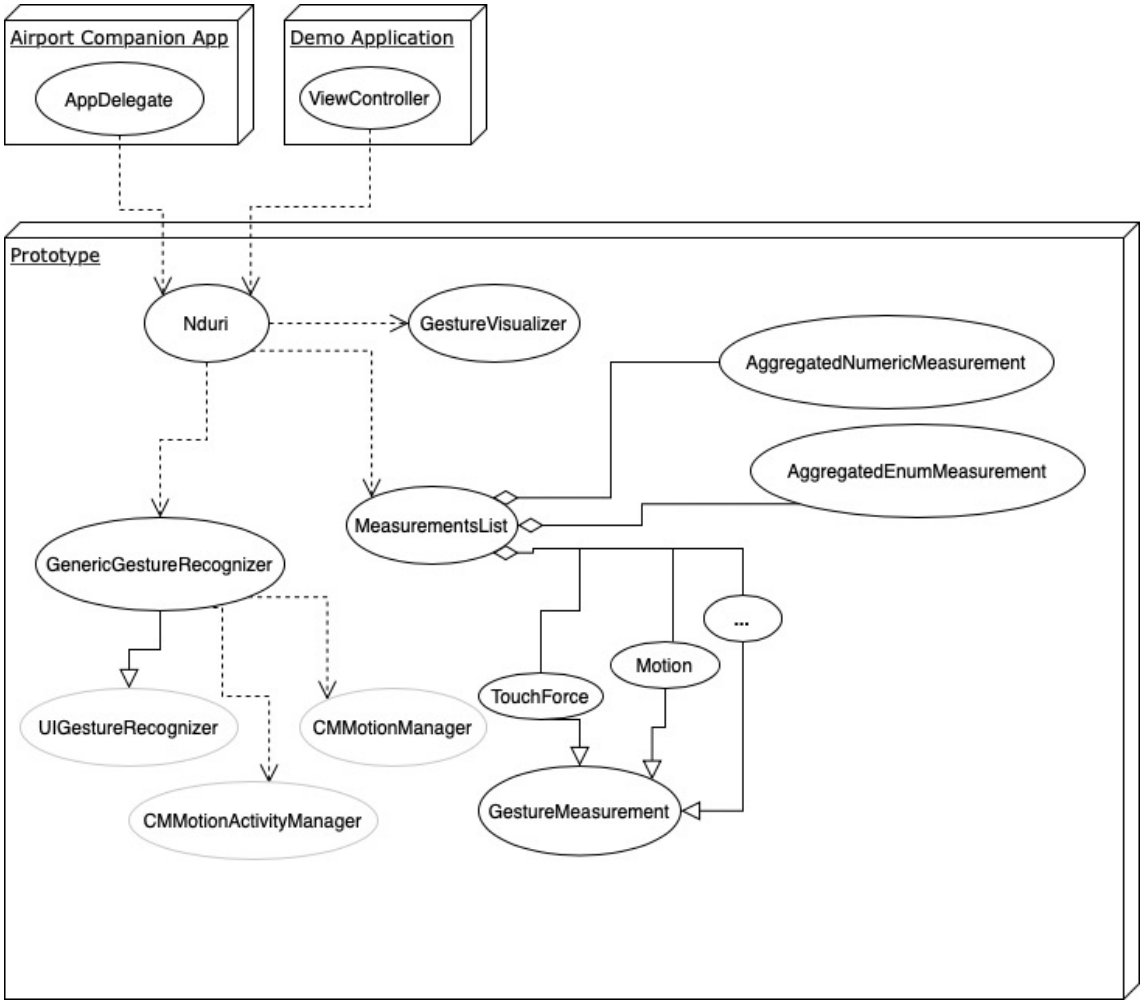


Figure 5.1: High-level Architecture of the Prototype

The Proof of Concept (PoC) has been build as a *Swift* package. This means that it is loadable by every application making use of the *Swift Package Manager*. The components will be described briefly within this chapter with the help of Figure 5.1. The core of the package is the *GenericGestureRecognizer* class. It captures the raw interaction between the user, screen, and other *iPad* sensors. In case conditions are met, to qualify as a measurement or metric described in chapter 4, the data is pushed

into the *MeasurementsList*. This class can append every item to its internal list, which inherits from *GestureMeasurement*. *GestureMeasurement* is used to create a unified structure for all measurements, defining the data field (yet of type '*Any*') and a not alterable timestamp when instantiated. The *MeasurementsList* can be JavaScript Object Notation (JSON) encoded. This is used when exporting the data for further analysis. For several reasons, *Apple Inc.'s Airdrop* technology was selected to handle the export.

- It is easy to trigger the share functionality built into *iOS / iPadOS*
- It does not require the setup of a server
- The user remains fully responsible for the disclosure of the data collected

In the first iteration of the PoC, data is not aggregated but exported in its raw form. The exported data will be analyzed manually.

The second iteration implements the aggregation of the data. The export functionality is obsolete since the computed template and samples are compared, and decision-making is done solely on the device. This approach has the advantage of being detached from the stability of network connectivity. Moreover, it does not keep the network adapter busy, concerning a large number of transmissions over time (if not already aggregated on the device).

Nduri is the heart and central access point of the prototype, exposing other necessary components through it's API. The host application can include the package by calling the function `Nduri.shared.setup(in: UIView)` inside the so called *AppDelegate / UIApplicationDelegate*. It is responsible for providing lifecycle hooks, like when the app has been created, is about to be minimized (will be brought to background), or brought to the foreground again. The *AppDelegate* holds an array of all application windows, wherein the case of many, the last item is the foremost one. The *GenericGestureRecognizer* needs to be attached to that last window, which is ultimately the root view of the screen.

From then on, every event is tracked by the *GenericGestureRecognizer*. It exposes an API as well for exporting the data. As mentioned before, the export is only an intermediate step for further development of the package, so the *Airdrop* export is implemented inside the host application.

For debugging purposes and show-casing, the prototype includes a demo application. To visualize events that include finger movement in the screen, it calls `Nduri.shared.addVisualizationTo(window: UIWindow)` as soon as the application's view did appear. *Nduri* utilizes the *GestureVisualizer* to draw the paths, that follow finger movements from then on. The demo application also outputs the logged measurements alongside the visualization.

6 Implementation

6.1 Data Extraction

Data extraction combines the data gathered from different sensors. The *GenericGestureRecognizer* is capable of reading on-screen gesture data by overriding the hooks of its superclass *UIGestureRecognizer*.

None of the gestures are narrowed down to a certain type, like swipe, tap, or using one or more fingers when inheriting from *UIGestureRecognizer*. It would have been an option to implemented various gesture recognizers for all kinds of different gestures and attach these to the host application's main view, but this limits the measurements to the available set of gestures. To be precise, seven concrete subclasses are available since *iOS* 7. Furthermore, gestures have different priorities in *Swift*. By default, only the recognizer with the higher priority is considered by *Swift*, and the rest is ignored. This is mainly done to prevent ambiguous behavior of the UI when gesture recognizers act concurrently. To overcome this, the following function had to be implemented (hook provided by the *UIGestureRecognizerDelegate*):

```
1 func gestureRecognizer(_ gestureRecognizer:
    UIGestureRecognizer,
2                        shouldRecognizeSimultaneouslyWith
    otherGestureRecognizer: UIGestureRecognizer) -> Bool {
3     gestureRecognizer is GenericGestureRecognizer ||
    otherGestureRecognizer is GenericGestureRecognizer
4 }
```

Listing 6.1: Simultaneus Gesture Recognizers

It enables all other gesture recognizers of the host application to still work alongside the prototype's *GenericGestureRecognizer*. By overriding the function, one can decide which gesture recognizers can act simultaneously. It is called for every combination of recognizers that try to act at the same time. This would be more and more

prone to error as the number of recognizers rises, which is also an argument against the usage of multiple recognizers within the prototype. In the case of the *GenericGestureRecognizer*, it can be done without side effects on the host application, since the recognition of a gesture never leads to a UI state change.

The primary functions that have been overridden by the PoC are: `func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent)`, `func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent)` `func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent)` All functions share the same signature. The most important argument of the function, in regards to the prototype, is a set of *UITouch* objects, where each item represents a single finger on the screen. According to the Developer Documentation, *UITouch* is supposed to give access to

- The view or window in which the touch occurred
- The location of the touch within the view or window
- The approximate radius of the touch
- The force of the touch (on devices that support *3D Touch* or *Apple Pencil*)

Since the event listener is attached to the entire window, the touched view is always the same, and therefore, the most inner view needs to be computed manually. A recursive function, called *frameOfTouchedView* has been developed to extend the API of *UITouch*. It returns the frame of the leaf element of the UI components tree that has been tapped, to mainly give access to its width, height, and position. The frame is used along with the second item of the list above, to compute the *RelativeTapDevianceDirection* measurement. The last two remaining items on the list can be used without transformation for the *TouchForce* and *TouchRadius* measurements.

The three digest cycle hooks that have been listed above act in the following way[17]:

1. Whenever a user lays one or more fingers down on the observed view, *touchesBegan* is triggered for every initial contact separately.

2. (optional) If the user does not release the finger(s) right after, but starts a movement, *touchesMoved* is called repeatedly until the movements stops.

3. Finally, *touchesEnded* is called when a gesture was completed.

Multi-touch gestures are not considered as part of the PoC. Consequently, the implementation of gesture recognizer aborts as soon as more than one finger is detected. On *touchesBegan*, a stopwatch is started to track the tap duration and the duration of stroke gestures. The array to track the gesture path is reset, and the initial touch location is stored. *touchesMoved* does only add the new location of the finger to the gesture path array. *touchesEnded* triggers the computation of most of the measurements and pushes them into the log.

An example for a computed measurement is *LinearStrokeDevianceDirection*. It is supposed to gain confidence in whether the user controls the *iPad*, preferably with the left or right hand. The possible values of this measurement are *Direction.west* and *Direction.east*.

The direction is calculated by finding the determinant of the vectors from starting point of the gesture (**S**) to end point of the gesture (**E**) and from starting point of the gesture to the point with the maximum deviation of the line from start to end (**P**): $d = Det(S, E, P) = (Ex - Sx) * (Py - Sy) - (Ey - Sy) * (Px - Sx)$. If the sign of the determinant (**d**) is negative, and (**Sy**) is greater than (**Ey**), it means that it deviates to the left, else to the right. It is the contrary, for results with positive signs, because it indicates the direction from the perspective of the startpoint and not from the person in front of the screen.

$$D(d, Sy, Ey) := \begin{cases} west & \text{for } (a < 0 \wedge Sy > Ey) \dot{\vee} (a \geq 0 \wedge Sy \leq Ey) \\ east & \text{for } (a < 0 \wedge Sy \leq Ey) \dot{\vee} (a \geq 0 \wedge Sy > Ey) \end{cases}$$

Every particular measurement is inheriting from *GestureMeasurement*. While *GestureMeasurement* defines the data field as type *Optional(Any)*, a get-only timestamp and a *computed var* (variable) to return the data field as *String*, the actual measurement subclasses define a customer init function to set the proper type of the data field. This is an example measurement definition:

```
1 public class Tilt: GestureMeasurement {
2     init(data: Double) {
3         super.init(data: data)
4     }
5 }
```

Listing 6.2: Tilt Measurement

By having a common superclass, the subclasses can be put inside a strictly typed array. The *computed var* is used to be able to create a representation of the measurement, that can be JSON encoded and then exported for further analysis:

```
1 public var dataString: String {
2     if let nonNil = data, !(nonNil is NSNull) {
3         return String(describing: nonNil)
4     }
5
6     return ""
7 }
```

Listing 6.3: Data To String Conversion

Computed var in *Swift*, means that its value is calculated by the instructions given in the closure, that follows the type declaration, every time it is used. The first half of line 2 shows the attempt of unwrapping the optional data field. *Optionals* are fields that may or may not hold a value during runtime. If the *optional* holds a value, it is further checked if the value is not null. These are the prerequisites of being able to cast it to a string. To ensure that the *computed var* is always a string, even if the actual data field does not contain a value that can be represented as one, the fallback value is defined as an empty string (line 7).

During the initialization of the *GenericGestureRecognizer*, the two listeners for *iPad* motion measurements are put in place. The setup is slightly different for both, the *CMMotionManager* and the *CMMotionActivityManager*. The first one provides a feed of data from the different sensors, like the accelerometer, and the activity manager will notify when the state has changed. It means for the *Motion* measurement that only the changes between 'stationary', 'notStationary', and 'inMotion' are logged, and the time during the events can be analyzed afterward. This leads to the slightly heterogenous post-processing since all other metrics can be build by calculating averages and deviations out of the raw measurements right away.

