Gathering Contextual Information About Users of Mobile Applications

Matej Rajtár

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2019

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, June 14, 2019

Matej Rajtár

Acknowledgements

I would like to thank my supervisor, Prof. Dr. Michael Haller, whose guidance and expert suggestions have been invaluable and made this work possible. I also wish to extend my gratitude to the team at bluesource - mobile solutions, especially to DI Karin Scheiblhofer for her great patience and insightful advice throughout all stages of the work. In addition, I would like to express a special thanks to my family for their constant help and support.

Contents

Abstract

Research shows that distraction from sound, movements, mobility, lighting, and different ways of interaction with devices in real-life conditions impact user's performance in terms of usability measures, such as error rate and delay. However, while in-vitro usability evaluations do not provide authentic data about user's context and often face shortcomings of an unrealistic laboratory setting, in-situ field studies with actual users can be challenging to budget or schedule and may lead to only limited results. Additionally, such analyses do not truly capture the dynamic character of constantly varying environments and might mislead mobile application development. Therefore, the main goal is to find a solution that can be used continuously after deployment, to gather data for evaluation of the contextual parameters, and subsequently provide insights to mobile application developers. The aim is to leverage algorithms from the field of ubiquitous computing, in combination with smartphone sensors that are present on most devices, in order to generate contextual datasets, helping developers to learn more about the typical user's setting, as well as the way of holding and interacting with the device. As a result, mobile applications should be easier to learn, easier to use, and aesthetically more pleasing, leading to their higher usability and acceptance. Based on latest work in ubiquitous computing, algorithms from the following research domains were selected: activity, handedness, indoor/outdoor condition, sound levels, weather detection.

Kurzfassung

Die Forschung zeigt, dass Ablenkung durch Geräusche, Bewegungen, Mobilität, Beleuchtung und verschiedene Interaktionen mit Geräten unter realen Bedingungen die Leistung des Benutzers im Bezug auf Usability-Aufzeichnungen wie Fehlerrate oder Verzögerung beeinflusst. In-vitro-Usability-Versuche liefern keine authentischen Daten zum Benutzerkontext und weisen häufig Mängel einer unrealistischen Laborumgebung auf. In-situ-Feldstudien mit echten Benutzern sind jedoch schwierig zu finanzieren und zu planen und liefern eventuell nur begrenzte Ergebnisse. Darüber hinaus erfassen solche Analysen den dynamischen Charakter sich stetig ändernder Umgebungen nicht wirklich und können die Entwicklung mobiler Anwendungen in die falsche Richtung lenken. Das Haupziel besteht daher darin, eine Lösung zu finden, die nach der Veröffentlichung verwendet werden kann, um Daten für die Auswertung von Kontextparametern zu sammeln und anschließend Entwicklern mobiler Anwendungen einen Einblick in den Benutzerkontext zu gewähren. Ziel ist es, Algorithmen aus dem Bereich des Ubiquitären Computings in Kombination mit Smartphonesensoren, die auf den meisten Geräten vorhanden sind, zu nutzen, um kontextbezogene Datensätze zu generieren und Entwicklern dabei zu helfen, mehr über die typischen Benutzerumgebungen und die Art des Haltens und der Interaktion mit dem Gerät herauszufinden. Infolgedessen sollten mobile Anwendungen leichter zu erlernen, einfacher zu verwenden und ästhetisch ansprechender sein, was zu einer höheren Benutzerfreundlichkeit und Akzeptanz führt. Basierend auf aktuellen Arbeiten im Bereich Ubiquitäres Computing wurden Algorithmen aus dem folgenden Forschungsbereichen ausgewählt: Aktivität, Händigkeit, Innen- und Außenbedingungen, Schallpegel, Wettererkennung.

Chapter 1

Introduction

1.1 Motivation

Since the beginning of the era of computers they have been used in diverse environments for different tasks. However, it has not been until the paradigm of mobile computing that one device would be constantly changing the context of use. With the mobile platform currently reported to account for approximately a half of the global market share, based on the internet traffic [85], the effect of context on device usage appears to be a more prominent interface usability factor than ever before. As the research on this topic shows that typical usability measures such as error rate and delay are strongly influenced by context in real-life conditions [24, 33, 50], Tsiaousis et al. explain that it is the both task and distractions concurrently demanding the limited user resources, which may diminish user performance and thus usability [70]. For instance, according to the study by Lin et al. [39], when on the move, visual resources are split between interacting with the mobile devices and maintaining awareness of the surrounding environment. Besides that, the user interaction with mobile devices can be also negatively affected by a vast number of other contextual factors [59], generally coined by Sears et al. as *situational impairments* [66]. Usage situations vary, and so the general mobile applications ought to be designed with their use in various circumstances in mind [25]. Also the ISO organization describes the usability as *the extend to which a product can be used with effectiveness* (number/percentage of completed tasks within allotted time, number of errors)*, efficiency* (time to complete a task) *and satisfaction* (subjective user attitude) *in a specified context of use* [27]. But despite of the fact that the analysis of usability has become a standard part of mobile development, the situational impairments are often misleadingly not taken into consideration during UX evaluations. While user studies in a laboratory (in-vitro) face shortcomings of an unrealistic setting by definition, field (in-situ) studies are challenging to budget and schedule, complicate data collection, and reduce experimental control [50].

The evolution of smartphones has seen a significant increase in the amount of embedded sensors, as can be observed from the clear upwards trend in Figure 1.1. Current mobile devices typically offer a wide range of built-in detection components, utilized already numerous times in the past in order to create novel and useful mobile applications [36]. Mainstream smartphones are commonly equipped with a majority of the following:

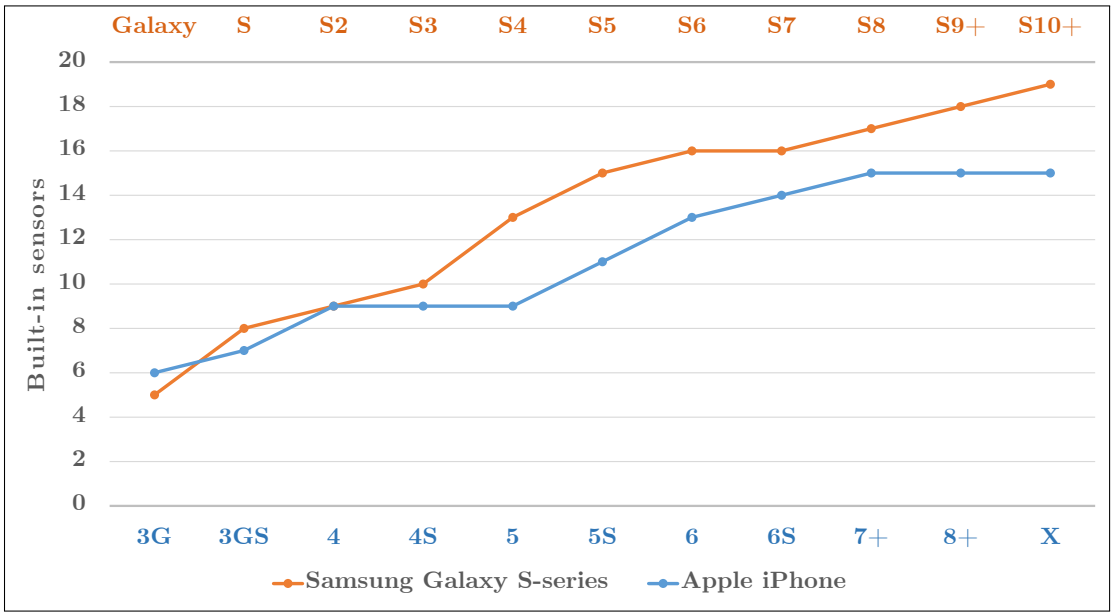


Figure 1.1: Number of sensors in smartphones in selected flagship smartphone models produced by Samsung and Apple, ordered by release date (2009-2019).

accelerometer, gyroscope, microphone, image sensor, GPS, light sensor, magnetometer, temperature, proximity, and pressure sensors. These can be very effective tools to understand one's context [25], which can in turn help developers to make mobile applications easier to learn and easier to use. This is important because given the limited space on the saturated consumer market, usability can be even considered as one of the most important attributes that determine the success of a mobile application [43].

Although context abstraction from raw sensor data has been subject to research in the field of ubiquitous computing for many years, the main purpose typically concerned health monitoring [31], alternative input development [21, 41], or performance improvements [11, 19] and ignored the advantages in the field of UX design. One solution to the problem with context not taken into the account could be therefore to gather data for evaluation also from real users in post-release phases, and using the aforementioned wide variety of sensors included in current mobile devices to detect context. The ultimate aim is then to combine the data of traditional usability evaluation techniques and the data generated by sensing, to be able to improve results of the both supervised and unsupervised usability evaluations. Similarly, as in the related research by Thurnher et al. [69], who used a few external sensors attached to an early smartphone device in order to measure context, it is argued that context information measured by modern smartphones' internal sensors adds value to usability evaluations, as it provides many additional facts to explain the behavioral data. The novelty of the resulting project code implementation lies thus not on its detection abilities individually, but instead on their collective use to abstract situational context.

1.2 Goals and Contributions

This section specifies the set goals in the order as they were accomplished.

Firstly, to talk about context and be able to cherry-pick detection algorithms from the current state-of-art ubiquitous computing field, the initial goal inherently had to be the identification of context types. Moreover, due to the fact that various authors approach already the mere context definition differently and produce separate classifications, the integral part of this process was also further literature review to compose a comparative definition.

Secondly, the research fields that abstract meaning from sensor data, such as ubiquitous computing, usually offer multiple algorithms with miscellaneous resulting success rates and computational requirements. Because of the limitations that stem from mobile computing, the goal was to find such solutions that prioritize efficiency over the highest possible accuracy, and that for every selected kind of context.

Thirdly, as the scientific papers on the topics were mainly published without provided testable software implementations of the discussed algorithms and often even excluding the pseudo-code (or in case of machine learning the trained models), the goal was a re-implementation of the described formulas into a reusable Java library for Android, called Context Information Toolkit (CIT). One of the requirements was that apart from the algorithms it would also contain export of the sensed data into a CSV file format for later analysis.

The implementation process was further followed by the empirical part of the research and involved two phases: the pilot testing and the user study. In the first phase, a preliminary usability test of a sample application was conducted, so to test the functionality of the created library, as well as to describe key usability issues of the app for later reference. Afterwards, the experiment involved two small-sample groups both performing the same task under different conditions. The aim was to showcase results possible to achieve with the library and gather the evaluation data.

Lastly, with the objective to present the generated data spreadsheets in a visual form, the D3 JavaScript library was used to develop an interactive exploratory visualization.

1.3 Challenges

A major challenge was to recognize the context parameters, which are not only confirmed by substantial and relevant research to have an influence on user's behaviour, but at the same time such that can be detected by embedded smartphone sensors and are meaningful in terms of studying usability. For example, while social contexts, e.g. nearby people are known to considerably influence the user's behavior, this detection is hardly feasible in terms of privacy once used in production of mainstream applications. On the other hand, inferring temporal context does not pose a threat to privacy, but in most cases neither straightforwardly influences human performance.

Another challenge was to correctly understand and transform into code the referenced works on ubiquitous computing. It had happened multiple times throughout the project that the explanations contained in the sources were too scarce to advance with the re-implementation, or the resulting code did not show to be nearly as accurate as declared, and thus had to be later replaced with another approach.

Challenging was also the development of an easy-to-read visualization consisting of high-dimensional time-stamped data, as only its joined implication describes the overall context. In addition, another requirement was to display at the same time the user-flow through the test scenario, hence it was also necessary to cope with the screen-size limitations.

1.4 Outline

This chapter briefly introduces the theoretical background, motivation, challenges and goals of this thesis. Chapter 2 explains the main terms used throughout the text, states the research problem and summarizes different types of context. The related research projects and commercial products, as well as their comparison to this work is contained in Chapter 3. In Chapter 4 the developed concept is presented in detail and further shows the ways of its implementation into other projects. Chapter 5 describes the two separately performed evaluations and reports all the discovered results. Finally, Chapter 6 explains the limitations of the presented system, proposes future work and concludes the thesis.

Chapter 2

Usability in Contexts

This chapter defines the fundamental terminology, gives an overview of how other relevant research refers to user's context, and identifies its types across classifications.

2.1 Comparative Definition

The topic with only three words, 'context of use' implies different things to different researchers. While the term is often simply abbreviated to 'context', the ISO organization offers the following definition [27]:

> *. . . users, tasks, equipment (hardware, software, and materials), and the physical and social environments in which a product is used.*

As a systematic approach Jumisko-Pyykkö and Vainio [29] distinguish nearly the identical categories but in addition mention the temporal and information aspect, as well as the properties of mobile context (level of magnitude, dynamism, pattern, and typical combinations). Väätäjä [72] specified numerous subcategories of each of these components and eventually characterized the context of use in mobile work. This helps to better understand the core principles behind the term, while still focusing on its concrete implications. Using their design sketch, Savio and Braiterman [61] place the information context into the category of existing user's goals and further outline the aspect of internet and cellular connection's speed, reliability, and cost. Other, less specific definitions include: any information that may influence the user [32] or characterize their situation [2, 69], a set of interaction-influencing conditions or user states [8], visual, auditory, and social cues under a form of distractions [71], and the circumstances under which the activity takes place [57].

2.2 Influences on Usability

The influence of context on user's performance has been established by research already numerous times, but each scientific study naturally focuses just on a subset of context types. Tsiaousis et al. [70] observed effectiveness and efficiency in using a mobile website to be influenced by sound semantics, level of light, and motion of nearby objects. Hummel et al. [23] measured decreased performance of users in terms of higher error rates

and delays once exposed to similar environmental changes, but also additionally tested for temperature and humidity fluctuations with inconclusive results. Larsen et al. [36] did conclude that the contextual information offers valuable insights with implications in mobile UX, while investigating the behavior in regards to the proximity to other users (social aspect) and location. The results of Barnard et al. [8] indicated that the contextual changes in motion and lighting had a strong impact on behavior and performance, measured in task completion times, and error rates. Lettner and Holzmann [37] found that the context-awareness on mobile phones is an important aspect, which should be taken into account, when evaluating usability. Bevan et al. [10] pointed out that any context may influence or even change the usability of a product, mostly if these conditions were not considered during the product design and evaluation. Kaasinen [30] claims that device's mobility itself introduces a great variability to the context of use, which might even change during a session.

Other studies verified the difference between testing in the actual content-rich environments versus in idealized laboratory conditions. In their research, Pedell et al. [52] argue that context of use has to be accounted for in the evaluations, as the mobile use involves much more than the interaction of a user with a device. They discovered that the metadata sensed in realistic conditions provide a deeper understanding of usability issues and help to detect problems that did not even appear during test in a laboratory. The need for realistic user studies is further directly supported by Brewster [13], who noted that a realistic environment can change the interaction, which must be considered when designing and testing mobile devices. Furthermore, Hertzum [22] shows that conducting unsupervised field studies is cheap and does not require much preparation.

The limitation of a study that omits distractions arises mainly from the negative effect of distractions on performance and information processing [6]. Attention span and short-term memory (working memory) are constrained by cognitive load [5], which refers to *the total amount of mental activity imposed on working memory at an instance in time* [49]. An example of this can be the work of Coursaris et al. [16], who proved that distractions had a significant negative impact on the efficiency and effectiveness of mobile device use.

Since all of the context types may have an impact on the final UX of mobile systems, this information should be attractive to app creators, in order to improve the usability on mobile devices. What is more, the definition of context-aware computing says that it is the ability of a mobile user's applications to discover and also react to changes in the environment they are situated in [63]. Therefore, all of the identified contexts should be thought of also in a broader spectrum than just usability evaluation, for example additionally offering developers the advantage of triggering a relevant change in the interface. Finally, the collected contextual data can enable stakeholders to learn more about the users together with finding out more about the quality of the interface. Based on the previously stated state-of-art research on the topic, the following research questions have been formulated:

1) Which ubiquitous computing methods can identify contextual states of a smartphone user that are helpful at uncovering potential usability issues?
2) Is there a correlation between user-error rate and time-on-task in relation to the distraction of a user, detected by measured contextual parameters?