In contrast to *Motion*, the *Tilt*/pitch needs to be captured in intervals manually. To be precise, the prototype captures the motion and, therefore, the device attitude once every second. The result in radians is logged without further processing. For more info about the values, see chapter 4.2.

To calculate the *Deflection* values, four thresholds have been defined for the slope value derived from the startpoint and endpoint of a gesture: -2, -0.5, 0.5, and 2. The thresholds have been visualized in Figure 6.1. In which eighth of the coordinate system the slop falls, determines the *Deflection*.

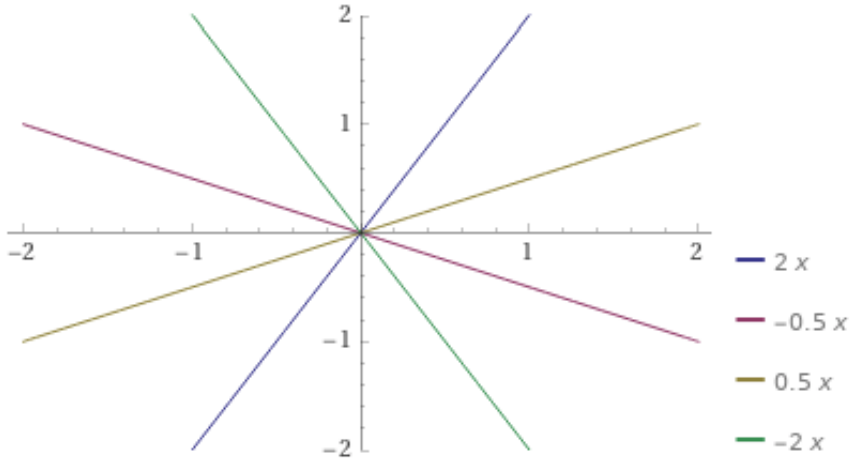


Figure 6.1: Eight directions between the lines $-2x$, $-0.5x$, $0.5x$, and $2x$

6.2 On-Screen Visualization

For a minimum of visualization within the demo application [1] , which is shipped within the prototype's *Swift package*, all gestures involving movement of a finger on the screen are drawn as orange paths on top of the demo application.

The start and endpoints are connected by a straight, blue, dotted line, as displayed in Figure 6.2. The same visualization can be easily integrated into other applications. However, to not disturb test subjects during data collection, the overlay was not added on top of the foremost view of the *Airport Companion App*. To draw the path

[1] Please note that this application does contain very few view components and would therefore not be suitable for creating actual test data that could give significant hints to the usefulness of this prototype.

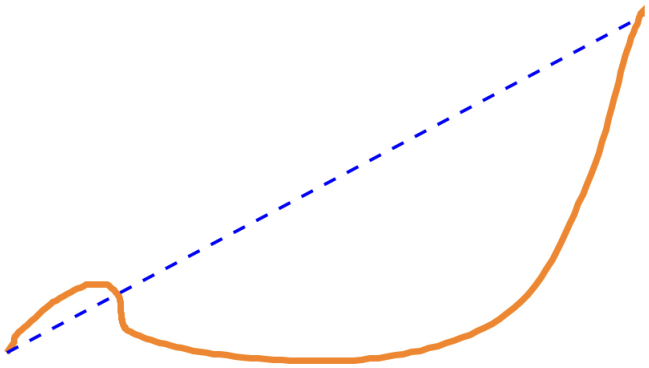


Figure 6.2: Screenshot of the visualization of a randomly executed gesture

and the straight connector line, the application in which the prototype is embedded simply needs to listen to the *gestureEnded* event of the *GenericGestureRecognizer*. Every time the event is fired, a new gesture path can be retrieved from the instance of the *GenericGestureRecognizer*. It is then passed on to the visualizer for display.

All entries that are pushed into the measurement log are also printed to the console to get instant feedback when testing functionality.

6.3 Storage and Test Data Generation

The first iteration of the prototype was simply holding an array of all logged entries in a raw format. Arrays are held in memory only, similarly to arrays in the C language, but with slightly more overhead, since *Swift* makes, for example, sure that the chain of points from memory location to memory location is safe to use. There is technically no limit to the size of an array, apart from the available memory. While access to specific array entries has the complexity of $O(1)$, which is constantly 1 regardless the array's size, the analysis of the logged measurements requires the exact opposite: Touching each of the entries at least once ($O(n)$).

For the first iteration of the prototype, where the array is accessed only during the export while encoding it as a JSON object, this should not cause any issues having

in mind that the sample data is collected within one hour. The test setup consists of a couple of Quality Assurance (QA) scenarios that were current at the time of data collection. The scenarios are several regression tests paired with the validation of a newly developed feature. They are reproducible and, consequently, suitable to generate datasets that contain a more or less equal sequence of tracked events. This approach's benefit over random application usage is that it best suits analyzing both the inter-user and intra-user variability. Especially the consistency in behavior can be analyzed with this approach since a single person can repeat the scenarios on different days, and the results can be compared easily. This could give good insights into whether it is also possible to save templates over several sessions and compare them with the actual values immediately after login. It would otherwise only be possible to build up templates at the beginning of a session before starting the verification phase.

The main weakness of this type of storage is its volatility. If a QA scenario would lead to an app crash (which was, in fact, the case for one of the first test cases), the array holding the already tracked measurements and, hence, the associated memory is lost. This leads to delays in this thesis's data collections phase and additional work for all test subjects.

In the real world scenario, where an airline agent uses the prototype throughout the entire shift, the data would be collected for around eight hours (initial data collection and data collected to compare with the computed template). This might not be an issue for the memory to ultimately hold an array that results in around 3,5 MB when converted to JSON[2]. Still, as one can imagine, traversing the raw array is the most inappropriate approach to detecting anomalies efficiently. How the data is prepared and analyzed will be explained in the next section.

6.4 Processing and Aggregation

Without further optimization of the raw data, it would be a challenge in terms of computation, to process the information and make decisions. For some measurements, the computation of the sample mean and variation might be sufficient;

[2]Extrapolation of the sum of the test data to 8 hours.

others might also require the temporal distance between the most current measurement and the previous one of the same kind. Hence, a set has been created to hold the minimal aggregated data needed for the analysis.

- the cardinality (size of the list of measurements of a certain type)
- the sample mean
- the sample standard deviation
- the last recorded sample

The mean is calculated incrementally, so that not all n values need to be stored:

$$\bar{x} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

Likewise, to incrementally calculate the standard deviation whenever a new sample needs to be considered, the following formula has been used to first calculate the variance:

$$\sigma_n^2 = \sigma_{n-1}^2 + (x_n - \bar{x}_{n-1})(x_n - \bar{x})$$

The standard deviation can be easily derived from the variance for a cardinality of two and above:

$$\sigma = SD(n) := \begin{cases} 0 & \text{for } n < 2 \\ \sqrt{\frac{\sigma_n^2}{n-1}} & \text{for } n \geq 2 \end{cases}$$

For every kind of measurement, an entry in the set exists with the above-listed attributes. Using mean and standard deviation might be an overly simplified approach to calculate a matching score. However, this thesis's focus lies in analyzing the available measurements and might, therefore, sufficient for the thesis' prototype.

Some measurements, like *Motion*, hold strings inside their data field that have been derived from enumerations. For *Motion* in particular, the string literals of *stationary*, *notStationary* or *inMotion* are stored. This data is non-numeric data that does not

allow the calculation of average values or standard deviation; therefore, only the enum cases' occurrences are counted as part of the PoC's aggregation.

Motion does even differ from other measurements that hold enums within their data field. The reason for that is the assumption behind: The longer the device is in motion, the more certain one can be that the application should prompt for credentials again. Yet, this assumption is bound to the specific use case of the *Airport Companion App*, where agents might not walk for a long period after they reached their working area; hence the usefulness should be revised when industrializing the approach.

7 Manual Analysis

The following chapter explains to the reader the analysis of the collected data and the conclusions drawn.

The analysis is done three steps.

At first, the questions that should be answered by the analysis need to be defined. Not only do the questions need to be asked in a way that they can be measured, but the data also needs to be suitable for answering the questions. This prototype aimed to collect various measurements and see whether they could answer the questions. Nonetheless, the questions need to exist before or at least need to be raised during the measurements' design to ensure that they aim in the same direction.

Secondly, the data needs to be analyzed. This can happen by plotting on diagrams, creating pivot tables, and finding correlations between measurements.

Finally, the results need to be interpreted to conclude.

The three steps are successively described and documented in the next sections.

7.1 Questions to Answer With the Help of Data Analysis

The main question regarding this thesis's topic is whether the approach of turning behavior into a fingerprint-like template that is comparable to a sample from other people's behavior or even to a sample of the same person at another point in time is possible at all. However, it might be easier to ask the contrary: Is it impossible to do so? Eleven measurements have been picked to support the approach. The measurements aim for different purposes and try to cover a broad range of possible measurements. The list of measurements is not exhaustive but should indicate whether there is a chance of creating proper metrics out of the measurements that

could support a product robust enough to sense the difference between non-identical users. Proving that every or at least most of the measurements are useless should be sufficient to decide whether the approach has failed.

The first step to reach this knowledge lies in analyzing every measurement in an isolated manner to answer the following questions:

- Do samples of the same measurements occur in any kind of pattern?
- Does the pattern show continuity over a certain period?
- Do the patterns or single measurements differ enough between test subjects?
- Does the pattern differ from session to session of the same subject?

At least the first three questions should be answered with yes to keep the measurement for further analysis.

Multiple metrics should be combined into a fingerprint-like set of characteristics to strengthen the expressiveness of a template. This approach is comparable to what is done in terms of bot protection for web applications and APIs by, e.g., *Imperva's* 'advanced bot protection'. *Imperva* claims that the use of just IP addresses would only lead to the recognition of around 40% of all calls initiated by bots, while the false positives drop to zero when relying on four different sets of attributes like IP address, request header information, and sender device attributes.[18] Nonetheless, fusion, in this case *heterogenous feature-level fusion*, in biometric systems, does not always lead to better results. The amount and compatibility of the metrics needs to be carefully tested when a complete system is ready.

After identifying suitable measurements, if any, the next step is to build up metrics for the fingerprinting. Every single one of the direct measurements defined within the prototype might not only be converted into a single metric, but it might result in combinations of different values. Before the correlations can be detected, another set of questions becomes relevant:

- Can the single measurements be treated similarly?
- Do only two groups of measurements (numerical and enums) need different approaches of analysis?

- Can numeric and enum measurements be easily unified to one or more metrics?

Regardless of the particular measurements and metrics, another aspect of the analysis covers finding different strategies that could be applied to the prototype's evolving implementation. To find and develop the strategies, the main questions that have been picked are:

- Does the repetition of similar scenarios lead to similar behavior?
- Is the behavior of the same person differing from session to session?

Different test series have been set up to address all the above-listed questions successively. These series and their purpose are described in the next section.

7.2 Recording Setup

Two different test series have been put in place to analyze the data regarding different aspects. The series will be explained briefly, and their purpose highlighted referring to the questions mentioned in Chapter 7.1.

7.2.1 Inter-User Variation

The main chunk of logs was generated by two subjects to compare the inter-user similarities regarding their app usage when confronted with the same tasks. While the first subject had to document the work during the scenarios' execution, the second subject did just replay the scenarios consecutively without meaningful disruption. This leads to fewer events in total since the *Tilt* measurement is recorded every second and therefore triggered more often, but in both cases, the recoding led to enough tracked events. To be precise, the same 14 different scenarios were chosen for both the subjects. An example of such a scenario is: "Identify the passenger with the following surname 'XYZ', on the specific flight '1A 123', at a given date. Try to accept the customer for check-in and resolve all the pending regulatory issues on the way." On the one hand, such scenarios prevent the subjects from randomly using the application. On the other hand, they give the user a certain degree of flexibility while reaching this goal.

iPadOS, as most operating systems, offers multiple controls to reach the same outcome (mainly for greater UX). An example of that is the back button usage, located at the top left corner of the screen within the navigation bar. On *iPadOS*, it can be substituted by swiping from the very left edge of the screen towards the center. Both interactions result in navigating between the current and the previously shown screen.

The outcome is the same for both gestures, but the tracked events are different regarding the described prototype's logs. Navigating via the back button will trigger logging of *TouchRadius*, *TapDuration* and *RelativeTapDevianceDirection*, while the swipe gesture results in logging *LinearStrokeDeviance*, *LinearStrokeDevianceDirection* and *StrokeSpeed*.