2.3 Supportive Usability Metrics

To make use of the measured contextual data and put it into perspective of simple interactions, it is essential to simultaneously record or log the happening interaction. But seeing the whole chronological (ideally in average correct) user path through the task scenario might not be enough. To discover problems with an interface it is important to identify the incorrect patterns known as cognitive friction. Table 2.1 offers an insight into those, seen in the data recorded during this project, including suggested solutions.

This friction occurs once a user is confronted with an interface or affordance that seems to be intuitive but delivers unexpected results, which will in turn cause frustration and impair the user experience [82]. These issues appear in session data generally as random actions coming from the user. Lettner et al. [38] claim that these dynamic metrics can identify unused components, misleading localization, and navigation glitches.

Friction pattern – user's actions	Possible solution
Visits and immediately leaves a screen due to a navigational error.	Improve navigation and the app structure, icons or their labeling.
Performs a gesture with no assigned action.	Assign a suitable action to the gesture.
Misses UI-Element they intended to touch.	Increase the element's touch area.
Repeatedly touches UI-element despite of no assigned functionality.	Assign a suitable functionality or correct the provided affordances.
Switches back and forth between two views multiple times.	Try to place all the needed information on one screen or allow for scrolling.
Touches a UI-element accidentally.	Increase the spacing between elements.
Long inactivity between actions.	Decrease the interface's complexity.

Table 2.1: Friction patterns observed during the empirical parts of the project.

2.4 Types of Context

As the Table 2.2 presents, during this project overall seven different types of context have been identified that have, in stronger or weaker ways, an impact on the final usability in mobile computing. In total, these types consist of 31 components, which group related concepts together for easier comprehension. There is undoubtedly an overlap between some of the categories, such as people surrounding the user (bystanders), who could be apart from social aspect further considered also as an influence from the physical surroundings. However, in these edge cases the ultimate classification was made based on the strongest distraction impact. For instance, person standing by a user causes larger cognitive dissonance when engaging in a conversation with this user, rather than merely as a physical object in the environment. Furthermore, the column 'Example' aims to provide a non-exhaustive list of properties, to which the respective components can equal. The last column 'Detection from' shows literature review results and contains some potential ways of detecting specific contextual data describing the situation, in that the interaction takes place. In case of the physical type, these context assumptions are detectable exclusively via device sensors, which is the main interest of this thesis.

Type	Component	Example	Detection from
User	Identity	Status, job, personality, age	Registration
	Goal	Entertainment, communication, work	User flow
	Attention	Continuous, partial, full, intermittent	Delay
	Motivation	Intrinsic (curiosity), extrinsic (reward)	Result
	Skills	Technology experience, multi-tasking	Questionnaire
Task	Complexity	Number of steps, decision difficulty	(App-dependent)
	Regularity	Daily, yearly, during other activity	Usage history
	Interdependence	Conditioned by registering first	(App-dependent)
	Urgency	Security-code valid for 2 minutes	(App-dependent)
Technical	Hardware	Device model recency, camera quality, display resolution, available resources, current battery level	System API
	Software	OS, familiarity, capabilities	System API
	Connection	Speed, reliability, cell, internet	System API
	Carrier	Pricing model, services	System API
Physical	Activity	Sitting, standing, walking, running, driving, eating, shopping	Accelerometer, gyroscope, GPS
	Movements	Raising hand, standing up, walking stairs, jumping	Accelerometer, gyroscope
	Mobility	Traveling by vehicle, plane, skateboard	Accelerometer, gyroscope, GPS
	Grip	Holding/interacting hand, fingers	Touchscreen
	Lighting	Direct sun, shadow, night, artificial	Luminosity sen.
	Environment	Indoor, outdoor, spaciousness	Magnetometer, Cell sig., GPS, Luminosity
	Location	Spatial, geographical	GPS
	Sound	Ambient noise, music, talking	Microphone
	Artefacts	Physical objects, such as laptop	Camera
	Proximity	To artefacts, other people, charger	Proximity sen.
	Encumbrance	Carrying a bag, backpack, artefact	Accelerometer
	Weather	Temperature, humidity, wind speed, pressure, raining, snowing	Thermometer, GPS, Barometer
Social	Other users	Online, in virtual environments	(App-dependent)
	Bystanders	Strangers, researchers during a test	Bluetooth, Wifi
	Culture	Language, religion, law, etiquette	Location
Temporal	Duration	length of interaction, activity, delay	Logging
	Time	currently available, of a day, of a year	System API
Information	Knowledge	user name, password, search query	Input form

Table 2.2: Summary of all types of context of use discovered during the project, partially based on the works of Jumisko-Pyykkö and Vainio [29] and Väätäjä [72].

Chapter 3

Related Work

This chapter examines the related work in commercial and research domain. The software tools and frameworks were scrutinized in regards to the methods, which they use for collecting data that support investigating usability. Whereas some research projects do this by determining innovative contexts, the commercial tools tend to target the analysis and visualization of more traditional user statistics, such as conversion rates, A/B testing, and in-app behaviour.

3.1 Research Projects

3.1.1 AUToMAte

AUToMAte[1] (automated usability testing of mobile applications) is a logging toolkit for keeping track of context information and mobile device usage on Android devices [86]. It can gather the data without having access to any application's source code with the help of accessibility service on OS-level. The built-in contextual capabilities contain: tracking of light conditions surrounding the device via luminosity sensor, monitoring the device's battery status, and logging basic device information including the current screen rotation. However, the framework is published open-source and developed to be extendable, thus any additional feature can be added easily. In comparison to this work, it cannot be used as a code library for an existing application and neither work without permissions, which were originally intended only for the services directly benefiting user's with disabilities.

3.1.2 AWARE

Aware[2] is an open platform, developed by Ferreira et al. [18] for context measuring, inference, logging and sharing. As a feature-rich Android application, it enables an easy conduct of context-aware user studies. The collected data is stored locally in a database in case of local user studies and can be uploaded to a remote server for large-scale research. Optional plugins can be installed, making Aware extendable even by a user.

[1] http://mint-hagenberg.github.io/automate-documentation/
[2] http://www.awareframework.com

Unfortunately, most of them are in the current state outdated and not working on the recent Android versions until further update. From the contextual perspective, it claims to offer plugin modules for measuring ambient noise levels, detect weather conditions based on location data and recognize mode of transportation from Google Location API[3]. In contrast to the proposed CIT, this framework cannot be implemented into other apps to record user interactions, such as the order of opened screens and performed gestures. At the present time it neither supports computing interior/exterior condition, activity without location information, and handedness.

3.1.3 CenceMe

CenceMe[4] middleware, engineered by Miluzzo et al. [45], senses physical and social context and shares this information through network applications. Activity, mobility, indoor/outdoor, and conversation classifiers work together to detect high-level contexts. Its focus is partially on identifying user habits, such as gym and shop visits. These are subsequently kept as historical data for evaluation of life patterns. Another aim is to provide more texture to interpersonal communication by informing the nearby network users about self-presence. The data are captured locally and the recognition takes place either directly on the device or on remote back-end servers. The resulting estimated context is meant to be posted for friends on social media, hence is not used for usability evaluation.

3.1.4 CoConUT

CoConUT[5] [65] is an Android application for collecting frequency of interactions, as well as the mobile context during usability studies using sensor data. It tries to measure components from multiple context types, in order to abstract a more complex meaning. Regarding physical context it focuses on movement from GPS location, brightness detection with a luminosity sensor, and noise levels with the device's microphone. Temporal context is identified by time-stamps and social context by the number of nearby devices communicating via bluetooth connection. Since this framework aims to be used during experiments with present researchers, technical and task contexts are left to be recorded manually. Moreover, compared to the presented library, CoConUT is a standalone solution and cannot be used as a module for another application.

3.1.5 EgoSENSE

Framework developed by Milic and Stojanovic [44] is a context-aware service able to collect data from both built-in sensors (location, temperature, accelerometer), as well as external sensors (pulse, blood pressure). Selected events are presented as a notification and sent to the remote server for further processing. Instead of usability, EgoSENSE is particularly relevant for health care applications, as actions are designed to be dealt with in a safe and reliable way, simultaneously informing the user and other parties about a

[3] https://developers.google.com/maps/documentation/directions/intro
[4] http://cenceme.org/
[5] https://play.google.com/store/apps/details?id=at.ac.univie.cosy.coconut

potential alarming situation related to the user's health condition. The framework can be also used for non-invasive monitoring of everyday patient's life, offering doctors a more accurate and honest lifestyle overview.

3.1.6 Mobile Context Toolbox

The MCT [35], similarly as other works, derives higher-level user context from multiple sensors that provide low-level information. In order to observe users, while they are using mobile devices and applications in a real-world setting, it uses accelerometer, bluetooth, GPS, Wi-Fi and GSM technology. The sensor readings were compared to the ground truth, labelled single time manually by a user. This approach enabled the researchers to identify concrete locations, such as home or work and activity, eg. having dinner or working. The authors consider the main potential of the toolbox in self-management, health monitoring, and providing the collected information to other installed applications. The toolbox was created for Symbian OS, hence for older devices without a touchscreen, which is the biggest difference from this work.

3.1.7 UEProject

Contextual data collection framework by Kronbauer et al. [33] used accelerometer for detection of horizontal and vertical position, GPS for location, luminosity sensor to capture the environment's lighting, and microphone in order to find out the noise levels. Besides that, they also utilized the Experience Sampling Method (ESM), based on the collection of users' feelings through questions. The assessment of the combined data enabled the researchers to phrase more specific assumptions about the behaviour of the test users. This work, published in 2012, is regarding the primary intention possibly one of the most similar projects to the proposed library. However, the key difference is the level of context abstraction from the sensor data that each of the solutions aims for. While UEProject only logs the sensor outputs, our work uses intelligent ubiquitous algorithms to analyze and process the data further, assuming more complex contextual scenarios.

3.1.8 Where-How-What Am I Feeling (WHWAIF)

The work by Filho et al. [17] presents a toolkit for automatic unsupervised app evaluation. Apart from other functionality it also detects relevant user context information, such as moving/stationary status, location, weather conditions and data connection availability. Additionally, it also uses a camera sensor to infer facial expressions and abstract adverse and positive emotional events. This toolkit differs from the work described in this thesis mainly in the types of context it infers and in the complexity of the registered usability metrics. Whereas the Where-How-What Am I Feeling logs only tap interaction for simplicity, our aim is to capture all types of interactions, as well as time-on-task and time-on-screen.

3.2 Commercial Tools

There are numerous products on the market for mobile application analytics, although, these frameworks serve mainly as marketing tools. They focus either on user statistics, such as user growth, demographics, and behavioral analytics or commercial metrics, e.g. in-app purchases, conversion funnels, and engagement. Some offer displaying a combination of various data and might contain additional features, including crash statistics, traffic attribution, and sending push notifications. Several of these solutions aim to also automate usability tests, but typically ignore emotional feedback and user context.

3.2.1 Amplitude

Amplitude[6] is a product analytics for both web and mobile applications. The offered analysis of user behavior allows product owners to predict and visualize new user retention and engagement, and further breakdown the user base on smaller groups, based on actions taken. The clustered user profiles can be then provided with personalized customer experience, to optimize conversion rates and increase profits. Despite of providing not only statistical views, but also detailed interactions of every single user, the generated data do not contain contextual information.

3.2.2 Appsee

Appsee[7] is a real-time mobile app analytics that features user interaction recordings presented as video, aggregated touch heatmaps to provide visual data on how users interact with different parts of an app UI, as well as conversion funnels, action cohorts, and retention analytics. All these are meant to provide in-depth view on a mobile app user behavior. The authors promise the developers and marketers to get fundamentally different and personalized insights, since they have the opportunity to actually observe the user's actions remotely. However, this analysis is not supported by any contextual data.

3.2.3 Apptimize

Apptimize[8] is a mobile A/B testing and release management toolkit that allows the developers to try multiple versions of a mobile app, without the need to release all its versions to production. By comparing the conversion rates and other user statistics, the superior version can be later pushed to the whole user base. Similarly to other commercial frameworks, neither the analysis from Apptimize is placed in context, in which the users interact with different app versions.

[6] http://www.amplitude.com/
[7] http://www.appsee.com/
[8] http://www.apptimize.com/

3.2.4 Flurry Analytics

One of the aims of Flurry[9] is to offer information about audience perception and detect the app's user base. It delivers usage statistics, for example new users per week or average daily users. Besides the collection of demographic information, i.e. age or gender, the framework also tracks custom events to analyze user interaction behavior. These events can be triggered by any user action inside an app, but have to be implemented programmatically. An additional possibility is to track app crashes and identify the underlying issues to find the cause. Flurry Analytics is a powerful tool for improving user experience of mobile apps but it does not provide any way to collect contextual data.

3.2.5 Google Analytics

The analytical platform from Google[10] for both mobile and desktop advertisers, publishers and mobile app developers provides a tight integration with other products, such as AdWords, AdSense, Google Display Network and others. The list of features includes data collecting across websites, apps and internet-connected (IoT) devices, data access via mobile app, configurable APIs, email notifications, multiple levels of user access controls, and more. The platform contains a so called Google Context API, offering the developers seven context signals that are not primarily intended for improving application usability, but rather for context-aware app's, which may react to context changes. The signals include time, location, places (place type such as restaurant, based on location), beacons (nearby bluetooth and wifi devices), headphones (connection state), activity (low-power sensors), and weather (based on current time and location).

3.2.6 Localytics

Localytics[11] primarily focuses on measuring key metrics, such as new users, OS being used, drop off rates, geographic breakdown and collecting other similar datasets, used in mobile app marketing for increasing the engagement with a company. This solution offers targeting various marketing campaigns on different subgroups of recognized customer profiles. The ultimate intention is hence to help reaching a desired ROI by adequate market segmentation, rather than improving usability of mobile interfaces.

3.3 Discussion and Comparison

From the analysis of existing solutions (summarized in Table 3.1) there appears to be a market gap. In contrast to the available research work, the commercial platforms do not offer measuring of user context. The current state-of-art algorithms provided by ubiquitous computing research can help better explain the reasons behind a specific user behavior and yet they are not included in the solutions on the market. In terms of context measuring, the analytical software from Google appears to be the the most

[9] http://www.flurry.com
[10] http://www.google.com/intl/de/analytics/
[11] http://www.localytics.com/

	Implementable as a module	Context measuring	Interaction measuring	Focus on usability	Domain
AUToMAte	○	●	●	◐	research
AWARE	○	●	○	◐	research
Amplitude	●	○	●	●	commercial
Appsee	●	○	●	●	commercial
Apptimize	●	○	●	●	commercial
CenceMe	○	●	○	○	research
CoConUT	○	●	◐	●	research
EgoSENSE	●	●	○	○	research
Flurry	●	○	●	●	commercial
Google Analytics	●	◐	●	◐	commercial
Localytics	●	○	●	●	commercial
MCT	○	●	○	○	research
UEProject	◐	●	●	●	research
WHWAIF	●	●	◐	●	research
CIT (this work)	●	●	●	●	research

Table 3.1: Comparison of selected tools for mobile application analysis. The first column 'Implementable as a module' describes the property if a project can be used as a library, when developing a new standalone application. The property in column 'Interaction measuring' expresses if the project contains features for registering and logging user's actions, such as tapping on buttons, switching apps, and others. ● = yes, ◐ = partially, ○ = no

advanced from the offered services, but it currently lacks more advanced usage of the low-power sensors and measurements of usability.

On the other hand, while research until now presented apps for collecting usability metrics and contextual data of surroundings, these have to be installed in addition to the monitored app, in case the developer decides to collect context. The Context Information Toolkit provided in this thesis is a library, which enables an easy implementation into an existing application that is then ready to measure context right after installation. The scientific projects with the most similar aims are either outdated or do not compute higher-level contexts presented in this work, such as activity, interior/exterior condition, and handedness.

Chapter 4

Implementation

This chapter first introduces the proposed solution consisting of five different types of context detection algorithms. The technical design, as well as the main ideas behind algorithms are presented. Then, the usability metrics module and data exporting is briefly explained. Later, the chapter describes notable parts of the concept implementation and its integration into a demo application. Finally, the testing procedure is shown, involving a commercial app mobile-pocket.

4.1 Overview

A team of developers is working on an Android app that provides healthy cooking recipes. During market and user research they collected and included lots of features for the initial release, such as categorical searching or video instructions. To be successful on the market, they strive for good usability and previously conducted a user study that helped to shape the product. With the arrival to the Android application distribution service Play Store[1], they decided to implement an analytical framework, such as Amplitude (described in Section 3.2.1) with its behavioral platform, to help them understand their user base and target the marketing campaigns effectively. They now see the user flows, sessions, and usage of different features, but get inconsistent results. A part of their users seems to a have hard time navigating the interface, performing more repetitive actions, unresponsive gestures, and other input issues, in spite of that none of their test users showed such behavior. What their data do not show is that there are perhaps two user groups, which differ in their typical context. The satisfied user group might interact with the application mainly while cooking at home, with the device placed horizontally on the kitchen desk. The other group, unable to see and tap the too small navigational elements, may use the app in a shop or on their way there, looking for the ingredients to buy in hurry, distracted by the surrounding circumstances.

In such cases as in this considered example scenario, the developers have multiple options that will provide them with the needed information. Since the root of the issue is variability in the user group, it is possible to conduct field or diary studies and user interviews. However, the cheapest and the most permanent solution, considering also dynamic user habits, would be seeing the contextual data among the other analytics

[1] https://play.google.com/store/apps

in an overview. More specifically, knowing the user's mobility state, sound levels, and environment type, in addition to other metrics used in the described scenario, would enable the application developers distinguish the user groups and adapt app design.

Inspired by the existing use cases and lack of solutions (compared in Section 3.3), the following describes Context Information Toolkit (CIT)[2], a reusable context detection framework in form of an Android library, consisting of a variety of algorithms from ubiquitous computing. The context abstraction over raw data supports evaluation of user behavior, namely by providing an estimate of disturbance levels. Furthermore, the resulting code is supplementing interaction analytics with data describing the situation that can help to identify friction in regards to usability issues in an interface. As Barnard et al. [8] mention, context is a multifaceted construct by nature, and thus it cannot be defined by specific components independently, but must be considered as a whole. Therefore, the proposed concept strives to provide the developers with as variate context insights from device sensors as possible.

All of the context types shown in the Table 2.2 can be possibly detected by an intelligent algorithm. However, it is the category of physical context that is the most suitable for detection by smartphone sensors. These calculations influence the battery life differently, vary in terms of detection accuracy and in the affected levels of user's privacy. On these three properties were based also the main criteria when choosing from the range of available algorithms (ordered by priority):

1. The lowest attainable demands on computing power, subsequently the battery.
2. The highest achievable accuracy in an average session.
3. The lowest possible intrusion into the user's privacy.

The first requirement is crucial, since combined context detectors might rapidly increase app's computational requirements. This puts machine learning approaches into disadvantage due to their typical high battery consumption [26, 55], mostly during feature calculation [67]. Performing constant statistical analysis on real-time sensor data is computationally intensive and offloading the calculations to the cloud compromises user's privacy [78]. As a result, the methods depicted in this thesis avoid using machine learning, even though these approaches were also surveyed and are mentioned in the respective sections. The second requirement was evaluated by the detection success rates claimed in the source research, although the high accuracy was verified by observation during the development and pilot usability study. The third requirement was judged based on the permissions that Android considers as dangerous[3]. Nevertheless, this criteria had the lowest priority, since the level of intrusion into user's privacy strongly depends on the concrete mobile application, attitude of the developers, and usage situation. One of the CIT's use cases is to be employed during unsupervised usability studies, when the custom app version can be distributed among only a sample of acquainted test users. Thus, it is meaningful for the library to also contain algorithms that require access to sensors, which would not be potentially used with the whole user base, such as location and microphone. What is more, if the nature of the app requires permissions considered as dangerous anyways (for the purposes of other features included in the app), the access to this data can be reused by CIT, in order to improve the usability.

[2]The word 'cit' means 'a feeling' in Slovak language.

[3]https://developer.android.com/guide/topics/permissions/overview#permission-groups

As a compromise of the stated requirements, the produced code consists of the following five physical context detection components, profiting from available sensor data. The sensing of the current user's activity, in terms of its classification among multiple different states, which is based on the feedback provided from built-in accelerometer and gyroscope (Section 4.3). The detection of the hand used to interact with the device, taking an advantage of the performed scroll gestures on the touchscreen (Section 4.4). The calculation of whether the device is located in interior or exterior, combining magnetometer, cellular network, and luminosity sensor (Section 4.5). The noise detection measuring noise levels in decibel units utilizing microphone (Section 4.6) and weather detection assuming the temperature, humidity, and other conditions from location and time (Section 4.7). This unique combination of contextual factors provides versatile data to a wide spectrum of application developers and is further supported by a sixth component, offering to log user's actions (Section 4.8) for identification of friction patterns.

Consequently, as a first step, the envisioned workflow involves the developer implementing CIT into their Android application. It is to be decided, if the new app version will measure the context in a supervised or unsupervised usability evaluation, hence if it will store the log files locally or should upload them to a remote server, respectively. Once the data are collected, they can be visualized using the provided tool and inspected for the correlation of usability measures (such as user-error rate, time-on-task, or friction patterns) with the determined contextual information. Following the evaluation, UX teams may be able to answer questions including:

- How many users of the app are accomplishing a specific task when walking?
- What number of users performs unresponsive gestures when in noisy environment?
- What is the proportion of users that open the left menu drawer with left hand?
- What percentage of users interacts with the application outside when a strong light shining on the screen is detected, or when it is cold and raining?
- Many users are tapping on the white space and make navigational errors, but this behaviour was not observed in previous in-vitro usability studies. Are they possibly distracted by context? Does increasing the text size help reducing errors?

4.2 Technical Design

The structure of CIT as visualized in Figure 4.1 describes all existing classes. In the leftmost branch of Events, the low-level helper classes are defined. The main detection logic is located in the components named 'managers', as inspired by AUToMAte framework [86]. Each manager is extended by a corresponding `TransmissionEvent` class that serves as a shell for measured data, before passed to `FileExportManager` to be logged in CSV files. The data are apart from being logged also locally-broadcasted back to the application for processing. This twofold informing about the data enables the main app to directly react on the detected context, if necessary. The main class `AwareManager` instantiates all the other manager classes, deals with registering the sensor listeners and also receives the touchscreen events for re-routing them to the `HandManager`. Moreover, at the end of the life-cycle it takes care of cleaning up the registered listeners and shutting down the continuous data export to CSV files. The code is easily extendable by adding further managers and registering them in the main class. The library requires

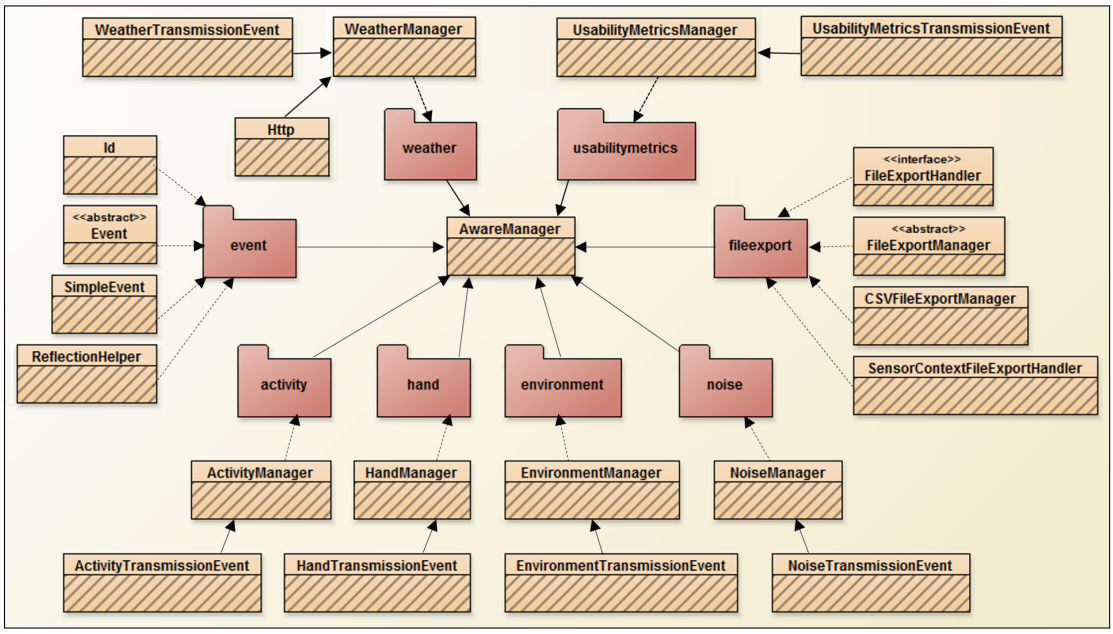


Figure 4.1: Context Information Toolkit composition, including in total 22 Java classes divided into eight separate parts. The arrows do not express dependencies, rather the hierarchical folder structure.

at minimum the Android smartphones with API 19 (4.4 KitKat) and later, which at the time of writing represents approximately 95.3% of all Android devices.[4]

The whole implementation requires only instantiating the main class `AwareManager` and starting the automatic process of recording contextual data. One single exception is the additional `UsabilityMetricsManager` class, which assumes a deeper connection with the parent application, if measuring also the supportive usability metrics (as further elaborated in Section 4.8).

4.3 Activity Detection

Coursaris et al. [16] found that among others, motion distractions have a significant negative impact on the perceived efficiency and effectiveness of mobile device use. For instance, walking has been found to have adverse effects on mobile interaction in completing typing and target acquisition tasks [20, 46] and cause deterioration in text legibility [60], cognitive performance, and reading comprehension [8]. The study by Ljungberg et al. [40] adds that mostly in combination with other visual or auditory distractions, walking will impact user's performance in a negative way.

Hence, it would be of considerable interest to mind the user activity when exploring measured usability data. Specifically, the following four elementary states (Figure 4.2) and the transitions between them were identified as the most common and relevant, regarding to everyday interaction with smartphones:

[4] https://developer.android.com/about/versions/android-4.4.html

Figure 4.2: Identified activity states. From left to right: smartphone placed on a table during the interaction, smartphone held in hand during standing, walking, and running.

- Stationary state, when the device is statically placed on a relatively flat surface during interaction.
- Still states, such as sitting and standing, when the user is holding the device in hand but is not additionally moving.
- Walking, in various paces but excluding stairs or steep surfaces.
- Running, as an extreme case that might be possibly interesting to the developers of sport applications.

Logging these states can serve as support during a field study, registering the data automatically instead of the researchers. More importantly, when used continuously after deployment, the data might provide insights, such as that a user made more errors and/or took longer to accomplish a task, which was related to their mobility state.

Activity recognition became a popular problem to be solved with machine learning. For example, Cakmakci and Laerhoven [73] leveraged an external accelerometer and Kwapisz et al. [34] embedded accelerometer sensor to detect various user motions, such as walking or climbing stairs. Lu et al. [42] used accelerometers to create a machine learning model able to distinguish between walking, race walking, and running condition. Filios et al. [19] additionally considered the states of sitting and lying. Reddy et al. [56] combined the motion detection with GPS receiver to reach an accuracy of 90% and higher, when deciding between stationary, walking, running, biking, or motorized transport states. Other researchers analyzed the data from different sensor combinations [67], device placements [15], and classifiers [76] to evaluate the impact on device's battery and detection accuracy. Moreover, Nikolic and Bierlaire [51] reviewed more then a dozen of transportation mode detection approaches, based on statistical tools or pattern recognition classification models. These works can have a good application in medicine, when monitoring the patients health, similarly as proposed by Ryder et al. [58]. However, for the purpose of usability studies, recognizing many various types of activities with a high precision is not a must, what is more, to protect user's privacy it might not even be desirable. Due to the use on mobile devices, more importance should be probably given to low battery consumption, so simpler algorithms that are using fast decision trees are more suitable for the needs of this thesis.

Using such approach, Schmidt et al. [64] tried to identify, in addition to user movement, whether a device is held in the hand, is on a table, or is in a suitcase. They employed basic signal processing, such as average (mean), standard deviation, quartile distance, base frequency, and first derivative to build a simple decision tree, based on thresholding. A similar but more robust approach can be seen in the work of Moreno et al. [47], who proposed an unsupervised decision tree based classification method to detect the user's postural actions. In their work, they claim to identify user states, such as sitting, standing, walking, and running by analyzing the data from a smartphone accelerometer sensor. This was also the method initially tried for this project. Their detection procedure starts by retrieving the three-axial acceleration data per a chosen sampling rate into a sliding window with an overlap of 50%. Afterwards, the current time frame of the sliding window is compared with the previous one, on values calculated from each of these time frames. These values, eventually resulting in a decision between states, are gained by multiple subsequent mathematical operations, such as normalization, cross-correlating the different accelerometer axes, Euclidean distance analysis, range search, integration, derivation, and mean absolute deviation (MAD). The authors claim that this signal processing is sufficient to differentiate between the individual states with an above 90% overall accuracy. However, apart from the mentioned mathematical foundation, they do not provide an application, code, and neither pseudo-code, which would be helpful to verify this claim. Despite of the efforts to re-implement this method and reproduce their experiment, in which one of the early smartphone models was used, the method did not show an accuracy better than 50%, when distinguishing between the calm states (sitting, standing) and active states (walking, running). This might have been caused by using a different device and other experimental conditions than those used in the original work, or by an incorrect code implementation. Nevertheless, it was necessary to replace the detection algorithm, in order to obtain a higher accuracy.

A subcategory of activity recognition are step detection algorithms. The step detection can serve as a hint, when to transition into a state of walking or running, thus can also be partially used to perform activity recognition. Kang et al. [31] used a phone's gyroscopic sensor to detect walking motion and count steps. Brajdic and Harle [12] evaluated common accelerometer-based walk detection and step counting algorithms and discovered that the best performing ones were calculating standard deviation and signal energy, despite of that none of the tested algorithms was reliable in all situations. This work implements one of such solutions that is based on signal energy, a Privacy Friendly Pedometer App [87] developed open-source by Karlsruhe Institute of Technology. The algorithmic logic, depicted in Algorithm 4.1 (for a simplified diagram see Figure A.1 in Appendix A), lies in computing the total acceleration value, in every given moment defined by the sensor's sampling rate. At first, this value is checked for its sign (direction on axis) to determine, whether it is the same as the previous value's sign, otherwise it represents a peak (change of direction). Later, it is tested for significance against a threshold value and for similarity in signal energy to the time of the previous valid step. Lastly, the step's frequency is investigated, as in the context of other identified steps it should not be unrealistically fast and neither slow, and at the same time it should be regular over time. In this way, the detected step event functions as a trigger, suggesting that the user's state transitioned into walking or running.