Ultimately, this test series pursues comparing two different subjects' behavior, but it will also be used to analyze the particular measurement.

7.2.2 Intra-User Continuity

One big advantage of conventional biometrics compared to traditional authentication factors like using passwords is the continuity of biometric traits. Fingerprints might be rendered useless by cuts and bruises or being exposed to water for a more extended period, but the trait will typically revert to normal. Hence, the biometric traits are hardly lost, stolen, or forgotten.

The second test series was executed by the same subject spread over multiple sessions. It is supposed to demonstrate the continuity in behavior over time. The test is set up so that the subject will have to perform the same steps within every repetition. This includes that the aim of the scenario and intermediate tests have to be described, but all device interactions. The physical setup is also mattering. While the first test series allowed the user to sit on a table and occasionally lay the device down, the second series is meant to be recorded standing upright. All interactions are described as unambiguously.

Ideally, this setup should lead to the same number of tracked events for all repetitions (not including the *Tilt*, because all runs would have to last the same length to the second). The scenario itself is fairly quick to execute but run ten times with small breaks in between:

1. Pick up the *iPad* and stand up
2. Kill the *Airport Companion App* using the app switcher
3. Open the *Airport Companion App* and let the auto-login authenticate
4. Tap on the 'Flight' tab
5. Select search-mode 'Flight List'
6. Tap on the date input field and use the picker wheel to scroll three days into the future
7. Clear the 'From' input field by tapping on the 'x' symbol
8. Type 'LHR' into the 'From' input field
9. Select 'Arrivals'
10. Tap on the 'Search' button
11. Tap on the 'Status' column header to open the filter menu
12. Select 'Not Disrupted'
13. Dismiss the menu using the 'Cancel' button
14. Tap on the 10^{th} row
15. Tap on 'Menu'
16. Tap on the menu entry 'Customer List'
17. Use swipe gesture three times to go back to the root view of the 'Flight' tab
18. Tap on the 'Settings' tab
19. Tap 'Export Nduri log'
20. Share the log via Airdrop

21. Lay *iPad* on the desk

Wrong execution leads to immediate discarding of the test run.

7.3 Raw Data

The raw data outputted by the first iteration of the prototype is a large array of events where every entry contains the measurement name, the logged value, and a timestamp.

```
[
    {
      "event" : "TapDuration",
      "data" : "20142.167",
      "datetime" : "2020-08-21T14:03:52Z"
    },
    {
      "event" : "RelativeTapDevianceDirection",
      "data" : "northeast",
      "datetime" : "2020-08-21T14:03:52Z"
    },
    ...
]
```

Listing 7.1: Sample Raw Data Exported as JSON

The data has been uploaded to a trial version of *Splunk Enterprise* for further analysis. *Splunk* and *Splunk Enterprise* will be used synonymously from now on. The instance is locally deployed, cannot be accessed from other machines, and is only set up for analyzing the data produced by the prototype. Therefore it does not interfere with data coming from other sources.

7.4 Manual Data Correction

After the recording of the first subject's behavior, an error within the data was detected while uploading the dataset to *Splunk*, that was related to the default

date encoding used by *Swift*'s JSON encoder. An example for the default encoding, which *Apple Inc.* calls the `JSONEncoder.DateEncodingStrategy.deferredToDate` is `619711430.68811297`. It corresponds to `2020-08-21T14:03:50Z`, if the International Organization for Standardization (ISO) 8601 format is applied. An *Xcode* playground, which is mainly a small, headless standalone project to evaluate smaller pieces of *Swift* code, has been set up to correct the encoding. It is done in two steps. Firstly, the corrupted log file is imported and decoded, and each event is decoded to the original *LoggableMeasurement* object. Finally, a new instance of the *JSONEncoder* is created and configured to encode the dates properly and pretty-print the output. The output is written to a new file. It is essential to highlight this post-processing step for parts of the recorded data since it adds a layer to the data extraction process, which could introduce undetected errors. The outputted ISO 8601 formatted dates match the approximate time frame in which the data was initially recorded, making it very unlikely that the re-formatting changed the date-time value.

7.5 Data Import

With the proper date-time format in place, *Splunk* is immediately able to recognize the source type and normalize the data to single event rows. This can be found in Figure 7.1.

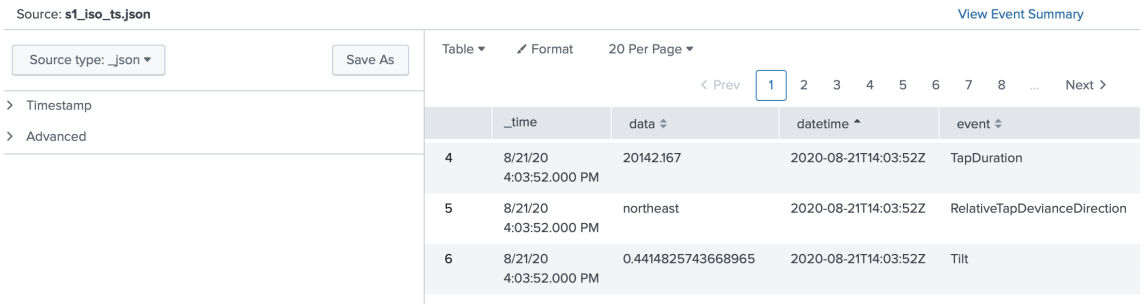
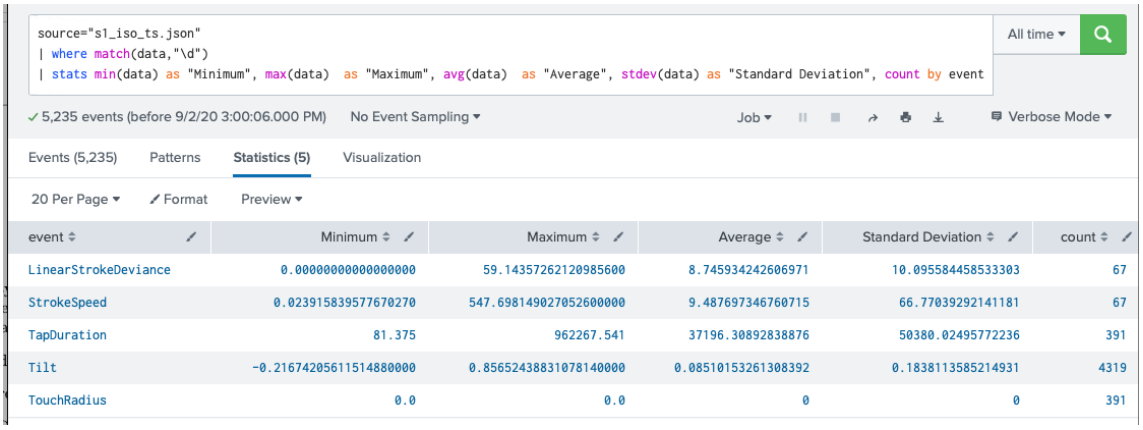


Figure 7.1: Screenshot of Splunk Data Import

The source type (JSON) and the three fields ('data', 'datetime', and 'event') of each event were automatically detected. In total, the dataset of Subject 1 contains 6235 events, recorded over roughly 1.5 hours.

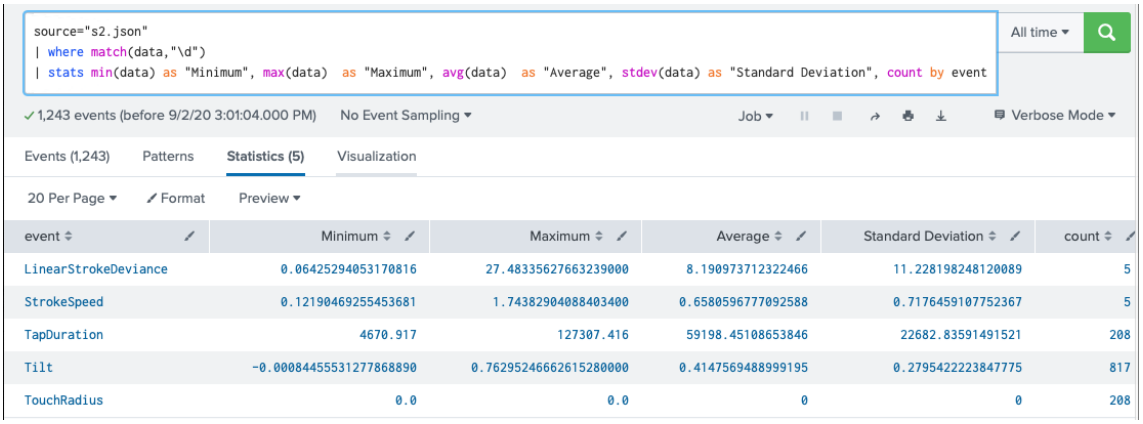
7.6 Data Visualization, Correlations, and Analysis

As previously noted, the analysis of numerical and non-numerical measurements has been split to determine whether it is possible to apply the same, or at least similar, strategies to all the same kind measurements. The numeric measurements can be seen in Figures 7.2 and 7.3 respectively. The non-numeric measurements are displayed in Figures 7.4 and 7.5 further down.



event	Minimum	Maximum	Average	Standard Deviation	count
LinearStrokeDeviance	0.00000000000000000	59.14357262120985600	8.745934242606971	10.095584458533303	67
StrokeSpeed	0.023915839577670270	547.69814927052600000	9.487697346760715	66.77039292141181	67
TapDuration	81.375	962267.541	37196.30892838876	50380.02495772236	391
Tilt	-0.21674205611514880000	0.85652438831078140000	0.08510153261308392	0.1838113585214931	4319
TouchRadius	0.0	0.0	0	0	391

Figure 7.2: General stats of numeric measurements (Subject 1)



event	Minimum	Maximum	Average	Standard Deviation	count
LinearStrokeDeviance	0.06425294053170816	27.48335627663239000	8.190973712322466	11.228198248120089	5
StrokeSpeed	0.12190469255453681	1.74382904088403400	0.6580596777092588	0.7176459107752367	5
TapDuration	4670.917	127307.416	59198.45108653846	22682.83591491521	208
Tilt	-0.00084455531277868890	0.76295246662615280000	0.4147569488999195	0.2795422223847775	817
TouchRadius	0.0	0.0	0	0	208

Figure 7.3: General stats of numeric measurements (Subject 2)

7.6.1 General Observations and Inter-User Variation

The most obvious rows in Figures 7.2 and 7.3 are the ones for the *TouchRadius* measurement, because even though the event has been triggered several times, the

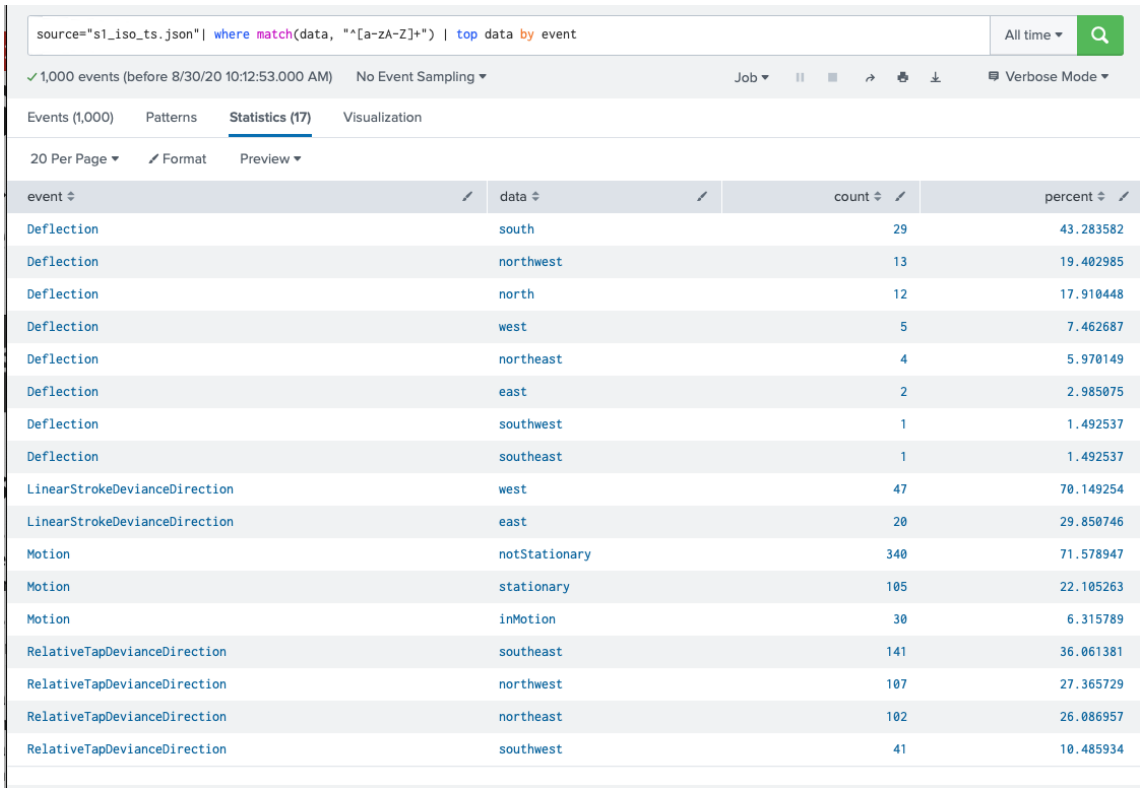
values are all 0 for both subjects. The second aspect that is closely tied to the *TouchRadius* always being 0, is the fact that *TouchForce* is missing entirely. It can be explained by the limitation to specific devices that support these measurements. Without knowing which devices would ultimately be used for testing, both measures seem to have potential from a theoretical perspective. The force applied to the device while performing gestures and the increasing amount of skin touching the screen surface with higher pressure might be highly individual. Nevertheless, this assumption cannot be proven, since neither of the *iPad* in use (*iPad* Air 3rd gen. and *iPad* Pro 10,5" 2017) were supporting *Apple*'s 3D Touch technology, which would require an additional sensor to respond to differences in applied pressure.

Due to missing sample data and the fact that the measurements would be bound to specific *iPad* models, it eliminates both measurements from the list of candidates for decision making. Furthermore, it does not seem that *Apple Inc.* plans to expand the range of supported devices, since a light version of 3D Touch, called 'Haptic Touch' was introduced, which does not require specific hardware support. Oversimplified, 'Haptic Touch' is a press-and-hold gesture that can trigger secondary options on a UI component, like opening an inline menu. [19]

Proceeding to look at the tables, especially the 'count' row, shows significant differences (again not considering the *Tilt* measurement due to the time constraint). While Subject 2 performed the 14 scenarios using only 7.5% of the number of swipe gestures as Subject 1. The tap gestures are significantly less, as well. As described in section 7.2.1, the user was free in their choice of how to execute scenarios even though rough steps to reach a goal were defined. A diverging number of total tracked events can therefore have multiple reasons: The user might have done extra steps that were not strictly required; The UI could have been misused by entering incorrect data into input fields and correcting the mistakes, or by activating and stopping the screen recordings for documentation purposes. Therefore, a metric derived from all (except the *Tilt* measurement) is the overall number of tracked events.

When looking at *LinearStrokeDeviance* from this high-level perspective, only the maximum value differs significantly between the two test subjects. The minimum value, average, and standard deviation are close to each other. *StrokeSpeed* and *TapDuration* differ the most between the subjects regarding their high-level stats. In Average, Subject 1 performs swipe gestures more slowly ($\sim 9.49 \frac{pt}{ms}$ vs. $\sim 0.66 \frac{pt}{ms}$) and with a much higher inconsistency looking at the standard deviation. All three measurements need to be analyzed more intensively to recognize patterns in the data

and identify possible jitter. The *TapDuration* is, on average, about 47% lower for Subject 1 and the standard deviation is again mich higher, leading to more widely distributed values.



source="s1_iso_ts.json"| where match(data, "^[a-zA-Z]+") | top data by event — All time

✓1,000 events (before 8/30/20 10:12:53.000 AM) — No Event Sampling — Job — Verbose Mode

Events (1,000) | Patterns | Statistics (17) | Visualization

20 Per Page | Format | Preview

event	data	count	percent
Deflection	south	29	43.283582
Deflection	northwest	13	19.402985
Deflection	north	12	17.910448
Deflection	west	5	7.462687
Deflection	northeast	4	5.970149
Deflection	east	2	2.985075
Deflection	southwest	1	1.492537
Deflection	southeast	1	1.492537
LinearStrokeDevianceDirection	west	47	70.149254
LinearStrokeDevianceDirection	east	20	29.850746
Motion	notStationary	340	71.578947
Motion	stationary	105	22.105263
Motion	inMotion	30	6.315789
RelativeTapDevianceDirection	southeast	141	36.061381
RelativeTapDevianceDirection	northwest	107	27.365729
RelativeTapDevianceDirection	northeast	102	26.086957
RelativeTapDevianceDirection	southwest	41	10.485934

Figure 7.4: General stats of enum measurements (Subject 1)

Coming to the overview of non-numeric measurements, the *Deflection* is the upmost one in the tables of Figures 7.4 and 7.5. *Deflection* on its own depends on the context of the application rather than the user's behavior. It affects the interpretation of the *LinearStrokeDevianceDirection* measurement, which will be discussed next. *Deflection* has the same sum of fired events as *LinearStrokeDevianceDirection*, which is a gesture intended to only concern real swipe gestures. This means it has never been fired for any execution of a tap gesture. This was a bug in the prototype's code, which has been adjusted for the second test series. Gestures with slight finger movement should be considered as inaccurate tap gestures rather than swipe gestures. Also, a misperception of the `touchesMoved` function of the *UIGestureRecognizer's* API impacted the test results: A gesture can end without triggering the function, but the initial touchpoint and endpoint might have different coordinates.

A condition has been put in place afterward to distinct between tapping and swiping

```
source="s2.json"
| where match(data,"^[a-zA-Z]+")
| top data by event
```

All time

✓ 406 events (before 9/3/20 9:37:25.000 AM) No Event Sampling Job Verbose Mode

Events (406) Patterns **Statistics (12)** Visualization

20 Per Page Format Preview

event	data	count	percent
Deflection	west	2	40.000000
Deflection	east	2	40.000000
Deflection	south	1	20.000000
LinearStrokeDevianceDirection	west	4	80.000000
LinearStrokeDevianceDirection	east	1	20.000000
Motion	notStationary	169	89.893617
Motion	stationary	19	10.106383
RelativeTapDevianceDirection	southeast	80	38.461538
RelativeTapDevianceDirection	northwest	72	34.615385
RelativeTapDevianceDirection	northeast	43	20.673077
RelativeTapDevianceDirection	southwest	11	5.288462
RelativeTapDevianceDirection	west	2	0.961538

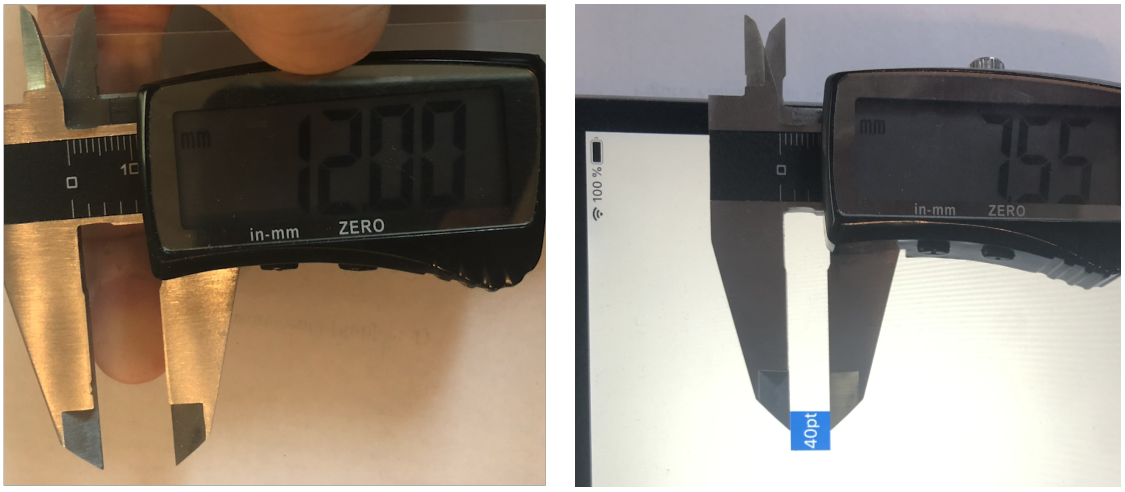
Figure 7.5: General stats of enum measurements (Subject 2)

by setting a threshold for the proximity of the initial touchpoint and endpoint at 15pt[1]. Also, triggering *Deflection* was changed to disregard the 'moved' status of a gesture after the first test series. This threshold has been chosen as it is desired to be less than the shortest distance needed to perform a swipe. Continuous debugging has shown that it is challenging to perform swipe gestures below 20pt. On the other hand, it was possible to push the flat fingertip hard on the *iPad*'s surface to log distance values close to 40pt, without intending to move the finger while pressing. On an *iPad* Pro 10.5", 40pt translate to 7.55mm (Figure 7.6(b)) while the touched surface of an index finger with medium pressure might be around 12mm (Figure 7.6(a)), keeping in mind that the size of a finger, as well as the applied pressure, cannot be seen as constants.

The adjusted code was used to track the measurements of **the second test series**, and the measurement will be discussed as part of it again.

Supposed that swipe gestures are not executed entirely linearly, but more with a

[1]Note that this is not the font size measure, but a unit that has been put in place by *Apple Inc.* to measure the on-screen distance and size of objects related to screen size and pixel density



(a) Finger on acrylic glass; medium pressure applied

(b) 40pt square on *iPad* Pro 10.5"

Figure 7.6: Physically tapped surface; Virtual sizes in points vs. physical Millimeters.

slight radial movement caused by the rotation of the wrist or finger joints, the executing hand can be inferred depending on the gesture's direction. The deviance might most likely point **away from the center of rotation**. This is illustrated in Figure 7.7, which shows pictures of the visualization of the sample app, which was made part of the prototype.

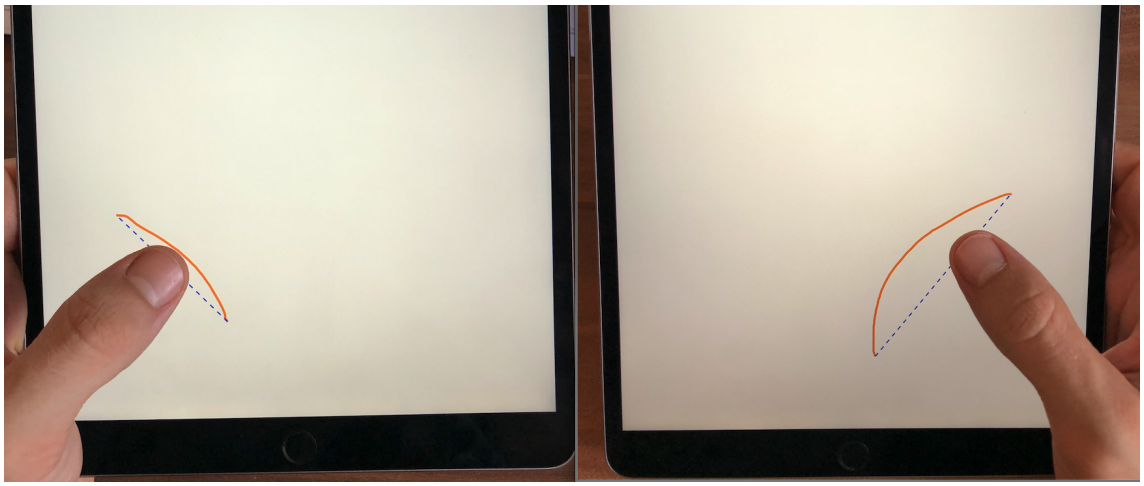


Figure 7.7: Pictures of a Swipe Gesture Performed With the Left Vs. the Right Hand

Under this assumption, swipes deviate to the right (east), to the left (west)[2], or almost not depending on the value of *Deflection* and the hand used. All likely

[2]Left/west and right/east are always meant to express the deviance looking away from the start point of the gesture towards the endpoint.

combinations, when holding the device, are shown in Table 7.1.