Algorithm 4.1: Step detection algorithm, invoked by an update from the accelerometer sensor. The result is given in the current number of identified steps.

```
1:  procedure StepDetection()
2:      sign, acc, time_prev, valid_steps ← 0.0
3:      signal_energy_threshold ← 0.65
4:      sign_prev ← 1.0
5:      acc_prev ← acc
6:      acc_x, acc_y, acc_z ← get current acceleration per axis
7:      acc ← acc_x + acc_y + acc_z
8:      sign ← signum(acc)
9:      if sign ≠ sign_prev then                    ▷ if equal, not a peak, keep waiting
10:         if acc > signal_energy_threshold then   ▷ if smaller, not significant
11:             acc_Δ ← |acc_prev − acc|
12:             if acc_prev · 0.8 < acc < acc_prev · 1.2 then   ▷ 80% and 120%
13:                 time ← get current time
14:                 if time_prev > 0 then
15:                     time_Δ ← get current time − time_prev   ▷ time between steps
16:                     if 500ms < time_Δ < 3000ms then
17:                         if time_Δ ±20% to previous time_Δ's then
18:                             if acc_Δ ±20% to previous acc_Δ's then
19:                                 valid_steps ← valid_steps + 1   ▷ increase step count
20:                                 add time_Δ and acc_Δ to data structure for comparison
21:                             end if
22:                         end if
23:                     end if
24:                     valid_steps ← 0
25:                 end if
26:                 sign_prev ← sign
27:                 time_prev ← time
28:                 acc_prev ← acc
29:             end if
30:         end if
31:     end if
32: end procedure
```

The complete activity detection in Algorithm 4.2 (for a simplified diagram see Figure A.2 in Appendix A) is then initiated with every update of accelerometer data from device's sensor in `ActivityManager` component. To decide between walking and running, a threshold of average acceleration magnitude is applied. The average value is first computed from the current sliding window per each of the three accelerometer axes. Then, the magnitude is calculated from those three resulting values by considering them as vector coordinates. The combination of registered steps with slow to medium acceleration magnitude implies the walking state and fast movements with steps mean the running state. In addition, to prevent frequent changes between walking and running, when the acceleration is similar to the threshold, the dynamic adjustment

Algorithm 4.2: Activity detection algorithm, invoked by an update from the accelerometer sensor. The result is one of the activity states.

```
1: procedure ActivityDetection()
2:     size ← 50
3:     sliding_window ← []                                  ▷ instantiate data structure
4:     stationary, flat, still, running ← 0.4, 0.3, 1.35, 7.5   ▷ decision thresholds
5:     acc_x, acc_y, acc_z ← get current acceleration per axis
6:     if sliding_window.size =< size then
7:         sliding_window add [acc_x, acc_y, acc_z]
8:         if sliding_window.size = size then
9:             for i ← 1 to sliding_window.size do
10:                sum ← sum + |sliding_window[i]|                ▷ per axis
11:            end for
12:        end if
13:    else
14:        sum ← sum − |sliding_window[0]|                        ▷ per axis
15:        remove sliding_window[0]                               ▷ per axis
16:        sliding_window add [acc_x, acc_y, acc_z]
17:        sum ← sum + |sliding_window[size − 1]|                 ▷ per axis
18:        magnitude ← Sqrt[(sum_x/size)^2 + (sum_y/size)^2 + (sum_z/size)^2]
19:        if magnitude < stationary then
20:            gyro_x, gyro_y ← get current gyroscope data per axis
21:            if g_x < flat and g_y < flat  and  for at least one second then
22:                send_intent(stationary)                        ▷ report the information
23:            end if
24:        else if magnitude < still then
25:            send_intent(still)                                 ▷ report the information
26:        else if steps detected then
27:            if magnitude < walking then
28:                send_intent(walking)                           ▷ report the information
29:                walking ← walking + 1                          ▷ increase walking threshold
30:            else
31:                send_intent(running)                           ▷ report the information
32:                walking ← walking - 1                          ▷ decrease walking threshold
33:            end if
34:        end if
35:    end if
36: end procedure
```

is applied. That means the threshold value is decreased after reaching the running state, and conversely increased with the walking state.

The identification of calmer situations is simpler. Gyroscope sensor is used to monitor current tilt on the two axes, which are parallel to the device's display. If the device is in a roughly horizontal orientation with the screen facing up, and simultaneously the current acceleration magnitude is approximately zero (within a sensor measurement

error), this is recognized, after a short delay, as a stationary state. The delay is implemented as a prevention against a rapid switching of the states, in cases the device is not placed on an entirely solid surface, or when a user slightly moves the device during touchscreen interaction. The still state is then chosen as no steps are detected and neither the acceleration values are close to zero. Because the transition to each of the states is conditioned by one or several fulfilled premises, the latest state is assumed until replaced by the next one. Once a state is determined, this is sent as a `ActivityTransmissionEvent` to `FileExportManager` and exported to CSV file. Furthermore, the state is also locally broadcasted to the parent app. Both of these are additionally supplemented with raw aggregated linear acceleration data for more insight, when viewing the activity data.

4.4 Handedness Detection

The size of contemporary smartphones makes them not well suitable for single handed usage. What is more, interface designers are often forced to place navigational elements to corners of the screen because of the visual design guidelines. Unfortunately, this in turn restricts the number of grips and gestures one can perform using single hand; to reach objects on the far side of the screen, one has to resort to using both hands. For instance, Wobbrock et al. [75] argue that the user's hand posture, while manipulating a mobile device, is a significant contextual factor affecting usability. To make usability research more aware of the interaction with the device, it might employ grip and handedness detection. When used with real application users, these algorithms might inform design teams, about better placement of screen elements and avoid the need for some A/B tests. During usability testing in a laboratory is generally all interaction recorded on camera, but handedness and grip detection might be still of help in field studies, enabling unobtrusive interaction monitoring and precise logging with time-stamps. Besides that, in terms of context-aware applications that are adjusting to the usage situation, apps may provide slightly different interface layouts dependent on the current grip.

Taylor and Bove [68] implemented Graspables, which used capacitive touch sensors to differentiate between numerous grips. Despite of a high detection accuracy, their system embodied a physical set of sensors attached to various objects and devices, which would not be feasible for a use outside of supervised testing conditions. Goel et al. leveraged built-in accelerometer, gyroscope, and touchscreen to propose GripSense, a system able to detect the user's interacting hand. Their approach used machine learning and in a controlled study they were able to achieve over 80% accuracy. Similarly, Löchtefeld et al. trained a model that is able to correctly distinguish one- and two-handed usage as well as left- and right hand during unlocking process, in over 98% of cases. However, Nelavelli et al. [48] show that a comparable task can be accomplished also without knowing any a priori information and statistical models. Instead of approaching the problem as a machine learning classification task, they use geometric curves during swipe events for hand identification. They claim correct results for 99.53% of their data, collected from swipes done by fingers on both-hands. Since the device is already collecting touchpoints, no additional battery consumption from sensors takes place, and thus this solution is superior to the other works based on machine learning. Additionally, the algorithm works within linear time complexity and so is well suitable for the implementation in this project. As an adaptation, the detection in this work

Algorithm 4.3: Handedness detection algorithm, invoked on screen touch and in case of a relevant swipe event and results in a binary decision meaning left or right hand.

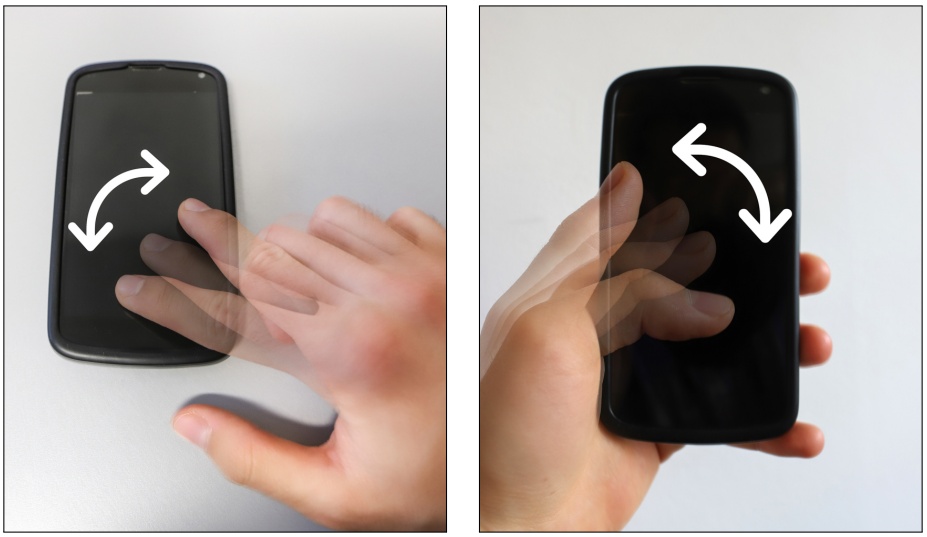
```
1:  procedure HandednessDetection()
2:      e ← get touchscreen event
3:      d ← 60                                  ▷ minimal relevant swipe distance in pixels
4:      switch e do
5:          case action_down                    ▷ finger touches the screen for the first time
6:              points ← initialize empty array
7:          case action_move                    ▷ finger moves across the screen
8:              points ← append touchpoint from e
9:          case action_up                      ▷ finger stops touching the screen
10:             n ← points.length                                   ▷ point count
11:             if n > 2 then                                       ▷ disregard taps
12:                 if points[0] − points[n] > d then               ▷ disregard too short swipes
13:                     c ← quadratic_regression(points)            ▷ curve's third coefficient
14:                     if c < 0 then
15:                         send_intent(Left hand)                  ▷ report the information
16:                     else
17:                         send_intent(Right hand)                 ▷ report the information
18:                     end if
19:                 end if
20:             end if
21: end procedure
```

receives the touchpoints already transformed according to the device's orientation in the hands of a user. That enables independence from the way the device is rotated and recognizes the interacting hand in portrait, as well as horizontal display mode.

The determination process exploiting touchscreen swipe events is based on multiple concepts, and therefore a number of steps needs to be performed, as seen in Algorithm 4.3 (for a simplified diagram refer to Figure A.3 in Appendix A). First of all, the swipe gesture itself needs to be identified and saved as a series of touchpoints. This can be easily done by extending existing Android gesture class `GestureDetectorCompat`[5]. In order to prevent false results, it is appropriate to continue with the detection process further, only if the swipe event consists of more than a certain number of touchpoints or covers longer distance than a threshold. Secondly, the gathered points have to be described by a characteristic polynomial function. To do this, it is possible to employ the curve fitting method called polynomial regression. Specifically, implementation of this method using the Least Squares algorithm is recommended for its low complexity. The basic idea of the least squares is to minimize the sum of the squares, defined as the square distance between a data point and the sought line.

When this has been done, the resulting curve already provides the information about handedness as follows. Due to the typical swipe shape, it can be assumed that the curve will certainly be parabolic, hence described by a standard equation. The coefficient

[5] https://developer.android.com/reference/android/support/v4/view/GestureDetectorCompat



(a) (b)

Figure 4.3: Performing swipe gesture with the right index finger (a) and the left thumb (b).

of the leading term (the term with the highest degree) in this equation can be either positive or negative, thus distinguishing between the two states. Finally, by comparing the orientation of the curve to the natural way of holding the phone, a conclusion can be made regarding the hand which is used to hold the smartphone.

This approach works due to the anatomy of a human hand, more specifically because of the way human fingers bend in its metacarpophalangeal joints[6] (see Figure 4.3). However, the method is limited to vertical scrolling. Once a user performs a swipe that is rotated to their orientation by more than 45° and so is identified by the application as a horizontal swipe (for example to open a side menu drawer), the detection can be no more considered truly valid. This limitation was empirically observed and arises from the fact that once a user performs a horizontal swipe, the imaginary arc drawn by a finger can randomly differ in the direction of its bend.

4.5 Environment Detection

4.5.1 Indoor and Outdoor Conditions

Environment type might influence user experience from multiple standpoints. The difference in lighting between indoor (Figure 4.4 (a)) and outdoor (Figure 4.4 (b)) places has an impact on readability of the fonts on displays. This directly affects the decision about font size used in an application, depending on a typical or major conditions in user's setting during interactions. Other example can be unavailable cell signal in (usually underground) indoor locations, such as metro or tunnels, or unavailable wireless internet connection in case of remote outdoor locations, which limits the smartphone's

[6] Joints situated between the bones in the hand at the palm with rounded ends, and the first finger bones shaped as caves.

(a) (b)

Figure 4.4: Indoor (a) and outdoor (b) environment.

capabilities. UX design teams that understand the average surroundings of their target group can make more informed conclusions, and therefore increase usability. Bisio et al. [11] exploited ultrasounds to detect indoor/outdoor condition, which despite of its high accuracy does not fit the purpose of this work due to its obtrusiveness. A more realistic approach appears to be using magnetometer, as well as cell signal and luminosity sensor [53, 74], in some cases supplemented also by other sensors such as Wi-Fi and GPS [3, 4, 54]. Naturally, this detection might be the most useful when used to gather data from continuous production, as during the supervised user studies this condition is clear.

Anagnostopoulos et al. [4] created an application for easy collection and tagging of environmental sensor data, later used for training of a statistical model. Having access to information from user's activity, pressure, lighting, accelerometer, magnetometer, and number of surrounding WiFi points, they were able to achieve almost 93% accuracy in distinguishing between the places the device was exposed to during the data collection. However, their model is stateful, meaning it is not able to adapt to unseen patterns, which makes it unsuitable for a use in unsupervised conditions and in this project. Ali et al. [3] proposed SenseIO framework that differentiates more complex environments, including rural and urban areas and underground locations. Similarly, Wang et al. [74] identify open outdoor, semi-outdoor, light indoor, and deep indoor conditions and Radu et al. [54] categorize campus area, city center, and residential (indoor) area. Although the methods in these works are an improvement to the first mentioned, since they are stateless and work in unfamiliar circumstances, they also entirely rely on previously collected datasets and subsequent computationally-intensive fine-grained pattern recognition. In comparison, Zhou et al. [53] demonstrate with IODetector that in order to determine a basic indoor/outdoor condition with a sufficient accuracy, no pre-collected data is necessary and it can be performed using fast decision trees and probabilistic weighting. Due to this, it is the most suitable algorithm for the implementation in this project. Their technique involves intelligently aggregating the light-weight sub-detectors, i.e. light sensor, cellular module, and magnetism sensor. If the device is inside pocket or bags, the light detector cannot provide accurate detection results,

but in this project's case that is not a limitation, as the data is always measured only during the interaction with a turned-on display. The authors claim a consistent above 88% detection precision, which should be satisfactory for discovering the average user conditions during app usage.

The approach requires computing of certain data in advance of the detection for correct calculations. The first are approximate sunrise and sunset times at the user's location, to be aware of when to use and when to exclude the impact of the light sensor on the final decision. During the dark hours the algorithm then uses the probabilistic weighting narrowed to only cell signal and magnetism. These times and resulting daylight duration can be precisely deducted after knowing the user's location. But since for this rough estimate merely knowing the region or country of the user might suffice, other possibilities than directly requesting location permission also exist. One option might be to utilize Android Locale[7] for gathering the set user's country and/or timezone as general location. Another option would be to fall back to the last known location in case the application does have location access but the GPS is currently not running. As the latitude and longitude gained in this way and resulting sunrise and sunset times might not be very accurate, a joined approach with a time offset compensation for this inaccuracy should provide a rough estimate. Once the location is obtained, this work implements the calculation of the sunset and sunrise times, with the help of the algorithmic logic provided by the Twilight Service in Android [83]. The second needed information is related to the magnetic sensor. In order to precisely account for the changes in a magnetic field surrounding the device, the mean value of the magnetic field's strength at the users location should be retrieved from a database. For simplicity, the described model uses the mean value of the magnetic field at the magnetic equator on the Earth's surface, which is a constant [79].

Once the the required values are known, the Algorithm 4.4 (for a simplified diagram see Figure A.4 in Appendix A) starts every time one of the sensors provides an update. Then, the values are processed using a simple infinite impulse response low-pass filter [80], which helps to smooth out extremes from the raw data. Each of the three inputs is further converted to the binary probability based on the thresholds as in the original work and they are eventually aggregated.

- The luminosity in the exterior is brighter than common artificial lighting in the interior, even in cloudy or rainy conditions. The more extreme bright or extreme dark the measurement is, the more it is to set to influence the final decision.
- The cellular signal inside of buildings is typically weaker than outside, due to the walls and other parts of the construction blocking the radio waves. Analogically to the previous example, the closer the strength of the current signal is to a possible minimum or maximum, the bigger is its impact.
- Magnetic field shows a high variance across places near and inside buildings, because of the disturbance from steel structures and electric appliances. On the contrary, it is rather stable in the open space where the only fluctuations are caused by the Earth's magnetic field. Therefore, in this case the probabilities are calculated from the signal's variability over time, rather then direct values. Highly volatile or completely stable magnetic field influences the result the most.