Deflection	LinearStrokeDevianceDirection	
	Left-Handed	Right-Handed
north	east	west
northeast	-	west
east	west	
southeast	west	-
south	west	east
southwest	-	east
west	east	
northwest	east	-

Table 7.1: Likeliest Combinations of Deflection and LinearStrokeDevianceDirection

Figure 7.8 supports the assumption that the diagonally (relative to the screen) executed gestures might deviate very little if the hand is parallelly aligned to the gesture's path.

deflection_data	avg(deviance_data)
south	13.54346636274176
northeast	10.759789460882898
east	9.688172627419162
west	5.255638069833289
north	4.508888669930919
northwest	3.4687541052825077
southwest	2.944848238456622
southeast	2.3780606413195398

Figure 7.8: Subject 1: Averages of Deflection Grouped by Deflection

The above-explained correlation of *Deflection* and *LinearStrokeDevianceDirection* cannot be read from the overall stats, since it is necessary to look at the pairs for every tracked gesture.

Figure 7.9 shows the *Splunk* search results for Subject 1's combination of the two measurements and ranking them based on the number of occurrences. The most executed swipe gestures were, by far, south strokes deviating to the right. This suggests that the subject was using the left hand. However, the overall distribution

	combined_data	count	percent
1	south - west	25	37.878788
2	north - west	8	12.121212
3	northwest - east	7	10.606061
4	northwest - west	6	9.090909
5	south - east	4	6.060606
6	north - east	4	6.060606
7	west - west	3	4.545455
8	west - east	2	3.030303
9	northeast - west	2	3.030303
10	east - east	2	3.030303
11	southwest - west	1	1.515152
12	southeast - west	1	1.515152
13	northeast - east	1	1.515152

Figure 7.9: Subject 1: Deflection and LinearStrokeDevianceDirection Combined

gives no apparent tendency to either side. With more subjects available, the measurements could result in three categories: left-handed usage, right-handed usage, ambidextrous usage.

7.6.2 Intra-User Continuity and Comparison to First Test Series

The data analyzed in this section originates from the test scenario described in Chapter 7.2.2. The focus will again first lie on the high-level observations, followed by drill-downs regarding each measurement's detailed data. The prototype contains the new measurement *StrokeDistance* at the time of recording the logs for this test series. The measurement is supposed to bring more meaningfulness to the *StrokeSpeed* measurement, which is calculated relying on the distance.

Figure 7.10 shows the changes of *Tilt* over the ten test runs. Starting at roughly $0.63 rad \ / \sim 36°$, the numbers slightly decrease to finally land at sightly below 30° in average.

This could indicate that the user intends to hold the *iPad* at a certain angle, but then the wrist adjusts the angle due to the device's weight. The above-explained behavior could be one pattern to look out for when determining the change of users.

Figure 7.10: Intra-User Tilt (Y-Axes Normalized: 0.46rad - 0.68rad)

Recalling the data from the first test series, Subject 2 had an average of 34.377°, when removing all data points from the array, where the device laid flat on the desk ($-0.1 < datapoint < 0.1$). This proves the continuity of the measurement over several days as well as in different postures. Applying the same filter to the logs of Subject 1, who ran the second series of tests, *Tilt* has an average of about 19°, which is far away from the other subject's range as mentioned above.

Except for three seconds during the second test series's fourth run, the device did detect that it was neither stationary nor in motion. In total, *Motion* was logged 105 times across the ten test runs. This leads to an accuracy of over 98%. *Tilt* has only fluctuated by a few hundreds of a radian during this period (Figure 7.11), which has led to the device perceiving itself as stationary in position. It is still necessary to not only rely on the *Tilt* measurement to determine the motion, since *Tilt* does not take acceleration into account. It is theoretically possible to move with very little deltas of the pitch, yaw, or roll by mounting the device on a self-stabilizing 3-axis gimbal.

The tendency of *LinearStrokeDevianceDirection* towards one of the two possible values continues to hold for the second test series. 30 out of 30 swipe gestures were performed with a westbound curve. Having in mind that the only swipe gestures to be performed during the execution of the second test series are horizontally aligned, support the theory that these kinds of swipe gestures are bent upwards between the

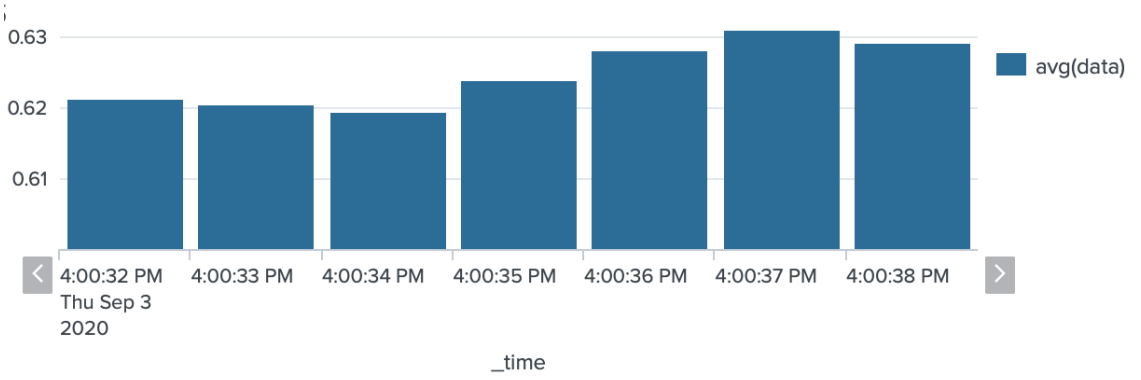


Figure 7.11: Drill-down on tilt where motion was recognized as stationary

start and endpoint. Swiping from the left edge of the screen towards the center will result in a westbound gesture, while the right to left executed gestures might result in the opposite.

Figure 7.12: Relative Tap Deviance Direction per Run (light green = southwest, darker green = southeast, brown = northeast, orange = northwest)

Where the user has tapped inside a leaf-view of the UI hierarchy-tree, is reflected by the *RelativeTapDevianceDirection* measurement. Figure 7.12 shows pie charts for every test run of the second test series. Except for the first run, the southeast is the most tapped area, followed by the northeast. In any case, the eastern side of a view takes either close to or above 75% of the taps. Since all of the tapped UI elements (during the test runs) are relatively wide but not very high, there might be random factors involved for the northern/southern pairs. However, both axes' behavior seems to show some continuity: East/right before west/left and south/down before north/up. An observation during execution backs the theory that this might be a behavior triggered by a subconscious preference: The subject did switch hands

frequently during the execution of the runs. Mostly the south-east axis could be influenced by left-hand/right-hand usage. Half of the test runs would need to be executed purely with one hand and the other half of the runs with the other hand to prove this theory ultimately.

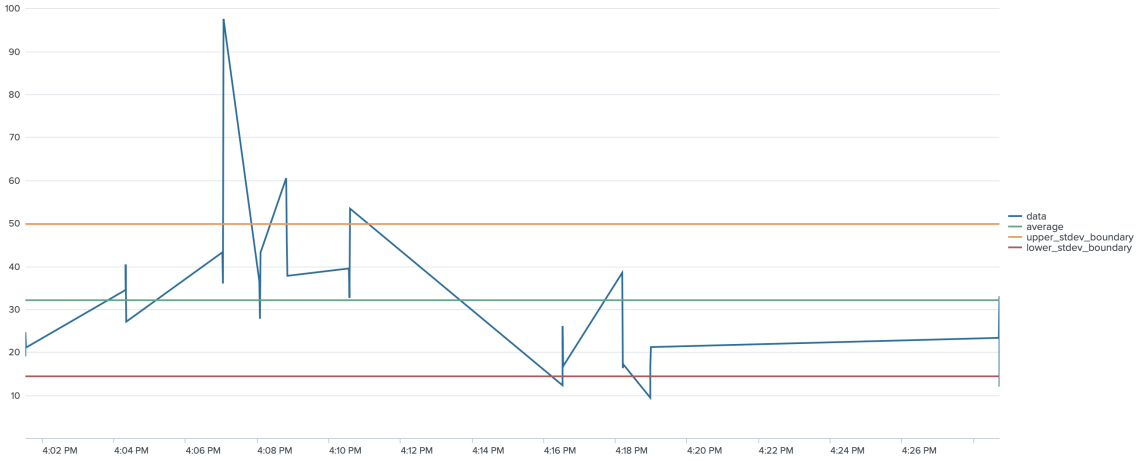
After the first test series, the adjustments that were done to the prototype lead to a drastic reduction of occurrences of the *Deflection* measurement. It was only logged twice while 140 taps were performed across the ten test runs (14 taps each). One was recorded during the 6^{th} and another during the 10^{th} run. After investigating the prototypes source code, a bug has been encountered, which prevented the logging of *Direction* events for swipe gestures.

The implementation bug leads to a broken connection between *Direction* and *LinearStrokeDevianceDirection*, but since one can infer from the scenario description that all swipes have to be executed from left to right for the gesture to work, *Direction* would have always been right. Therefore, the test series does not have to be repeated.

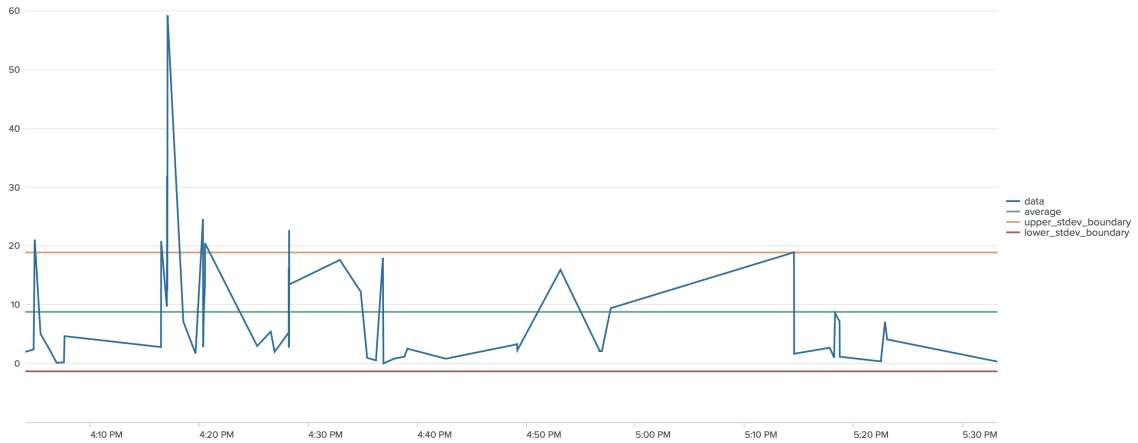
Since the *Direction* measurement was only triggered twice, no analysis is possible on these values.

This supports the already mentioned observation that it is quite challenging to deviate from the initial position when tapping enough to move the touchpoint without swiping (with low or medium finger pressure). An extensive set of subjects would need to perform tests to see whether the number of times the measurement was recorded might be a factor to be included in the template, but this cannot be done as part of this thesis. The technical restriction and the lack of subjects are enough to stop the analysis of this measurement.

LinearStrokeDevianceDirection has already been discussed above. A measurement that might be interesting with or without the direction of deviance from a straight line is the distance of deviance. Figure 7.13(a) shows the deviance in points where the point at every change of direction of the blue line indicates the average value of one of the ten test runs. The other three straight horizontal lines show the overall average and the upper and lower boundary of the standard deviation. Figure 7.13(b) shows the same analysis for Subject 1 of the first series. The picture is a lot more homogenously for Subject: The gesture lines were drawn straighter and with fewer



(a) Second Test Series (Single Test Run = Point of Change of Direction of the Blue Line)



(b) Subject 1 of the First Test Series

Figure 7.13: Linear Stroke Deviance And Lines for Average, Upper and Lower Boundaries of Standard Deviation *Note: y-axes not normalized*

outliers from the average. This does reduce the standard deviation automatically due to the fixed lowest values of zero.

As mentioned before, *StrokeDistance* has been added to the set of measurements after the first test series to support *StrokeSpeed*. They will be analyzed and set in relation below.

The tables in Figures 7.14(a) and 7.14(b) are both sorted by source / test run to make the comparison easier. The higher the numbers for average and for standard deviation are compared to the other rows, the darker the shade of red highlights a specific table cell. Focussing first on the *StrokeDistance*, one can read from the table that the average distance differs hugely between the test runs. There is also a big

source	average	stdev
d1r1.json	256.1375509826339	12.858167246070641
d1r10.json	288.9789021941161	18.38589486604421
d1r2.json	483.9627289134589	50.32882408676869
d1r3.json	722.2593711802136	41.32615532013126
d1r4.json	633.5690320972377	16.21873457005917
d1r5.json	610.2900977069398	25.009897266496818
d1r6.json	631.9910585860584	10.47987599687478
d1r7.json	318.9954552708263	55.527914375000464
d1r8.json	328.8943015562424	70.7446648571272
d1r9.json	309.67964287461353	9.625652371541207

(a) Stroke Distance Stats per Test Run

source	average	stdev
d1r1.json	2.3491759779482746	0.27707618959955754
d1r10.json	3.132633705085079	0.3218202208601536
d1r2.json	1.2699699050615505	0.042899868508531676
d1r3.json	1.52392287902123	0.0655771626732241
d1r4.json	1.6315184209143432	0.33368407831536695
d1r5.json	1.6600412683563366	0.44868707658487306
d1r6.json	2.052028998187529	0.22208313282517006
d1r7.json	4.121217201297251	0.7599030541934678
d1r8.json	3.5767738899295622	0.8506267316596566
d1r9.json	3.561315660653688	0.5693363623332751

(b) Stroke Speed Stats per Test Run

Figure 7.14: Intra-user variation: Stroke Speed and Stroke Distance in Comparison

gap between the higher numbers and the lower numbers without a clear tendency towards either group. The first run and last four ones show average values between 256 and 329 points. Run three to six lie between 610 and 722 pt. The third run's average resides almost precisely in the center of the gap between the two ranges.

There is an explanation for the higher number in the middle of the series: The numbers are based on three consecutively executed gestures, that are at the same time, the final functional steps to execute before finishing the test run. Right after these three gestures comes the navigation to the app's settings, followed by the log export. The condition for completing a test run successfully is that all steps are strictly followed, or the run needs to be repeated. Unlike other swipe gestures used to, e.g., scroll through a list that exceeds the viewport, these gestures can fail the purpose. It seems that the gestures are supposed to mimic browsing a physical book. If one does not set enough momentum in motion, the page would bounce back

to its original location instead of making space for the other page. Therefore the swipe needs to be performed rapidly if short, or the page would need to be dragged wide enough not to bounce back. Since the second and the third test run failed during one of the three swipe gestures, it led to 'fear' of failure, and the gestures were executed overly cautious during the next test runs. After a few repetitions, the self-confidence was regained, and the gestures were executed as in the first run. For all tests, runs hold one similarity: The standard deviation relatively low. This indicates that muscle memory might be another factor in the execution of similar gestures. A further test would be necessary to follow this theory. Series with similar test setups would be needed but containing more variations of swipe gestures.

As explained above, speed is a crucial factor in succeeding in this type of gesture. The logged data also reflect it: Swipes that have been performed more rapidly were not dragged as wide as the slower moving ones. Regardless of the necessity, it might also be a personal preference to either perform the gesture quickly or drag longer but more slowly. The correlation between the two measurements makes it a useful metric for further development.

LinearStrokeDeviance was looked at from a high-level perspective with the outcome that only the maximum values but not averages and standard deviation did differ between the subjects. The above-explained phenomenon also includes changes in the deviance from the line between the swipe gestures' start and endpoint (Figure 7.15). The longer the linear distance, the higher does the average deviance from that line get. If this can be seen as a habit / personal practice can be decided after test series with more subjects, since none of the subjects did perform e.e. long, yet straight gestures.

The high-level analysis that has been performed on *TapDuration*, continues to show the same trend for all ten test runs of the second test series. The difference between the subject lies in both the average as well as the standard deviation. The average of averages of the second test series equals $66251\mu s$, which is close to the average of the 2nd subject's average of $59198\mu s$. The same goes for the standard deviation: An average of $25427\mu s$ for the second test series vs. $22682\mu s$ during the 2nd subject's run of the first test series (reminder: same person).

Subject 1's average lies at 25.528,5, which is even more fare away from the other person's average than the other person's standard deviation. The standard deviation of Subject 1 is more than twice as high as for the second subject. *TapDuration* can

source	average	stdev
d1r1.json	21.551670787607076	2.862030986438086
d1r10.json	22.745368342407883	10.5366286335825
d1r2.json	33.97431588618762	6.6741466410817045
d1r3.json	58.91438664950442	33.65441431723761
d1r4.json	35.79789980454099	7.703822756707143
d1r5.json	47.94219699193709	11.552931299868684
d1r6.json	41.814312224201835	10.58908490160208
d1r7.json	18.335565331120232	7.040565308276108
d1r8.json	23.946359825300608	12.58576350556599
d1r9.json	15.691566477403393	5.938272906079419

Figure 7.15: Intra-User Differences: Linear Stroke Deviance

source	average	stdev
d1r1.json	39671.11307142857	19777.87516400718
d1r10.json	52400.738000000005	20629.595425183466
d1r2.json	68011.33628571428	23197.95676016912
d1r3.json	82011.26485714286	20019.713043861728
d1r4.json	79665.74399999999	58790.34759440562
d1r5.json	62284.07457142857	18138.885287590856
d1r6.json	63119.52971428571	12469.460223845326
d1r7.json	66811.77364285714	28110.138113035184
d1r8.json	75923.15771428573	25908.00688277016
d1r9.json	72613.003	27230.843236541554
s1_iso_ts.json	37196.30892838876	50380.02495772236
s2.json	59198.45108653846	22682.83591491521

Figure 7.16: Tap Duration of All Sources

therefore be considered a promising factor that can be made part of the biometric template.

7.7 Interpretation of the Analysis

All aspects of the particular measurements that have been made part of this thesis' prototype were discussed in the previous section. How the measurements have been categorized after the analysis can be read from Table 7.2.

Measurement	Status after analysis
TouchForce	excluded
TouchRadius	excluded
Deflection (for tap gestures)	further analysis
LinearStrokeDeviance	further analysis
LinearStrokeDevianceDirection	further analysis
StrokeDistance	further analysis
StrokeSpeed	further analysis
TapDuration	further analysis
RelativeTapDevianceDirection	conditionally accepted
Tilt	conditionally accepted
Deflection (for swipe gestures)	accepted
Motion	accepted

Table 7.2: Status of measurements after analysis

TouchForce and *TouchRadius* are too specific to a small set of devices that support these measurements since a physical sensor is needed to generate the data. These two measurements have been eliminated from futures steps. All six measurements labeled with 'further analysis', need test series with more subjects and ideally more diverse test cases to conclude their usefulness.

Measuring the *Deflection* pursues two purposes. It was triggered rarely for tap events, yet manual debugging showed that it is possible to deviate significantly from the initial touchpoint without performing a swipe when tapping with medium or high force. This aspect of the *Deflection* measurement needs further analysis. However, it is already accepted to provide crucial information to other measurements like *LinearStrokeDevianceDirection*.

RelativeTapDevianceDirection and *Tilt* proved that they could be different enough between subjects to exist as part of the biometric template. It would still be useful to rely on data that has been collected from a broader range of subjects. Hence, they are labelled as '**conditionally** accepted'.

Motion is the only metric that has been analyzed in regards to its accuracy since it is supposed to be used independently from the subject's behavior. *Motion* should be correlated to a set of fixed rules where one example could be phrased like: "If the device is in motion for a longer period without being locked, this could be a sign of theft." The numbers logged during the test runs prove the reliability; it can be accepted for further development.

Only two out of eleven measurements were categorized as unsuitable. Few correlations were identified on top of the raw measurements, and also time can be used - not only for the *Motion* measurement - to support the decision making.

On top of the insights gained out of the available data, a lot more data is needed to find a final set of characteristics for a reliably working biometric system. The measurements are also not exhausting the full capabilities of what can be tracked.

One of the untouched areas is to let keystroke dynamics flow into the user's behavioral fingerprint. Unfortunately, it is impossible to add a gesture recognizer directly to the virtual keyboard, where the actual gestures could be measured. It can still be used for 'traditional' typing biometrics by creating models of how single words are put together through interception of the keystroke events.[20, Chapter 8: Keystroke Dynamics Authentication]

The analysis of the measurements that have been put in place did reveal that at least the numeric measurements can be approached similarly. However, it is not as simple as calculating min, max, average, and standard deviation and make decisions right away. Post-processing might be need depending on the particular measurement. One example is the *Tilt*, which does not make much sense in combination with the *Motion* being 'stationary'. The data was filtered for values above $0.1rad$ or below $-0.1rad$ to compensate for time intervals where the device laid flat on a desk. Such correlations are significant to get right. The context plays a vital role in every single measurement. Measurements were set in context during the analysis to explain the logged values. To interpret the measurements, it might be necessary to know all the possible scenarios in which a particular gesture needs to be executed and what effects these scenarios would have on the outcome. One example of that is the existence of several scrollable areas within the UI. If the user is performing short swipe gestures several times in a row, but suddenly changes to executing the gestures entirely differently, it could indicate a change in behavior. It could as well mean that the scrollable areas were relatively narrow before and would only move slightly before reaching the other end, and the user switched to using more expansive fields where one could scroll a lot more content.

One could naively implement the rest of the prototype and test how it performs based on the knowledge gained during this thesis, fine-tune thresholds and plot the results on ROC curved. However, the suggestion is instead to analyze how the context-awareness would influence the results' interpretation first.

This does also leads to the next step: Fostering a score calculation strategy. The most naive strategy might be to count violations of previously calculated stats. On top of that, one could enrich the logic by not relying on the one measurement that seems to changes, but considering the user session invalid as soon as the measurements would changes, e.g., three metrics.

The analysis revealed that if measurements can be correlated with other measurements, change values together. A study might be needed to see what to do if both measurements change, but in a way that the correlation is still consistent. Does it benefit the decision making at all to monitor the correlation between measurements? Would metrics be required to formalize every correlation and see these metrics as coequal to other measurements regarding the resulting score?

The missing context awareness and the uncertainty around some of the measurements' usefulness lead to excluding the score calculation and decision-making from the prototype.

The analysis has revealed that not only the context plays a role in how consistent a person is in the behavior, but also other factors like the mood, or perhaps pressure put on the user by customers, can influence the behavior. Other assumptions taken in the beginning, regarding the necessity of a purpose that is supposed to be pursued by individual measurements, does not seem to have much weight in determining the usefulness of the particular measurement. It is more the ability to differentiate between people when measuring account. As an example, *LinearStrokeDevianceDirection* was intended to gain insight around whether the user is left- or right-handed. Since the actual purpose is to distinguish between individuals, one does not precisely need to know which group of people they belong to. It is just necessary to know whether the person is performing gestures more with one hand or the other when controlling the particular application.

8 Future Steps

As can be inferred from the analysis, the most apparent action for the future is the assembly of various test scenarios, which need to be executed by a representative amount of subjects. If the eight of the raw measurements and the combinations of compatible measurements still prove to hold after these test series, they could lead to a solid set of metrics for the biometric template. To create a matcher module for the system, proper strategies to know how to proceed after creating the genuine user's template need to be created first.

Suggestions for matching might be to determine the genuine user's template's readiness by setting a time limit or a minimum threshold per measurements could be implemented. Afterward, it could go over into a phase where other samples would be constructed in the same way as the template, and as soon as the sample has the same size as the template, the could be compared. Another approach might be to match single measurements against the template's values and create thresholds for too different results and a maximum number of times the thresholds were exceeded. Finally, one could create a copy of the original template, enrich it with the new data over time, and compare how the two datasets diverge.

An optional yet promising idea is to use machine learning to make sense of the raw measurements. *Apple Inc.* provides the *Core ML* library to create models and make predictions based on the data that has been used for training. Besides the standard models for image, text, and sound, it is also possible to create custom models.[21] This interface could be used for training a model with the measurements and metrics discussed in this thesis. The advantage might be that it eases the comparison and decision making by abstracting away the complicated logic. A concern is the loss of control regarding the setting of thresholds and how decisions are taken.

After the actions described above have been performed, the quality of the approach that has been developed during this thesis could be measured as described in chapter 2.3.

9 Beacons

In comparison to biometrics as a continuously utilized tool for user-re-authentication, hardware beacons will be discussed, and their advantages and disadvantages will be listed on a theoretical level. Hardware beacons that use the iBeacon protocol within the *Apple Inc.* ecosystem are devices that transmit their Universally Unique Identifier (UUID) and very few other bytes of data over BLE. This enables application developers to generate UUIDs specific to their app, that can be deployed on the beacons and trigger events, like location-specific advertisements, when the device running the app is close to the beacon. The identifiers do not necessarily have to be uniquely distributed to single devices, but they can also be transmitted by devices that are programmed to have the same purpose. Since the transmission of a static UUID is a one-way communication, with a message that does not change, it can be easily spoofed and replicated. This type of attack is called *piggybacking*. The described use case does not involve any severe abuse case scenario. Therefore it might not be necessary to protect the authenticity and integrity of the message. By default, no protection layer is part of the iBeacon protocol or added on top.

In this form, it would render beacons useless for a scenario in which the beacon is used as a second authentication factor from time to time.