[7] https://developer.android.com/reference/java/util/Locale

Algorithm 4.4: Environment detection algorithm, invoked every time one of the relevant sensors provides an update and results in a binary decision meaning either inside or outside condition.

```
1: procedure EnvironmentDetection()
2:     α ← 0.5                                   ▷ smoothing factor of the exponential average
3:     l, c, m, mΔ ← 0.0
4:     mag ← 31.2                                ▷ mean value of magnetic field on the Earth's surface
5:     now ← get current time
6:     t_l, t_c, t_m ← 2500, 50, 20              ▷ decision values (thresholds)
7:     p ← 0.5
8:     if daytime(now) then
9:         l_prev ← l
10:        l ← get current light level
11:        l ← α · l_prev + (1.0 − α) · l
12:        if l < t_l then p ← p + 1
13:        else p ← p − 1
14:        end if
15:    end if
16:    c_prev ← c
17:    c ← get current cell signal strength
18:    c ← α · c_prev + (1.0 − α) · c
19:    if c < t_c then p ← p + 1
20:    else p ← p − 1
21:    end if
22:    m_prev ← m
23:    m ← get current total magnetic field      ▷ sum of the mag. fields on each axis
24:    m ← α · m_prev + (1.0 − α) · m
25:    mΔ_prev ← mΔ                              ▷ magnetic field variability
26:    mΔ ← α · m_prevΔ + (1.0 − α) · abs(m − mag)
27:    if mΔ < t_m then p ← p − 1
28:    else p ← p + 1
29:    end if
30:    p_average ← mean from the five latest p values
31:    if p_average < 0.5 then
32:        send_intent(outside)                  ▷ report the information
33:    else
34:        send_intent(inside)                   ▷ report the information
35:    end if
36: end procedure
```

In the end, the last five aggregated probabilities are averaged to the final result, to prevent fast changes to the states in cases of ambiguous conditions. The delay of this detector is on the account of slow environmental changes longer than with other algorithms in this work, hence is the most suitable for apps with complex use cases.

4.5.2 Lighting Conditions

In addition to the detection of environment type, luminosity sensor provides useful information also on its own. Research [77] showed that when lighting levels increase, users are forced to re-position themselves or the device to avoid glare, which can affect usability. It might even be as difficult to return to performing a task after such visual distractions as after an auditory distraction [9]. Therefore, for a later evaluation, the `EnvironmentManager` offers the option to log separately also the light levels surrounding the device.

4.6 Sound Detection

Regarding the distractions caused by sound, Thrift [1] detected more cognitive load in a general noisy environment as opposed to the quiet environment. Sarsenbayeva et al. [60] observed significant effects of the urban outdoor noise on participant's performance. Similarly, Cassidy et al. [14] showed a significant negative effect of outdoor urban ambient noise on free, immediate, and delayed recall tasks when compared to no noise. The results of Banbury and Berry [7] demonstrated that also office noise and speech can disrupt performance on memory for mental tasks, and that the effect is independent of the meaning of the sound.

On the contrary, some studies have found that the variance and semantics of a sound are more powerful distractions than intensity or duration [28]. When considering sounds generated by surrounding people, performance on task was slower in the experiments by Hummel et al. [24]. Finally, Tsiaousis et al. [70] claim that the extent of the distraction may be related to a number of sound properties, such as volume, semantics, and duration.

As a result, it might be reasonable to distinguish between multiple sound properties, when evaluating usability data collocated with these audio measurements. For purpose of detecting the sound semantics, AWARE framework [18] offers a plugin that identifies conversation from recorded sound. Such algorithm might be useful for usability studies or (with acquired consent) even for research conducted in the field. But despite of not storing the raw audio, it is unlikely such analysis could potentially be part of an assessment of real users without violating their privacy. Already for the mere sound detection on a sensor level, permissions to the microphone considered dangerous by Android platform are necessary. As a compromise implemented in this work, CIT offers the possibility of measuring the sound levels from a buffered unsaved audio[8] after it acquired run-time microphone permissions by polling the user, but does not go further by analyzing its meaning and neither conversation durations. This way it is possible for developers to distinguish between noisy (Figure 4.5 (a)) and quiet (Figure 4.5 (b)) environments.

Once the parent application is turned on, the audio volume detection algorithm, provided in Algorithm 4.5 (for a simplified diagram refer to Figure A.5 in Appendix A) starts in a separate thread. This thread is invoked once per second to avoid constant demands on the processing resources. As a first step in the thread, the microphone sen-

[8] Android offers the possibility to set the output file of the recording feature `MediaRecorder` to null, resulting in the audio only buffered in random access memory.

(a) (b)

Figure 4.5: Taking pictures in the noisy environment of a music festival (a) and using navigation in the quiet environment of a forest (b).

Algorithm 4.5: Sound detection algorithm.

```
1:  output ← '/dev/null/'                                   ▷ string
2:  r ← new mediaRecorder(output)
3:  mEMA ← 0.0                                              ▷ average amplitude
4:  EMA_filter ← 0.6                                        ▷ exponential moving average
5:  procedure SoundDetection()
6:      amp ← get maximum amplitude from r
7:      mEMA ← EMA_filter · amp + (1.0 − EMA_filter) · mEMA
8:      volume ← 20 · log10(mEMA)
9:      send_intent(volume)                                 ▷ report the information
10: end procedure
```

sor is polled for the maximum detected audio amplitude that was sampled since the last call. Next, this amplitude is processed with a low-pass filter using exponential moving average. In order to get a value in decibels from the amplitude, a conversion formula[9] is used, similarly as in other popular open-source noise level meter applications[10] published on the Android application distribution platform. Since decibels are a relative unit, the feature cannot serve to an unambiguous evaluation of the audio levels in comparison to other situations on other devices. However, it is argued that the varying levels throughout the session or historical data from the same device might be helpful when considered in respect to other data, such as user errors and time-on-task.

[9] https://en.wikipedia.org/wiki/Decibel#Field_quantities
[10] https://github.com/Arpapiemonte/openoise-meter

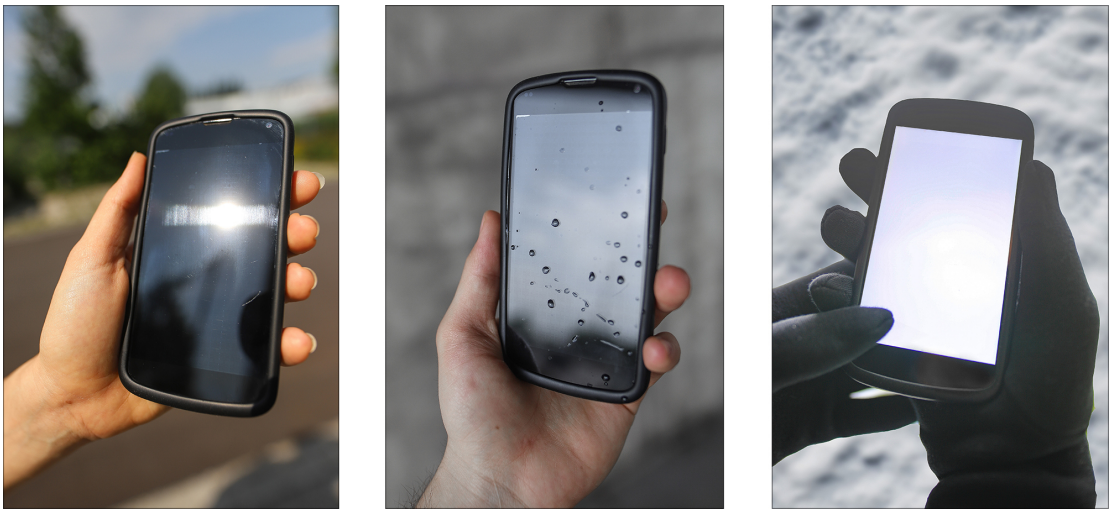


Figure 4.6: Influence of weather conditions on smartphone interaction. From left to right: strong sunlight, rain, and snow (user wearing gloves).

4.7 Weather Detection

It has been established by research that weather in terms of temperature and humidity as an environmental factor might affect distraction level of the users [24]. Ultimately, higher error rates or above average time-for-task measurements might be more likely viewed as acceptable, in case of unfavorable weather conditions (Figure 4.6) at the place of interaction, including rain or low temperatures. To assume weather conditions, the easiest method appears to be using user's location and time, matched with a call to an API providing historical weather data. Similar approach [81] can be found among the plugins[11] for AWARE framework. This work reads the weather data from a public Openweather[12] API using a free key registered at the service provider. Because of the necessary location permission, gathering of contextual data of this type might be the most suitable for those applications that are already allowed to access the user's location, such as fitness trackers and navigation systems. The pseudocode can be seen in Algorithm 4.6 (for a simplified diagram refer to Figure A.6 in Appendix A).

After acquiring the location, the first step of the algorithm is to contact the API server with an HTTP request. The program waits until it receives a reply or the connection is stopped with a timeout. The response is returned as raw data that need to be parsed for further processing. The used service provides information about a large number of meteorological parameters, from which the following are extracted, logged, and locally broadcasted to the parent application: temperature, humidity, wind speed, cloudiness, rain, and snow. For simplicity, all of the parameters are left in the original units, while rain- and snowfall are converted to a boolean value. Similarly, as with the sound detection, this algorithm also needs to run in a separate thread. That is mainly due to the asymmetric task of waiting for the remote server response.

[11] https://awareframework.com/plugins/
[12] https://openweathermap.org/

Algorithm 4.6: Weather detection algorithm. The invocation happens after receiving the access to the user's location and results in a list of weather parameters.

```
1:  procedure WeatherDetection()
2:      k ← weather service API key
3:      t ← maximum allowed timeout in milliseconds
4:      w ← [6]        ▷ data array (temperature, humidity, wind, cloudiness, rain, snow)
5:      l ← get current location
6:      while l = null do listen for location update
7:      end while
8:      object ← HTTP_request(l, k)
9:      while object = null and elapsed time < t do listen for server response
10:     end while
11:     if object = null then
12:         print(timeout)
13:         return
14:     else
15:         w ← parse(object)                    ▷ unpack raw server response object
16:         for i ← 1 to w.length do
17:             send_intent(w[i])        ▷ report the information to be logged separately
18:         end for
19:     end if
20: end procedure
```

4.8 Usability Metrics

Based on the friction patterns mentioned in Section 2.3, the framework offers methods for a convenient measuring of various usability metrics. The methods located in `UsabilityMetricsManager` need to be called from the parent application at the screens and in the activities of interest, thus in comparison to other CIT features, these require a more complex implementation. The main idea stems from logging the user-flow, saving the order of performed actions and visited screens in a time-stamped manner for a possibility of its later visualization and analysis. At the initiation of the application, the method `sessionStart` is to be called. This step saves the time of the app opening and instantiates all the variables, as well as the logging process. Defining the start and end of a task was also considered, but due to the emphasis on generality, this was omitted, since complex tasks (and task times) in some applications might be anyways discovered only by analyzing the whole user-flow. In case the user interacts with the touchscreen, the method `registerGesture` is automatically called with the performed gesture type passed as a parameter. The class also contains the method `unresponsiveGesture`, which is meant to be called by the parent app, in case a user touches an element without an associated action, such as whitespace. Due to the complicated implementation of this functionality, the other offered option is inverse: the parent application calls the method `registerGesture` every time it registers an action coming from the user. Hence, in the resulting log will always be a pair of the identical actions (once registered automatically by CIT and once by the parent application), un-

less the gesture will not be identified by the app and so unresponsive. Analogically to the registered gestures, the method `sessionChanged` is meant to be called with every visited screen, in order to capture the whole user session. Similarly, by the methods `appFocusChanged` and `displayConditionChanged`, the user exiting the app and the turned off display during the session are registered, respectively. Finally, the method `sessionEnd` is to be called at the end of applications life-cycle to log the end of the session.

4.9 Data Export

The export of the logged contextual and usability data is inspired by the AUToMAte Framework [86]. Every implemented manager class generates a separate CSV[13] file in the device's memory. Default location of the stored files is in the downloads folder, grouped in a separate folder, since this can be accessed in a majority devices by pre-installed file explorer applications and the files can be gathered easily at the end of eventual usability studies. Each file has a predefined name, header with column values, and a linked `TransmissionEvent` class. Each entry in each of the files is saved with a corresponding time-stamp. The logs are continuously filled with real-time data to prevent data loss and the last output streams are flushed at the end of the applications life-cycle (with the dump of the main `AwareManager` class). The saved files are readable in spreadsheet software, such as Excel or directly presentable by the developed interactive visualization (described in Section 5.2) in web browsers.

4.10 Information Management

The communication between CIT and the implementing application is solved through sending intents[14]. According to the official Android documentation, they serve as a bridge between application's activities. If at one part of the code an intent is broadcasted, any interested component might register to receive it. Every intent can carry arbitrary data defined by the sender. In case of the proposed library, the detected states are sent from respective detection components to the rest of the system, such as the parent application. This creates a unidirectional pipeline, using which CIT may deliver new information every time it is available, without the constant polling from the main application.

The code snippet in Program 4.1 shows the actual Java code, which is a part of the library. First, a new intent is initialized by defining its name. Next, the data later carried by the intent are attached. Each piece of information is added to the intent separately, defining the information's name in the process. Finally, the prepared intent, containing the attached data, is broadcasted to the system for other parties.

[13] comma-separated values

[14] https://developer.android.com/reference/android/content/Intent

```
1 Intent intent = new Intent("activity");
2 intent.putExtra("message", currentState);
3 intent.putExtra("magnitude", String.format(java.util.Locale.US,"\%.2f", magnitude));
4 intent.putExtra("steps", Integer.toString(stepsTaken));
5 LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
```

Program 4.1: Sending an intent that contains current activity state, acceleration magnitude, and the number of detected steps.

```
1 LocalBroadcastManager
2     .getInstance(this)
3     .registerReceiver(mMessageReceiver, new IntentFilter("activity"));
```

Program 4.2: Component registering for receiving intents of the specified type.

```
1 private BroadcastReceiver mMessageReceiver = new BroadcastReceiver() {
2     @Override
3     public void onReceive(Context context, Intent intent) {
4         String msg = intent.getStringExtra("message");
5         if ((msg != null)&&(intent.getAction() != null)) {
6             switch (intent.getAction()) {
7                 case "activity":
8                     String magnitude = intent.getStringExtra("magnitude");
9                     String steps = intent.getStringExtra("steps");
10                    textViewActivity.setText(msg);
11                    textViewActivityMagnitude.setText(magnitude);
12                    textViewActivitySteps.setText(steps);
13            }
14        }
15    }
16 }
```

Program 4.3: Overridden method as part of the instantiated `BroadcastReceiver`, responsible for catching the respective intents and their processing.

```
1 if (thread == null) {
2     thread = new Thread(){
3         public void run() {
4             while (thread != null) {
5                 try {
6                     Thread.sleep(1000);
7                 } catch (InterruptedException ex) {
8                     return;
9                 }
10                double noise = getVolume();
11            }
12        }
13    }
14 }
```

Program 4.4: A thread polling the microphone once per second.

Each application component that wants to receive the intents of a specific type needs to explicitly declare it as in Program 4.2. This is by definition performed using an instantiated `LocalBroadcastManager`[15] class. However, in order to process the received data, the component is also requested to instantiate a `BroadcastReceiver`[16]. This instance handles every incoming message and gives the opportunity to process the data further. In case of the application example in Program 4.3, the data is extracted from the caught intent by the defined information names and eventually displayed as text.

Since some parts of the CIT's code are dependent on the updates from sensors or a remote server, these specific actions need to be executed asynchronously in separate threads. Each thread is given valuable processor time only for a very short time of processing the update and then suspended for a given delay. In Program 4.4 is depicted the instantiation of a sound detection thread, which polls the microphone once per second. After the received value is saved to a variable, the thread is stopped and set to wait until its next turn.

4.11 Demo Application Development

In order to showcase the accuracy and combined context-awareness of the concept presented in this chapter, as a second part of the project a demo application for Android was developed. This app is highly simplified and its main purpose is the implementation of CIT to display the detected values on screen in real-time. It consists of one activity class ensuring the access to the necessary sensors (permission control), receiving and processing local broadcasts sent from the library, and updating the interface elements with relevant information. Additionally, it dispatches the registered touch events further to the library for identification of the correct hand, in case a user decides to preform a swipe gesture anywhere on the screen. It also offers an option to temporarily pause and then continue the detection to keep a static snapshot of all the data displayed. The application (see Figure 4.7), same as CIT, was created for Android devices with minimum API 19 (4.4 KitKat) and later.