To secure the communications, beacon vendors have come up with different varieties of more or less the same principle: Rotation of UUIDs.

Examples of two different implementations of beacon security will be briefly described in the following paragraphs.

Estimote[22] uses UUID rotation in combination with a web service that they provide to prove the authenticity of the currently broadcasted identifier and, therefore, its origin. They also require the beacon owner to connect to their cloud to change the device's UUID. This is a measure to prevent the repurposing of the beacon.

BlueCats, as the second sample vendor, explains its approach of handling iBeacon security in its developer documentation[23] as the following:

> This advertisement must be decoded by the BlueCats SDK, via a proprietary multi-step decryption procedure and assures that only authorized apps (utilizing the BlueCats SDK and having been granted permission) can interpret beacons and trigger micro-location based actions.

Knowing that solutions exist to prevent *piggybacking*, the two theoretical approaches of using the technology for re-authentication can be explained:

Firstly, beacons could be installed stationary at points where airline agents would operate the most. The main advantage of this setup is that multiple agents in the area could be secured with a single beacon, which positively impacts the hardware costs. Furthermore, it equalizes the disadvantage of battery-powered beacons: Even though BLE enables the beacons to act in an extreme power saving way, where devices can last for months and years without a battery replacement, intentionally firing huge loads of requests to update the devices UUID will reduce the battery life, which can be seen as one possible attack vector. A stationary installed beacon, of course, binds the agents to the same location. This takes away the main advantage of mobile devices - They become somehow stationary again. To allow the agents to move freely, the application needs to support rather large proximity to the sender before triggering the sign-out mechanism. In this case, stolen devices can, therefore, continue to be used without being immediately noticed.

Secondly, users could carry personal beacons. Since the user would have the beacon handy anytime, sign-out could be triggered as soon as the beacon would leave the 'near' range (< 3m). This can be considered highly secure since agents should instantly notice when someone else uses their devices right next to them. Agents can as well move freely while carrying the beacon. While airlines are keen on sharing the *iPads* amongst multiple airlines to keep the costs low, there might also be a pool of beacons, where the users pick one when the shift starts. Therefore the application needs to be able to pair with an arbitrary device easily. Since the beacons are pocket-sized items, they could be forgotten at the end of the shift and might be carried home. Therefore, airlines should have beacons in spare as well.

Portable Bluetooth beacons cost from very few Euros for no-name products to over 25€[24]. With the security aspect in mind, high-quality devices should be preferred, where UUID rotation is built in.

Both approaches have one thing in common: Resilience needs to be considered. The application should still work if no beacon is available. The user needs to have the option to log into the application without a beacon if none is present. Offering this fallback option to users introduces the potential risk of always choosing the easier setup before the shifts: Not taking advantage of the beacons at all.

For B2B applications that are sold to various customers, vendor lock-in is another topic that needs to be kept in mind. To take advantage of beacons that are capable of handling security mechanisms, very specific development is necessary, like, for example, the integration of the *BlueCats* Software Development Kit (SDK). This leads to custom development to support different beacon vendors or to forcing the customers to buy beacons from a certain vendor.

10 Face Recognition

In contrast to behavioral biometrics on which this thesis's prototype is built, face recognition relies on physical characteristics.

Face recognition describes the process of using an optical sensor to capture the image of a person, find the face within the image, locate fiducial points, like chin, nose, and eyes, to create a mathematical representation of the face and match it against other acquired samples.[3, chapter 3]

As described in Chapter 2.4, *iPadOS* does natively support fingerprint (*Touch ID*) and face (*Face ID*) recognition depending on the *iPad* model. Both are restricted in the number of templates that can be enrolled. Furthermore, the matching requires the user's active participation and blocks the UI for that time. This chapter will briefly describe different approaches that could be used to utilize face recognition to verify the user's identity periodically. Finally, the approach that has been chosen for the comparison to the other techniques will be highlighted.

Several commercial solutions [1] on the market offer the services to detect or verify faces. The APIs are fed with images or videos and return mainly confidence scores, but might also provide additional information, like recognition of the gender, age, or emotions. The products differ in the way they are integrated into applications, their capabilities, and pricing models. Examples for commercial products are the *Microsoft Azure Face API* as an API solution or *Face++* offering their service as SDK and as API.[25, 26]

Facial recognition can be split into two main actions: Detecting faces on an image, followed by the actual recognition. While the first step is independent of individual identities, the second step involves the extraction of characteristics belonging to the particular person's face.

[1] The particular solutions were not evaluated. The information is based on what vendors claim on their respective websites.

A suggestion for light-weight usage of physical biometrics that benefits the data privacy aspect is to continually check for the existence of a face inside the *iPad* camera's viewport. Face detection alone does neither force the association to a real person nor the persistent storage of biometric information on the device. The idea is to base the confidence of the session's validity on the assumption that it is hardly possible for an unauthorized person to gain unnoticed access to a device without a period in which the device did not recognize a face in its viewport. If the *iPad* was put aside and therefore no face is pointing towards the device, the likelihood grows that someone else could pick it up. This approach's main drawback is that it is impossible to detect the unauthorized person anymore after the recognition of theft has failed.

The more rigid approach would be to rely on face verification where the reference template would be stored during login, and snapshots would periodically be matched against it. It would decrease the number of necessary checks drastically since the first approach might require a check every few seconds to be reliable. For this proposal, it would be enough to test, for example, every few minutes, since an attacker could not do much harm in such a short period. If a commercial library/product is used to verify the face, then this might have a massive, positive impact on the costs, depending on the pricing model. Where some vendors seem to sell the SDK for a specific price, API usage might be priced based on transaction volume.

Apart from the commercial products, open-source prototypes and libraries are available, which utilize *Swift's Core ML* library to base the decision making on machine learning. Examples for open source projects are Omar M'Haimdat's *face_ai* and Bohdan Mihiliev's *SFaceCompare*; both available on Github.[27, 28]

Core ML offers an interface to train standard model types, such as images or natural language.[21] The models can be trained on a *Mac* and, when mature enough, put into the target application, or even updated while in use. The advantage is that the data does never have to leave the device. The models are still unprotected from being analyzed outside the app since packaged apps can be unpacked, and the model can be inspected. This is a significant difference to *Face ID*, where the template is stored on a separate processor outside the main chip with strong cryptography applied. The so-called Secure Enclave includes an Advanced Encryption Standard (AES) crypto engine and an own key management system.[2, chapter 13.1.2] To protect the user's biometric traits, it would be required to encrypt the model and

templates. This implies decryption during runtime and, therefore, also the necessity of storing private keys somewhere safe.

Besides security concerns, privacy concerns need to be raised for both of the approaches, since the actual user might have the choice to opt into the additional feature of continuous re-authentication, but customers (in case of a B2B application) or uninvolved third parties could also get captures and process without consent.

Despite the difficulties concerning privacy and security, face recognition has proven over the years that it can be a reliable utility to secure assets. This chapter's second flavor will be compared to the thesis' other approaches within the next chapter.

11 Comparison of the Individual Approaches

The central part of this thesis, meaning the utilization of behavioral biometrics with finger gestures and the mobile device's attitude, will be compared to two alternative approaches. As a first alternative, beacons were introduced in Chapter 9 to be used as a physical authentication factor that can be continuously pinged to monitor the desired constant proximity from the user's mobile device to the beacon. Chapter 10 described the facial recognition as a utility to verify that the person on a picture taken during login is still the same as on snapshots taken in intervals while the user session is running.

While quality in terms of effectiveness in only known for facial verification, the comparison will be made under the assumption that the other two theoretical approaches might perform similarly well. Furthermore, the comparison presumes that biometrics are implemented so that the data does not leave the device for processing (e.g., with machine learning support by using the *Core ML* library).

This chapter does not aim at finding the best approach overall, but to highlight the similarities, dissimilarities, advantages, and disadvantages, to give the reader the option of choosing which approach would fit best into his or her concrete use-case scenario. The following long list of criteria comprehensively covers many different aspects to support the decision-making.

Environmental Circumstances As long as the user's device is capable of operating, external circumstances, like low light, would not influence the behavioral biometric system's reliability. Face recognition relies on the quality of the photo that is acquired by the device's camera. The beacon is transmitting a signal at a specific frequency range. If there is a massive overlap with other transmissions at a time, this could break the system.

Proximity All approaches are relatively immune to short periods in which the device is put aside, and the user may move away. As long as the device is not controlled while the legitimate user is not nearby, the beacon approach could still keep the session alive. It has, therefore, no disadvantage compared to the other approaches.

Benchmark The suggestion for both the behavioral biometrics and face verification is to rely on the native *iPadOS Core ML* library. Computation of machine learning algorithms and the storage of the trained machine learning models impact the device's resources. Newer devices with the 'Neural Engine' support on so-called 'Bionic' processors are optimized for hardware-accelerated Artificial Intelligence (AI). [29] The impact on app size is still expected to be much higher when well trained AI models are made part of the packaged app, and processing might also be more complex than simple checks for the proximity to a beacon. How high the impact is on the battery life of the mobile devices cannot be predicted precisely. However, since higher consumption of resources leads to higher energy needs, the beacon approach should be superior in this category.

Sample Acquisition Time Taking a photo or receiving signals from a beacon can be achieved without noticeable delays. In contrast to that, behavioral analysis can only be performed after observing the behavior over at least a few minutes of active usage. The amount of interactions between user and device is even more important in that matter. An attacker could, for example, pick a device up to just execute a very specific but harmful scenario. Before the behavioral analysis detects enough interactions to create a profile out of it, the device might already be back in the hands of the legitimate user.

Function Creep All three strategies could contain undesired behavior. The beacon system needs to be designed so that only one particular beacon could keep a session alive at a time. This particular beacon should also only be used singly for one device. The transmitted message should not be guessable and re-transmittable by another device.

The behavior can be mimicked to a certain extent. Therefore, it is necessary to add many behavioral biometrics criteria to make it robust enough against such attacks.

The behavior logged is still resistant against the biometric function creep when working with physical traits. Possession of a template, which was, for example, extracted by hacking a device, does not make it as easily reusable as with physical biometrics.

Maturity Face verification is the only approach that is proven to work reliably. As discussed, the comparison is made under the assumption that also the theoretical approaches would perform well, yet for face verification, it is a fact.

Complexity Face verification and behavioral biometrics would both rely on well trained AI models. This introduced the overhead of preparing a model that is mature enough to give reliable scores. Most of the complex logic is abstracted away by passing it to *Core ML*'s neural network. Decision making still needs to be coded and thresholds defined and fine-tuned. Moreover, behavioral biometrics needs lots of input parameters from different measurements and a custom machine learning model to hold the data.

Beacon signals, on the other hand, only need to be picked up and checked for integrity.

Resilience As indicated earlier, at least beacon usage and face verification could be impacted by external circumstances, which could be overcome by giving the option to the user to disable the feature temporarily.

Setup Costs Once implemented, the biometrics approaches should not introduce costs for setup.

Beacons would need to be purchased. Licenses might need to be acquired in case of API based integrity checks.

Maintenenance Costs AI models might needs updates from time to time to improve reliability, which includes implementation and rollout costs.

Beacons could get lost or damaged over time, so the maintenance costs need to cover purchases of replacements.

User Acceptance Biometrics implies infringement of privacy might, which might not be accepted by every user. This holds especially for physical biometrics traits. Beacon usage should be as convenient as carrying an additional key on the keyring and might be preferred by users.

Security All approaches have their weaknesses regarding security. The iBeacon protocol has been designed without a security layer on top. Therefore, the particular beacon vendor has to be chosen carefully to, as already pointed out earlier, guarantee that the beacons' integrity can be checked.

Biometric systems need to consider the safe storage of the collected data (infrastructure attacks). They are technically vulnerable to, for example, impersonation attacks where a malicious user could potentially use a picture of a face instead of presenting the actual trait.

Scalability The flexibility to respond to increasing user numbers is bound to the pool of available hardware when talking about the beacon approach. Not only the availability of the hardware plays a role, but also the costs associated with new purchases.

Biometrics can be easily rolled out alongside the application and is therefore independent of the number of users.

12 Summary

This thesis aimed to create a concept of protecting a running session on a mobile device from sudden user changes. The main focus lied on evaluating the usefulness of a fingerprint-like user profile of measurements extracted from how a subject controls an *iPad*. As a counterproposal, the usage of beacons as physical items that need to be kept near the mobile device was discussed. The third option, which was analyzed on a theoretical level, is the repetitive biometric verification of the user's face with snapshots of the device's camera.