Since the application requires multiple different permissions, as a fail-proof solution the requesting happens in three phases. Firstly, each of the permissions, listed in a string array, is verified separately as in Program 4.5. If any of the permissions are not granted, these are collectively requested from the user using the command in Program 4.6. This way, the app also avoids redundant asking for the already granted access. In case, the user rejects to grant some of these permissions, due to an error or on purpose, these are requested again (as shown in Program 4.7), with the only alternative of quitting the application. The reason for this is the safety during run-time with no further obligation to add a verifying condition before every statement using the access.

Once the process of acquiring the permissions is finished, the main class of CIT is instantiated and application starts receiving detected updates through the registered `MessageReceiver` component as intents. Each of the intents carries attached data, which is later unpacked and their content is pushed as an update to the `textView` interface elements, in order to be displayed.

[15] https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager
[16] https://developer.android.com/reference/android/content/BroadcastReceiver

```
1 public static boolean hasPermissions(Context context, String... permissions) {
2     if (context != null && permissions != null) {
3         for (String permission : permissions) {
4             if (ActivityCompat.checkSelfPermission(context, permission) !=
5                 PackageManager.PERMISSION_GRANTED) {
6                     return false;
7             }
8         }
9     } return true;
10 }
```

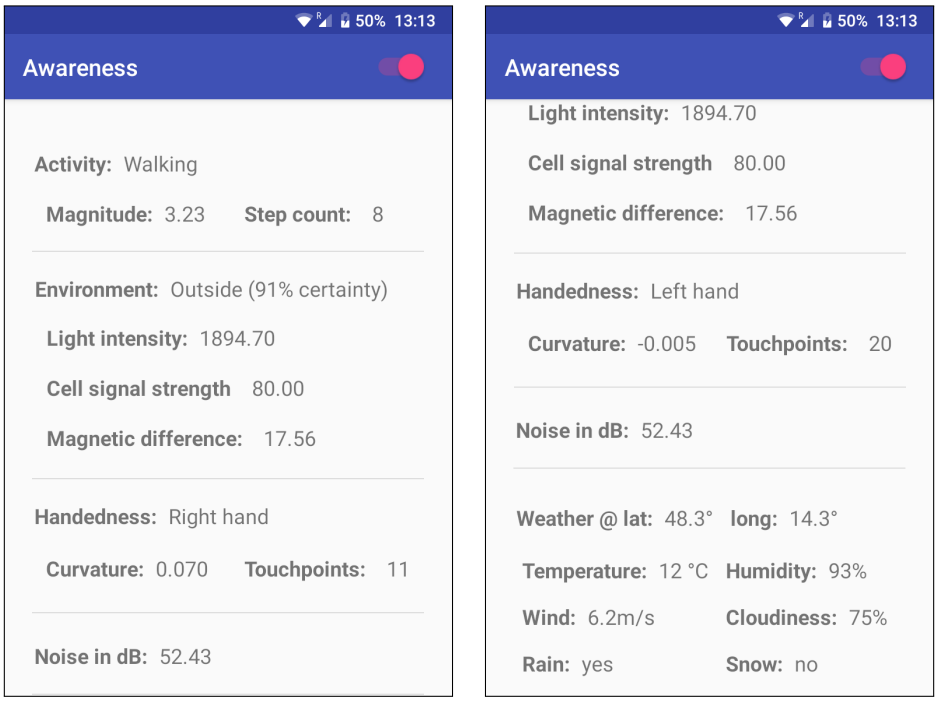
Program 4.5: Checks if defined permissions, such as `WRITE_EXTERNAL_STORAGE` are granted to the application (adapted from [84]).

```
1 private void requestPermissions() {
2     if(!hasPermissions(this, PERMISSIONS)){
3         ActivityCompat.requestPermissions(this, PERMISSIONS, PERMISSION_ALL);
4     } else {
5         awareManager = new AwareManager(this);
6     }
7 }
```

Program 4.6: Main method for permissions requesting called before any user-granted access. The variable `PERMISSIONS` is a string list and `PERMISSION_ALL` equals to 1.

```
1 public void onRequestPermissionsResult(int requestCode, String[] permissions, int[]
      grantResults) {
2     boolean permissions_granted = true;
3     switch (requestCode) {
4         case 1:
5             for (int result : grantResults) {
6                 if (result != PackageManager.PERMISSION_GRANTED) {
7                     permissions_granted = false;
8                 }
9             }
10            if (permissions_granted) {
11                awareManager = new AwareManager(this);
12            } else {
13                requestPermissions();
14            }
15            break;
16        default:
17            super.onRequestPermissionsResult(requestCode, permissions, grantResults)
18    }
19 }
```

Program 4.7: Listens for the result of the permission-requesting process. Since the request code can be chosen arbitrarily, switch construction is used for easy extensibility.



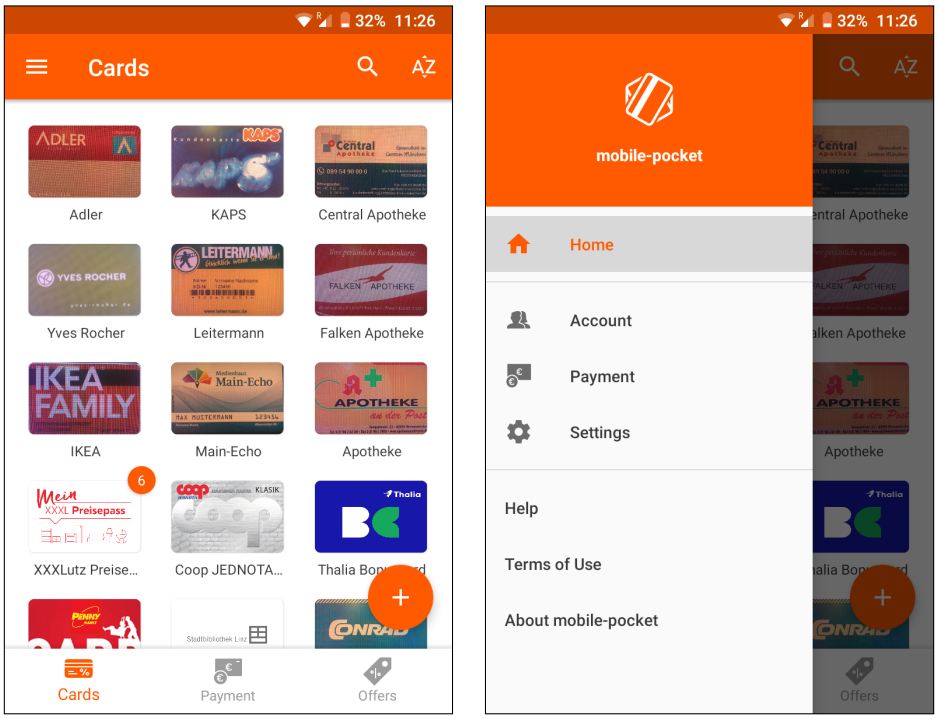
(a) (b)

Figure 4.7: Showcase application with the view scrolled to the top (a) and to the bottom (b).

4.12 Integration to the Application mobile-pocket

To test CIT in conditions for which it was developed, it was implemented in mobile-pocket[17], a commercial Android application with multiple features. Its main purpose is to serve as a wallet, storing loyalty, bonus, benefit, club, family, and any other cards from a bar-code, QR code, or a customer number. The application has a built-in list of numerous businesses, such as grocery stores, restaurants, and pharmacies, or other organizations, including libraries, and educational institutions. The user is able to search this list and add cards using their code, either by scanning them with the device's camera or by entering the bar-code digits manually. At the end of this process, the app is able to generate and display the codes on its own, which can be presented at the cash desk during a payment. Hence, the user does not have to carry the original cards with them and can conveniently own a higher number of these cards at the same time. Furthermore, users are able to create an account, where the cards are subsequently stored, protect the application with a password, receive exclusive deals with integrated coupons, and perform similar related tasks. In this thesis, the application is considered as a tool for quality evaluation of CIT proposed in this chapter. At first, the implementation of the library into the app is discussed. Later, this adjusted version is evaluated in a supervised study, confirming the CIT's functionality and discovering possible usability issues of mobile-pocket in the process. Finally, the results are reported.

[17] https://www.mobile-pocket.com



(a) (b)

Figure 4.8: Application mobile-pocket: main activity with the card list (a) and the main activity with an overlay of the opened side drawer (b).

The code implementation was realized with help of bluesource[18], the company affiliated with development of mobile-pocket. In comparison to the implementation into the demo application described in Section 4.11, here the permissions management was resolved, but the task involved apart from merely running the `AwareManager` detection core, also making more complex connections for measuring usability parameters. Individual methods of `UsabilityMetricsManager` were set to be called at the respective locations in code. For instance, start of the session was defined as initialization of the main activity (pictured on Figure 4.8 (a)) and transitions to other screens were bound to the respective navigation buttons. Some interface elements, such as search icon in the top right corner and secondary screens, including Help and Terms of Use section were ignored in this process, because they were not of interest in the subsequent usability testing. However, some actions that were needed, were not as straightforward and easy to implement. For example, in order to open of the left-side drawer (Figure 4.8 (b)), user has multiple options: pressing the hamburger-menu icon in the top left corner or performing a swipe gesture from the left edge of the screen to the right. Eventually, a decision was made to call the method `sessionChanged` during the drawer sliding animation, which occurs in both cases. As this action is an animation with a specified duration, this unfortunately led to numerous triggering of the method and the collected data had to be inspected for duplicate rows before analysis. Therefore, to assure correct results, it is crucial to consider all the possible times the hooked method will be called.

[18] https://www.bluesource.at/

4.13 Testing

As an early empirical confirmation that the Context Information Toolkit is working correctly, a usability testing with the application described in Section 4.12 in various contextual conditions was conducted. The secondary goal of this pilot study was to establish a list of usability issues, which could be later used for evaluation of the second research question (stated in Section 2.2). Moreover, the discovered problems with usability were later reported along with suggested improvements and the contextual data recorded by the library.

As a first step, the application was inspected with the expert review[19] method for any possible friction in a typical user-flow. Based on the gained experience and conclusions, three user testing scenarios were formulated. These tasks were designed to be fairly short, reflecting the simple nature of the application and, what is more, the time constraints posed by the usability testing event. Each of the scenarios pertains to a specific predicted usability problem or problems and is presented in a form of a short story:

1. *Imagine you are in XXXLutz*[20] *to make a big purchase. Unfortunately, you forgot your loyalty card at home. In order not to lose your customer points, someone from home sent you the bar-code number of the card. In the application, create a new XXXLutz card with the bar-code number and display the generated code to the cashier.*
2. *Your entire purchase has already been scanned by the cashier. You are asked to quickly present him the bar-code of your loyalty card. Find the PENNY Markt*[20] *card from the list and display the bar-code in full screen mode.*
3. *You want to add your new customer card from Germany to the app. Find and change the setting that enables it to you.*

Every one of these scenarios can be correctly accomplished only by performing a specific set of actions in a specific order (also called a 'sunny case' scenario). Tapping on any other interface element or drawing a different gesture results in increasing the time-on-task, confusion regarding the next step, and in some cases also in not being able to reach the goal without starting over. The specific steps per scenario, which the users are required to take, are stated in Figure 4.9.

As the Table 4.1 presents, in total six people (one woman) were asked to participate in the testing, for 12 minutes each. Every participant was requested to fill in a screening form, complete all three test scenarios, and answer a number of questions in a post-test interview. In average, the participants used the smartphone one to three hours a day and owned some loyalty cards. A half of the participants owned an Android device while the other half used iOS. The age mean of the sample was 27.3 years, and whereas four participants used devices with bigger screens (approx. six inches), the other two claimed smaller screen size (approx. four inches). None of the testers used mobile-pocket before.

[19] https://www.nngroup.com/articles/ux-expert-reviews/

[20] Popular physical retail chain with the pilot study participants.

ID	Age	Gender	Platform used	Size	Usage	Loyalty cards
1	31	male	Android	6"	1-3h	0
2	23	male	iOS	4"	1-3h	0
3	35	male	Android	6"	3+h	4-5
4	24	male	iOS	6"	1-3h	4-5
5	20	male	Android	6"	3+h	2
6	31	female	iOS	4"	0.5-1h	4-5

Table 4.1: Summary of participant's details, based on the filled screening forms. Size refers to the diagonal screen size of the device typically used by the participant and the usage specifies the time spent interacting with a smartphone per day.

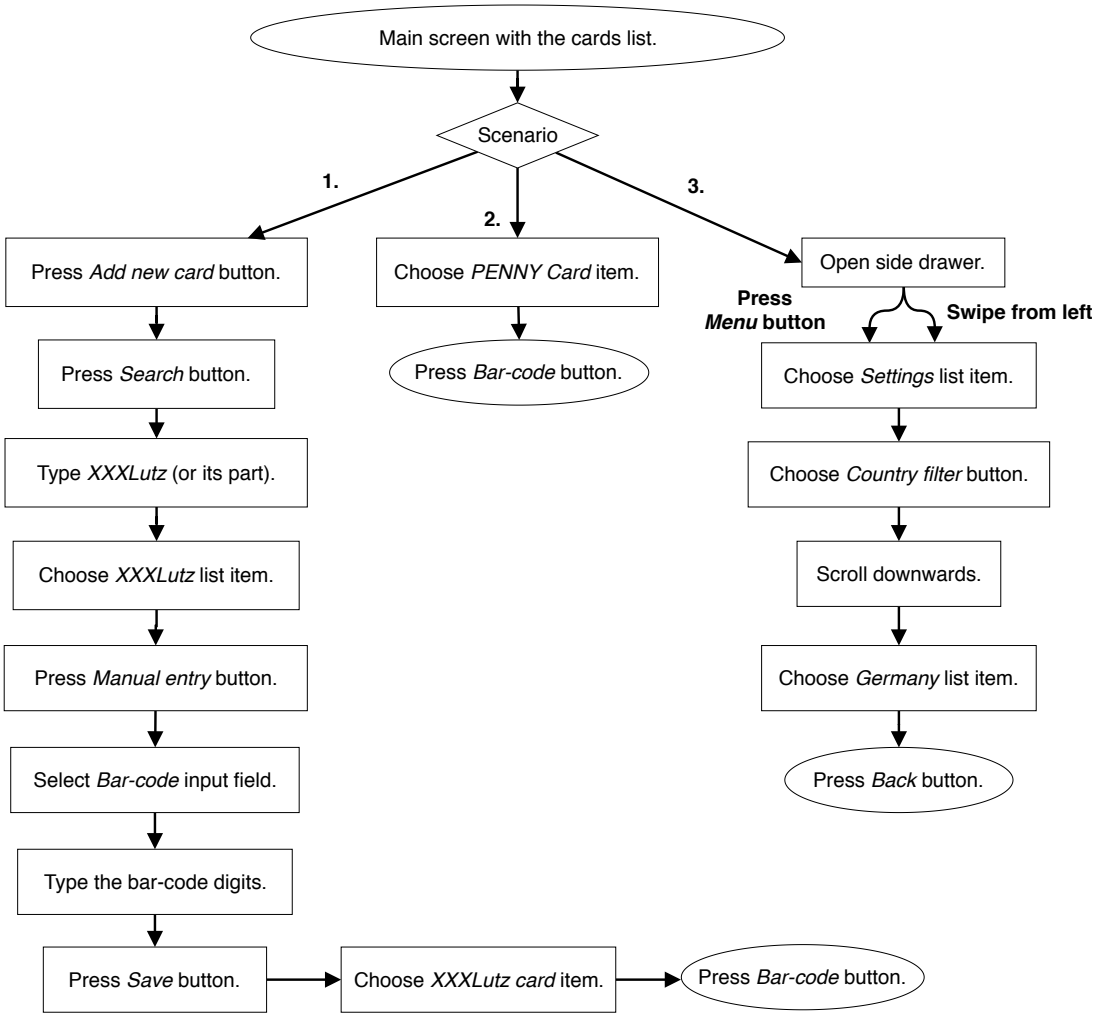


Figure 4.9: Concrete steps that are necessary to be accomplished inside of the mobile-pocket application to reach the goal of the described scenarios from the main screen (depicted in Figure 4.8 (a)). All of them begin on the same main screen and a user has a choice between two interchangeable actions, when opening the side drawer.

To capture the contextual parameters with CIT and to avoid the learning curve bias in data, the test scenarios altered between being accomplished while sitting by a table and walking. The walking took place in a long corridor in an ordinary pace defined by the participant. However, in case it was noticed that they started slowing down, they were requested to speed up. After each task scenario was completed, the log files containing the contextual data were backed up.

The setup involved two devices, namely Nexus 4 with 4.7" 1280×768 px display running Android 7.1.2 (Nougat) and HTC One with 4.7" 1080×1920 px display running Android 4.4.2 (KitKat). The two devices were interchangeably used to speed up the process between two task scenarios that involved saving the measured data. Apart from the testing devices, the setup also included a webcam, a multi-directional microphone, a laptop for making the recordings while sitting, and an additional smartphone for sound recording of the tasks performed during walking. The moderation was mostly done by the facilitator, the notes and recordings by the observer.

The participants were briefed about the reason for testing and were explained that the session will be recorded. After they filled and signed the screening form, they were asked the introductory questions, including if they use or heard of mobile-pocket and how many loyalty cards they own. Besides that, they were also inquired about the hand they normally use to interact with their smartphone. The participants were further asked to perform the described tasks in a randomized order, as depicted in Table 4.2, one at a time. The first task, demanding that the participant will enter a code from the card manually, consisted of them reading the number off a sticker placed over the smartphone screen.

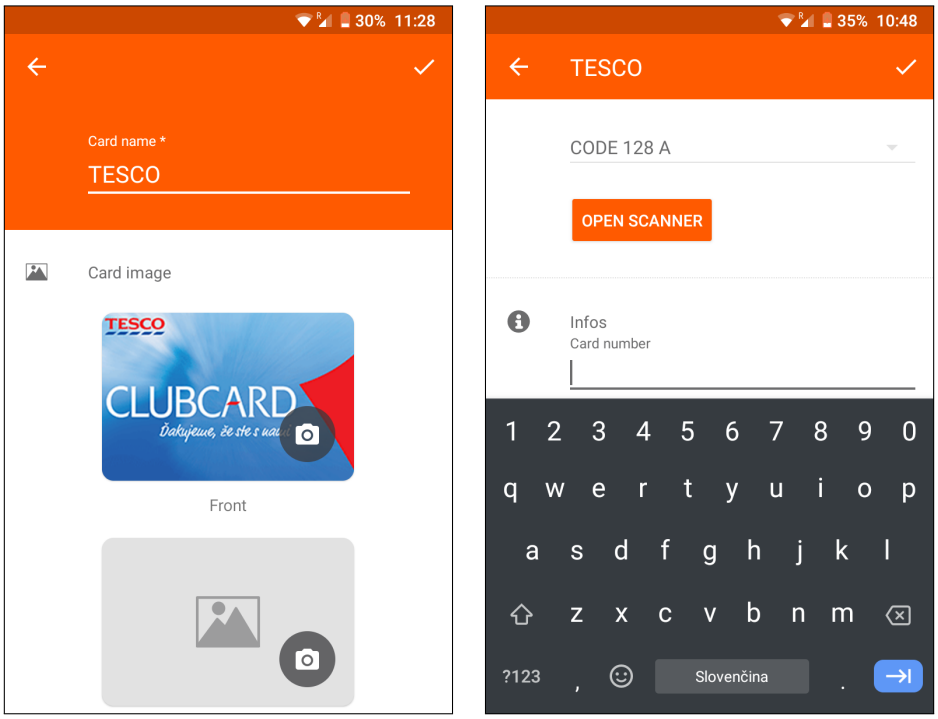
P1	P2	P3	P4	P5	P6
1: walking	1: walking	2: walking	2: sitting	3: sitting	3: walking
2: sitting	3: walking	1: walking	3: walking	1: sitting	2: walking
3: sitting	2: sitting	3: sitting	1: sitting	2: walking	1: sitting

Table 4.2: Scheduled order of six participants (P1-P6), their assigned task number (1, 2, or 3), and activity scenario (sitting by a table or walking) performed during the task.

To deepen the understanding of the cognitive friction points observed, and to receive qualitative feedback, the participants were asked the following questions at the end of the session:

- What would help you to work with the app easier?
- What do you think about the size of elements, such as buttons or symbols?
- How do you like the text in the app? Is it readable?
- How did you find the layout and organization of the app? Were the elements, such as buttons where you expected them to be?
- How do you find the difficulty of this task compared to when you could sit at the table / if you had to walk?[21]
- How would you find the app to be controlled just with one hand at all times?

[21] The participant was asked about the alternative, which they did not perform, for each task.

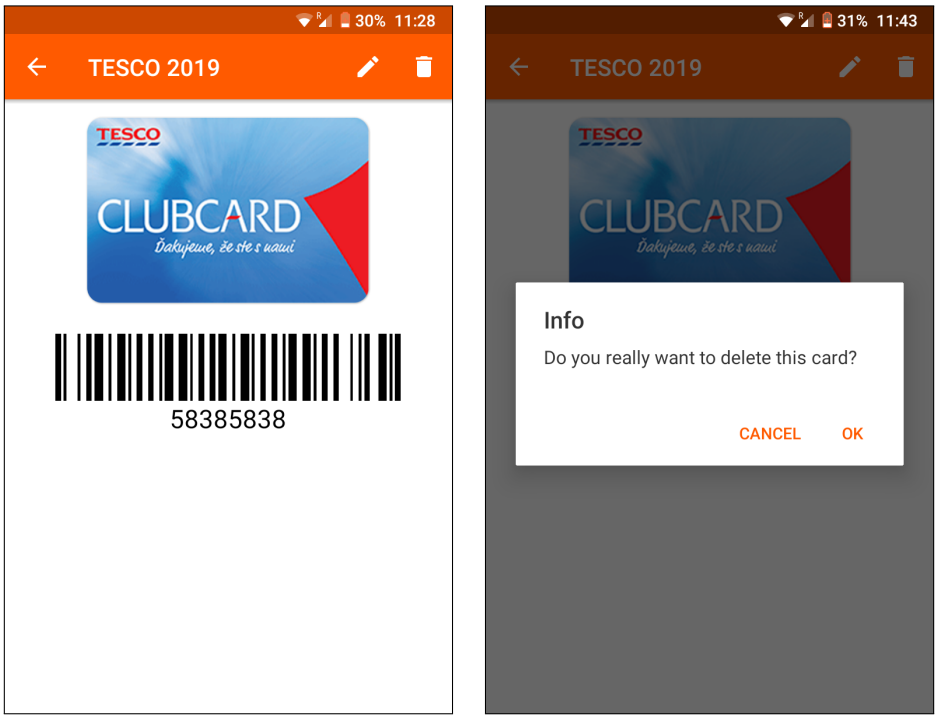


(a) (b)

Figure 4.10: Edit card screen (a) and the keyboard overlay opened (b).

Apart from one task with one participant, all were accomplished successfully. However, from 18 tasks performed in total, the facilitator had to assist six times and that mainly with the task 3 (four times) and task 1 (two times). The assistance was in form of a verbal hint, once the participant got confused, e.g. constantly looked for the scenario solution in a wrong place. In general, there was a couple of repeating issues with the mobile-pocket. In total, 14 unique usability problems were identified.

For the purposes of this thesis, only those issues from this pilot study are discussed, which are relevant to the formulation of the experimental task used in the following user study (see Section 5.1). One recurring problem was related to the card edit screen (Figure 4.10 (a)), when the tested participants would consistently overlook the save button (in the top right corner with the checkmark icon) and subsequently not save the edits made. Its bad visibility was later confirmed in the post-test interviews, since the symbol received negative feedback for its narrow font weight and small size. The situation was worsened by the order of actions a user generally performs on this screen. Firstly, the size of the two 'card image' elements caused that the input field for entering the card number was beyond the bottom fold. Therefore, the user had to find the input field by scrolling down. Secondly, once a user successfully entered the code, it was very often the case that they confirmed the action by pressing the blue enter key on the keyboard (Figure 4.10 (b)). However, this caused scrolling action and moving the cursor to the latter Infos field. The user had hence the suggestion to fill in this input field as well, as if it was mandatory, while finally expecting the confirm button at the very bottom. This distracted the user from looking for the form confirmation elsewhere.



(a) (b)

Figure 4.11: Card detail screen (a) and a pop-up dialog (b).

Some of the participants encountered similar troubles already on the screen proceeding the card editing. The card detail screen (see Figure 4.11 (a)) features similarly placed and sized icon buttons to the aforementioned checkmark button. The tested users had primarily problems with identifying the meaning of the edit button (the pencil shaped symbol in the top right corner on the left) and thus also finding the way of editing the card's properties. Additionally, even after recognizing its meaning, the users sometimes missed the small button area when tapping, which mainly occurred during the walking contextual condition. This, in the cases when the adjacent delete button was pressed, invoked a dialog window (see Figure 4.11 (b)) asking for a delete confirmation.

As a result of the pilot testing, it was clear that the task of editing a card's properties contained multiple friction points, with some of them possibly being negatively influenced by the activity a user is performing during the interaction. Another of the discovered outcomes was that recording of the time-on-task and time-on-screen parameters makes the complete sense only in usability testing that does not use the 'think out loud' protocol[22]. This is due to the increased cognitive load, typically posed on the participants in supervised usability studies, when they are, apart from performing the task, also further distracted by the usability facilitators asking questions. Key takeaways, regarding the context detection algorithms used, were that the observed detection accuracy was on the levels declared in the source research papers, and that walking activity takes several seconds to be recognized, thus in the subsequent experiment, the testers should be asked to start walking already before beginning the scenario.

[22]Method, when participants are asked to simultaneously describe the performed actions with words.

Chapter 5

Evaluation

In this chapter the presented concept, its workflow, and logging capabilities are evaluated based on a conducted experiment. The acquired data are analyzed using a custom-developed visualization tool and the results are summarized.

5.1 User Study

Considering the usability issues, which the pilot study (described in Section 4.13) had uncovered in the tested application, and the interest of the second stated research question about the relationship of usability metrics to context, an experiment was conducted. By focusing on the discovered friction points and comparing them in two contrasting context conditions detected by CIT, the context distinction might be visible also with a small test sample. Moreover, the experiment results may serve as a motivation to use the proposed detection algorithms later in a larger study, or even in production. The two context conditions compared in the experiment were:

- Calm environment with no audio or visual distractions, where the participant is sitting and the device is laying statically in front of them on a table throughout the whole task.
- Busy environment of a grocery store with typical noise levels, visual distractions in form of static obstacles and moving people, where the participant is forced to keep walking throughout the whole task (on a path with turns).

While formulating the experiment task, multiple factors were taken into consideration. On one hand, the resulting data from the experiment had to be complex, as otherwise it would not sufficiently demonstrate the differences in participants' performance in the two testing environments. On the other hand, data too complex may cause this distinction to become hardly visible or cause the test participants to lose interest in finishing the experiment that is too difficult. Another factor influencing the task formulation were the quantifiable values, on which the difference in experimental conditions can be shown. As the framework is capable of measuring the interaction times and various types of friction patterns, such as navigational errors, accidental touches, or unresponsive gestures, the two typical usability metrics time-on-task and error rate were considered. Specifically, instead of measuring both during the whole task, the following continuous scenario was chosen to record each independently, one after the other:

Find the Tesco Clubcard[1] *from the list by scrolling and display it in detail. Then, change the current name of the card to 'TESCO 2019'.*

All the substeps the scenario required from participants are in Figure 5.1. The first sentence talks about the part of the task, in which the time was measured. In order to be able to properly compare the time differences in various contexts, this searching part of the scenario was made on purpose more difficult, than it would be during regular application usage. The participants had to search for the correct card among 74 others and were instructed not to use the search functionality. The wanted card was located nearly at the bottom of the list. Once a participant found and selected the correct card, the subsequent task of changing the cards name (described in the second sentence) was no longer timed, but instead the number of errors was counted. Whereas measuring the number of errors in the first part would not be meaningful (unless for an unlikely scenario of a participant selecting an incorrect card that would be excluded from the final dataset), measuring time in the second part would be skewed by the participant's typing speed. This way, the scenario, which was accomplished by each participant in only one type of context, transitioned naturally between the two usability metrics.

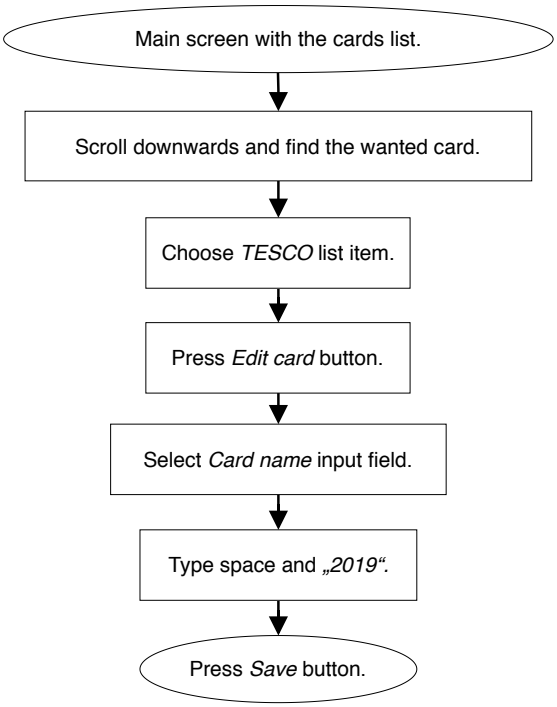
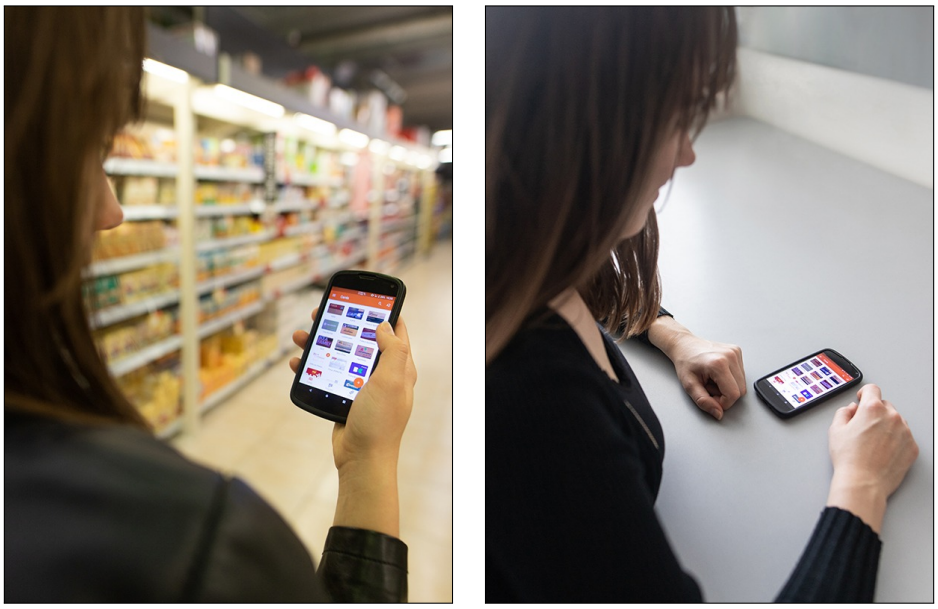


Figure 5.1: Six steps, necessary for the test participant to accomplish, in order to reach the goal of the assigned scenario. The task begins on the main application screen (depicted in Figure 4.8 (a)) and finishes once the participant saves the card being edited.

[1]Loyalty card of a physical retail chain popular with the test participants.



(a) (b)

Figure 5.2: Experiment setup of the busy condition (a) and the calm condition (b).