The topic was introduced by the explanation of the theoretical basics of biometric systems. The three technologies that have either been used or at least described the most have been introduced afterward.

All measurements to be analyzed regarding the behavioral fingerprint have been described to lay the prototype's foundation. Furthermore, the prototype's architecture was substantiated by explaining how the different components interact with each other.

Regarding the implementation's description, the focus was put on the basic mechanisms on which the prototype is built. The different steps from gathering the data to making it available for manual analysis or aggregation and further processing are covered. The central part of this thesis was the analysis of the usefulness of measurements that have been chosen in the beginning. While the extent to which all individual measurements differ between subjects is still partially unclear due to the non-representative number of subjects, it is a first step towards a working system that can make applications in charge of a critical mass of sensitive/confidential data more secure during running sessions.

12.1 Outlook

The *Amadeus Airport Companion App* was the inspiration for this thesis and the application into which the prototype was built. The access control system in front of the application narrows the possible test subjects down to a small set of *Amadeus* employees and some *Lufthansa Group* ground agents on airports in Frankfurt, Munich, and Vienna. Due to the COVID-19 pandemic impact on the travel industry, the audience was reduced again, and physical contact was restricted. It was planned to have around ten subjects available for test series. Ultimately, the number was reduced to two: An *Amadeus* QA engineer and the author of this thesis.

Apart from the number of subjects, the first test series required extensive test data setup before each run and was therefore suboptimally chosen. The measurements that are classified as promising after the thesis' analysis need to be tested on more subjects to conclusively gain a robust set of metrics that can be used to finalize the prototype.

12.2 Conclusion

The majority of the measurements that have been initially selected show promise in being useful for a production-grade system, but further verification is warranted to support this claim. Machine learning is a promising technology to be used in combination with either the behavioral biometrics or facial verification, especially since the data processing can happen solely on the mobile device.

Three approaches have been discussed as part of this thesis, and the advantages and disadvantages have been highlighted. All approaches introduce high costs for either software development or on the hardware side. Applications that process enough valuable information to benefit from these additional investments, besides strong authentication and short session lifespans, are, as pointed out earlier, in this thesis' focus.

None of the strategies seem to be universally superior to others. The security requirements of the target system should indicate which solution fits best. For example, when there is a strong need for immediate detection of security breaches, then a concept based on facial recognition has a high chance of being effective. As the

technology is known and nowadays reliable enough can be seen with systems like *Apple's Face ID*. It has the major advantage of being able to instantly acquire new samples where the behavioral biometrics approach needs to wait until enough interaction with the device happened to compute a new sample. Hence, it can also react to short switches between users. Simplicity is in favor of the beacon approach, but the chances of vendor lock-in are high since the necessary signal integrity protection mostly relies on proprietary techniques. Finding suitable measurements for fingerprinting individual human behavior profiles is possible. However, the programmatic interpretation can be challenging, and the resulting quality of such a system is unclear and might not suit depending on the actual security requirements.

It is, all in all, possible to add another good protection layer on the client-side that does not disturb the regular user-device interaction.

Bibliography

[1] Amadeus: *About us*, 2020. https://corporate.amadeus.com/en/about-us, visited on 10 Sep, 2020.

[2] Eckert, Claudia: *IT-Sicherheit*, volume 10. De Gruyter, 2018, ISBN 978-3-11-055158-7.

[3] Jain, Anil K., Arun A. Ross, and Karthik Nandakumar: *Introduction to Biometrics*. Springer, 2011, ISBN 978-0-387-77326-1.

[4] Oezkaya, Necla and Seref Sagiroglu: *An intelligent face features generation system from fingerprints*. 17:185, January 2009. https://www.researchgate.net/figure/A-generic-biometric-system_fig6_228552534, visited on 17 Jul, 2020.

[5] Khandelwal, Chhaya, Ranjan Maheshwari, and U.B. Shinde: *Review Paper on Applications of Principal Component Analysis in Multimodal Biometrics System*. Procedia Computer Science, 92:483, December 2016.

[6] Narkhede, Sarang: *Understanding AUC - ROC Curve*, 2018. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5, visited on 29 Jul, 2020.

[7] Informationstechnik, Bundesamt für Sicherheit in der: *BSI Evaluierung biometrischer Systeme Fingerabdrucktechnologien - BioFinger*, 2014. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/BioFinger/BioFinger_I_I_pdf.pdf?__blob=publicationFile, visited on 30 Jul, 2020.

[8] Wikipedia: *Touch ID*, 2020. https://en.wikipedia.org/wiki/Touch_ID, visited on 31 Jul, 2020.

[9] Wikipedia: *Face ID*, 2020. https://en.wikipedia.org/wiki/Face_ID, visited on 31 Jul, 2020.

[10] Maltoni, Davide, Dario Maio, Anil K. Jain, and Salil Prabhakar: *Handbook of Fingerprint Recognition*. Springer, 2009, ISBN 978-1-84882-254-2.

[11] Apple: *Swift*, 2020. https://swift.org/about/, visited on 16 Jun, 2020.

[12] Gruenderszene.de: *LEXIKON: iBeacon*, 2019. https://www.gruenderszene.de/lexikon/begriffe/ibeacons?interstitial, visited on 16 Jun, 2020.

[13] Wikipedia: *Bluetooth low energy beacon*, 2020. https://en.wikipedia.org/wiki/Bluetooth_low_energy_beacon, visited on 16 Jun, 2020.

[14] Apple: *Getting Started with iBeacon*, 2014. https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf, visited on 16 Jun, 2020.

[15] Inc, Splunk: *About Splunk*, 2020. https://www.splunk.com/en_us/about-splunk.html, visited on 29 Aug, 2020.

[16] Inc, Splunk: *Splunk 7.x Fundamentals 1 (eLearning)*, 2020. https://www.splunk.com/en_us/training/free-courses/splunk-fundamentals-1.html, visited on 29 Aug, 2020.

[17] Apple: *About the Gesture Recognizer State Machine*, 2020. https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/implementing_a_custom_gesture_recognizer/about_the_gesture_recognizer_state_machine, visited on 8 Sep, 2020.

[18] Imperva and Edward Robers: *The Evolution of Hi-Def Fingerprinting in Bot Mitigation*, 2018. https://www.imperva.com/blog/the-evolution-of-hi-def-fingerprinting-in-bot-mitigation/, visited on 16 Jun, 2020.

[19] Apple: *3D Touch*, 2020. https://developer.apple.com/design/human-interface-guidelines/ios/user-interaction/3d-touch/, visited on 2 Sep, 2020.

[20] Giot, Romain, Mohamad El-Abed, and Christophe Rosenberger: *Biometrics*, pages 157–177. June 2011, ISBN 978-953-307-618-8.

[21] Apple: *Core ML*, 2020. https://developer.apple.com/documentation/coreml, visited on 13 Sep, 2020.

[22] Wojtek Borowicz, Estimote: *iBeacon security: understanding the risks*, 2020. https://www.imperva.com/blog/the-evolution-of-hi-def-fingerprinting-in-bot-mitigation/, visited on 1 Sep, 2020.

[23] BlueCats: *Developer Documentation: Static Beacon Modes*, unknown. https://developer.bluecats.com/documentation/libraries/mobileSDK/usage/static-beacon-modes, visited on 16 Aug, 2020.

[24] Estimote: *Buy Proximity Beacons for $99*, 2020. https://order.estimote.com/buy/proximity-devkit-2018, visited on 16 Aug, 2020.

[25] Azure, Microsoft: *Facial Recognition*, 2020. https://azure.microsoft.com/en-us/services/cognitive-services/face/, visited on 19 Sep, 2020.

[26] Face++: *Face++ Cognitive Services*, 2020. https://www.faceplusplus.com/, visited on 18 Sep, 2020.

[27] Mhaimdat, Omar: *Github: omarmhaimdat/face_ai*, 2019. https://github.com/omarmhaimdat/face_ai, visited on 18 Sep, 2020.

[28] Mihiliev, Bohdan: *Github: BohdanNikoletti/SFaceCompare*, 2019. https://github.com/BohdanNikoletti/SFaceCompare, visited on 18 Sep, 2020.

[29] Gruenderszene.de: *Jameson Toole*, 2018. https://heartbeat.fritz.ai/ios-12-core-ml-benchmarks-b7a79811aac1, visited on 19 Sep, 2020.

[30] Apple: *Vision Framework*, 2020. https://developer.apple.com/documentation/vision, visited on 18 Jan, 2020.

List of Figures

List of Tables

Glossary

AES Advanced Encryption Standard. 66

AI Artificial Intelligence. 69, 70

API Application Programming Interface. 3, 4, 16, 23, 26, 29, 38, 46, 65, 66, 70

AUC Area Under the Curve. 15

B2B Business to Business. 8, 9, 64, 67

BLE Bluetooth Low Energy. 18, 62, 63

DCS Departure Control System. 3, 4, 8

ISO International Organization for Standardization. 43

JSON JavaScript Object Notation. 26, 31, 33, 34, 43

MFA Multi Factor Authentication. 9

PII Personally Identifiable Information. 9

PoC Proof of Concept. 25, 26, 29, 30, 36

QA Quality Assurance. 34, 73

ROC Receiver Operating Characteristics. 15, 59

SDK Software Development Kit. 64, 65, 66

UI User Interface. 16, 22, 28, 29, 45, 52, 59, 65

UUID Universally Unique Identifier. 62, 63, 64

UX User Experience. 9, 40

A Source Code

```
1  `Splunk Queries`
2
3  `linechart with average, and stdev lines`
4  source="d1*" event="strokedistance"
5  | eventstats avg(data) as average, stdev(data) as stdev
6  | eval upper_stdev_boundary = average+stdev
7  | eval lower_stdev_boundary = average-stdev
8  | table _time  data average upper_stdev_boundary
       lower_stdev_boundary
9
10 `non-numeric measurements`
11 source="s2.json"
12 | where match(data, "^[a-zA-Z]+")
13 | top data by event
14
15 `general numeric measurements stats by event`
16 source="s2.json"
17 | where match(data, "\d")
18 | stats min(data) as "Minimum", max(data) as "Maximum",
       avg(data) as "Average", stdev(data) as "Standard
       Deviantion", count by event
19
20 `timecart visualization for single measurement`
21 source="d1r1.json" event="Tilt"
22 | timechart avg(data) span=1s
23
24 `query used for pie chart visualization of a single
       measurement`
25 source="d1r1.json"
26 event="relativetapdeviancedirection"
27 | top data
28
```

```
29 `numeric stats by source`
30 source="d1*" event="strokedistance"
31 | stats avg(data) as average, stdev(data) as stdev  by source
32 | table source  average stdev
33
34
35 `numeric stats by source for visualization`
36 source="d1*" event="tapduration"
37 | stats avg(data) as average, stdev(data) as stdev by source
38 | eval upper_stdev_boundary = average+stdev
39 | eval lower_stdev_boundary = average-stdev
40 | table source  average upper_stdev_boundary
      lower_stdev_boundary
41
42 `combination of deflection and linearstrokedeviancedirection`
43 source="s1_iso_ts.json" (event="deflection" OR
      event="linearstrokedeviancedirection")
44 | eval linear_data=if(match(_raw,
      "LinearStrokeDevianceDirection"), data, null())
45 | eval deflection_data=if(match(_raw, "Deflection"), data,
      null())
46 | transaction startswith="LinearStrokeDevianceDirection"
      endswith="Deflection"
47 | eval combined_data=linear_data+" - "+deflection_data
48 | top combined_data limit=16
49
50 `average deviance grouped by direction`
51 source="s1_iso_ts.json" (event="linearstrokedeviance" OR
      event="deflection")
52 | eval deflection_data=if(match(_raw, "Deflection"), data,
      null())
53 | eval deviance_data=if(match(_raw, "LinearStrokeDeviance"),
      data, null())
54 | transaction startswith="Deflection"
      endswith="LinearStrokeDeviance"
55 | stats avg(deviance_data) by deflection_data
```

Theses

Continuously validating the user's identity benefits the security of critical applications.

Behavioral biometrics is one helpful utility to validate the user's authenticity.

The key to the robustness of a biometric system built on top of gesture execution and other behavioral characteristics is a broad set of different measurements.

The evolution of computational power, storage, and battery efficiency on mobile devices supports the development of complex systems, as needed for the score calculation of a biometric system.

Academic Honesty Declaration

I, Moritz Herbert, declare that the attached assignment is all my work and all references contained within it have been correctly cited and documented on the reference list.

(Moritz Herbert)
Niederdorfelden – September 25, 2020