In total, ten individuals were recruited for the experiment, five of which were male and five female. Four of the individuals were between 50 and 65 years of age. The remainder of the participants were between the ages of 20 and 30. All of the participants had at least some university or college education. A half of the participants submitted that they had a high technological comfort level, while the rest were either medium or low. Nine of the participants used a smartphone device at least once per day. None of the participants had used the mobile-pocket application before the experiment took place and neither they had been acquainted with it in advance. However, before the start of the experiment it was assured that all participants were familiar with the design, color combination, and brand logo of the specified loyalty card. All the experiment sessions were conducted on a Nexus 4 device with 4.7" 1280×768 px display running Android 7.1.2 (Nougat). One half of the testers absolved the task in a usual large supermarket during busy hours (Figure 5.2 (a)) and the other half was tested in a calm environment of a quiet room (Figure 5.2 (b)).

As a result of the experiment, seven log files[2] containing data were downloaded from the device per participant, which means 70 in total. One complete experiment session (including the second part) was in average 36.5 seconds long and during that time the Context Information Toolkit exported approximately 73 kilobytes of text data. It is argued that this is a negligible size, which can be easily transferable to a remote server and does not influence the user's mobile data plan, mostly after a suitable file compression. What is more, these log files can be further processed and transferred only as average periodic statistics or in case of actually detected usability friction patterns.

[2] Activity, environment, handedness, light condition, linear acceleration, sound level, and usability metrics.

5.2 Data Analysis

The raw time-stamped logs provided only little insight to the recorded sessions separately and in a textual form. To gain a better perception of the merged dataset, and in order to offer a similar automatically generated view to any developer using CIT in the future, a decision was made to program an interactive visualization supporting the library. This would later help to evaluate the files generated in the experiment, the results of which are reported in the Section 5.3. One of the requirements was to make a chart that is showing the log files and is displayable in an internet browser, which concurrently enables its compatibility with web-servers and online dashboards. As a popular web-oriented JavaScript visualization library, D3.js[3] was chosen for the project. It offers the possibility of preprocessing the data and mapping them to a timeline, based on the assigned time-stamps. The ultimate goal was to show all the measured parameters per testing session on a single screen, with the option to turn showing the data from specific detectors on or off. It is argued that this way the interacting user gets the opportunity to explore the dataset on their own, discovering relevant correlations and their implications on the app being developed.

Preprocessing data and categorizing them to correct data structures was the first necessary step to be accomplished. As mentioned in Section 4.12, some duplicate values needed to be eliminated. Several data points were excluded, based on a filter of extreme values. Moreover, the code-generated application screen aliases were renamed to corresponding names and contextual states were associated with numeric values as elaborated further. After the data were processed, it was possible to display first elements.

The timeline (x-axis) boundaries, relevant for the session being viewed, were given by some of the measured parameters. It can be assumed that the scenario starts after the main screen of the application is loaded. More specifically, to avoid taking any possible delay into consideration, the time-stamp that marks the timeline's beginning was set to equal the very first time a user interacts with the touchscreen. Conversely, the end time was given by the last step of the scenario, which is in case of the experiment described in Section 5.1 the action of pressing the save button on the edit card screen (Figure 4.10). Once the timeline (x-axis) was calculated, y-axis was set to an arbitrary interval from 0 to 100, since all the contextual parameters are in different units and needed to be placed on the graph recalculated to the same range. Besides that, the merged dataset contains various value types. Whereas luminosity, sound volume, and linear acceleration are continuous ratio data types, the rest including activity, handedness, and environment are of a discrete nominal type. This additionally complicated the overlapping representation on one timeline. The presented solution divides these two groups and displays only the ratio data on the scale, while showing the continuous changes in the nominal data throughout the scenario in the upper area, where the ratio values only rarely reach. Furthermore, for the group of nominal data, visualized as lines, also a dot representing every time-stamped entry (condition change) was added for better visibility.

In addition to the contextual parameters in the chart, screenshots from the application were positioned over the graph area to the places, where the respective screen change was detected. The images are sized only as thumbnails, but it is argued that this allows for a quick orientation in the scenario, while not obstructing the view of

[3] https://d3js.org/

the actual data. Next, the gestures, performed by the participant during the session on the touchscreen, were mapped to a series of black icons and those were positioned on the timeline at the corresponding time-stamps. The majority of gestures performed were scrolling and tapping, although some of the participants tried interacting with the application also by double tapping and the long press gesture.

In the background of the chart, a dark gradient symbolizes another information. The order of screens, visited by the test participant, is compared to the sunny case scenario of the experiment task. The part of the interval, where these two do not align, thus the tester made a navigational error, is highlighted as a grey gradient area. This enables a fast comparison of the error-free sessions to the ones, where the participants encountered problems.

Since the concrete parameter values are not displayed on the y-axis, as that would cause their overlapping, an overlaying vertical line following the mouse cursor was implemented. Next to this dynamic line, the measured states and numerical values, which correspond to the specific point in time, are shown.

Above the chart and screenshots, the visualization settings are located. A series of checkboxes depicted in the corresponding colours enable to hide or reveal any of the visualization's parts. In the top left corner, the interacting user is able to change the displayed session by choosing different source data from the drop-down menu. In the top right corner the total length of the session, currently mapped on the timeline, is written in seconds.

The scenario depicted in Figure 5.3 shows a recording of a participant's interaction in the busy condition. The numeric values, including luminosity (green), sound volume (yellow), and linear acceleration (pink) are, according to the expectations, strongly variable, as the physical circumstances in the user's environment were quickly shifting during the walk through the supermarket. The nominal data from environment (dark blue) were correctly evaluated to the interior condition and activity (light blue) was correctly identified as walking, with a short exception near the end of scenario. The brief change to the still activity condition can be possibly assumed to be a result of the participant slowing down shortly after a navigational error occurred. Handedness (red) changed throughout the scenario, since the tester interacted with the device holding it in both hands and using fingers from left, as well as right hand, sometimes also altering the grip. This might have been caused by the long time necessary for the participant to identify the wanted card from the list, as can be seen from the numerous scrolling gestures recognized in the first half of the recorded scenario. In addition to the scrolling on the main screen, the tap gestures can be also seen appearing later in the session and the scrolling action for the second time on the edit card screen. There the obvious aim of the participant was to reveal the content of the form, located beyond the bottom fold. This information can be assumed from the combination of the data about performed gestures and current screens, immediately visible from the screenshots and icons on the visualization.

Overall, the visualization is intended for large displays with a high resolution, due to the total amount of displayed data. However, the layout is able to adjust also to smaller displays, where it can be used mostly with lower number of parameters enabled at the same time.

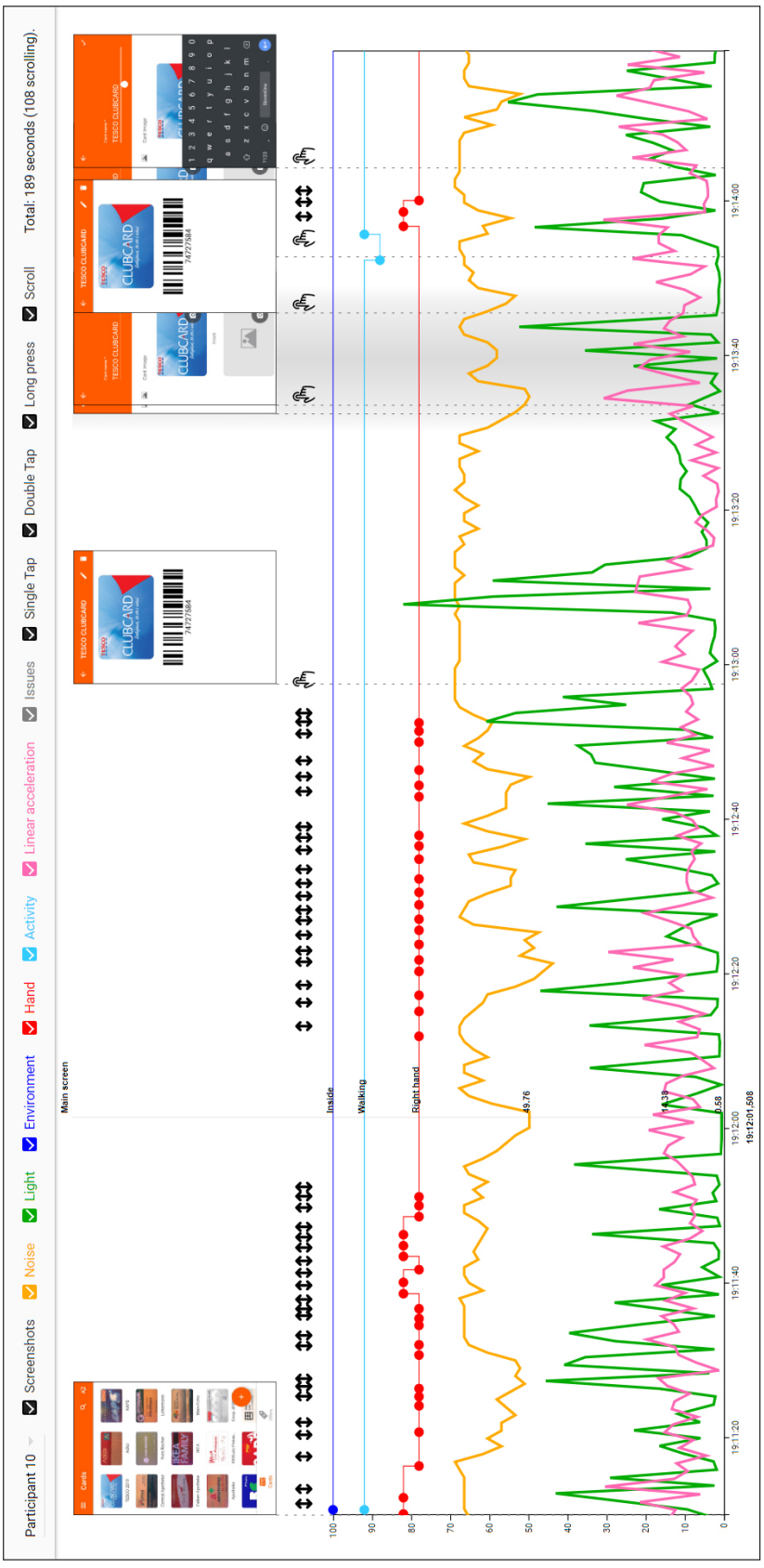


Figure 5.3: D3 visualization providing an overview of the captured session. Overlapping images can be sorted on z-axis by the user.

5.3 Reporting the Results

All the ten recruited participants accomplished the assigned scenario successfully. In contrast to the pilot study, no hints from the facilitator were provided. Once a participant encountered a usability issue, they were instructed to continue trying to find the task solution on their own. As the participants were also not instructed to verbally describe their actions, this allowed for insights from nearly realistic usage situations. Therefore, the results show varied types of usability friction, such as navigational errors when locating the edit card or save actions, accuracy errors when targeting the buttons, and delays in proceeding through the task, when looking for other interface elements.

As can be seen from the Table 5.1, majority of the participants (4) performing the task in a calm setting did not encounter a usability problem and only one had an issue with finding the action for card editing. More specifically, this participant tried tapping on the card image and the card's name in the title, before noticing the edit card button in the top right corner. Conversely, in case of the busy setting in a supermarket, a single user was able to complete the task without issues, but majority of the testers (4) did face difficulties. Two participants expected the edit card action to be associated with the card image. Furthermore, one tester had a problem with locating the save button in the edit card form and another incorrectly tapped on the delete card button when aiming for the adjacent edit card action. Participant 7 encountered two navigational errors during their session, both while searching for the action of editing the card: tapping on the card image and card name in the title, and eventually returning to the main screen and trying to find it there.

ID	Condition	Time-on-task	Error count	Problems with
1	calm	22.3	0	-
2	calm	23.2	1	finding the edit card button
3	calm	12.4	0	-
4	calm	16.2	0	-
5	calm	15.7	0	-
6	busy	11.1	1	tapping on the edit card button
7	busy	39.7	2	finding the edit card button
8	busy	41.0	0	-
9	busy	75.9	1	finding the edit card button
10	busy	107.5	1	finding the save button

Table 5.1: Results of the experiment, where half of the participants absolved the scenario in a calm context (1-5), while the other half in a busy contextual condition (6-10). Time-on-task is in seconds and concerns the first part of the task, from the beginning until the participant was able to find and select the correct card.

Searching for the correct card from the list of others might also show the difference between the two conditions. Whereas the time-on-task mean in the calm environment was 17.96 seconds, the same task took the participants in the busy conditions on average more than twice as long, 55.06 seconds. That is, despite of including the participant 6 with the fastest time-on-task result from the whole experiment.

5.4 Discussion and Summary

Following the pilot study with the outcome of a list of usability issues in the mobile-pocket application, an experiment with ten participants was performed, comparing the impact of some of these issues in different contextual conditions. In spite of the small sample size the results suggest a likely correlation of the contextual parameters, measured by CIT, and the popular usability metrics time-on-task and user error-rate. It appears that with increase in device's linear acceleration and larger variations in sound volume and lightness levels, detected by the implemented algorithms, the time a user needs for accomplishing a task also increases and users make more errors. What is more, this connection seems to be multiplied in case the application contains usability issues or unclear friction points that make the user insecure about the intended steps leading towards their goal.

Due to dynamics of the detected contextual conditions and low accuracy and sampling rate in most of the devices, a concrete assignment in a specific moment of time to an occurrence of a friction pattern cannot be easily made. However, the comparison of calmer and busier, stress inducing conditions can be clearly visible from the data. Using the created visualization, application developers can observe the aforementioned correlation in detail, enabling them to judge the contextual influence per parameter and per user interaction, down to the task level. Aware of the the context, in which their users spend most of the time, they might conduct usability studies, which are closer to the realistic interaction and make more informed design decisions.

For instance, Schildbach and Rukzio [62] suggested that the negative effect of different mobility states on target selection can be compensated by increasing the target sizes. Other researchers have also shown that adaptive text entry and providing audio guidance can compensate the negative effect of situational impairments [60]. Hence, the developers of mobile-pocket might, for example, consider increasing the size of the edit card and delete card buttons on the detail screen and the save button on the edit screen.

Similarly, noticing that in relation to situations with a busy context, many users tap on the card picture and then within a second return, might suggest the UX team that the action of showing the card image in full screen mode is not what is expected. Since in calm states, when the user is not distracted, it seems to occur only rarely, the scale of this issue or this usability problem entirely may be possibly overlooked in laboratory usability studies that exclude context. However, having the data from the proposed contextual library might help to reveal that the resulting action of tapping on the card should be exchanged. That is, either for a one determined by further user interviews or directly the one, which users usually perform as the next in the recorded user flow.

Finally, even though in case of mobile-pocket, the experiment scenario of searching the right card among a plethora of others is not realistic, it clearly shows that CIT is able to signalize an increased time-on-task in busy contexts, in comparison to what might have been the time measured in a traditional usability study without distractions. Subsequently, the measured datasets can be helpful in determining the typical user conditions and optimizing for them, not just in terms of effectiveness but also efficiency.

Chapter 6

Conclusion

This thesis presented Context Information Toolkit, a library supplementing interaction analytics with data describing the situation. At first, the motivation behind the work and ways of measuring different types of context were explained. Then, the related research projects and commercial tools were reviewed, resulting in an identified gap in the available solutions. Building on the algorithms developed in the field of ubiquitous computing, the main concept and structure of the project was proposed. Its computational logic was illustrated using flowcharts and examples. The implementation chapter also answered the first research question about which ubiquitous computing methods can identify contextual states of a smartphone user that are helpful at uncovering potential usability issues.

Subsequently, CIT was created according to the initially defined aims and fulfills the set requirements. It contains the functionality needed for logging contextual data from five components and additionally allows for measuring the usability metrics. It was implemented in Java programming language for the possibility to be used as an Android application module. The project code was refactored, is heavily commented and divided into separate files contributing to overall clarity. The library was tested in various situations and conditions and does not contain bugs or other unpredictable behavior.

The whole implementation was later described in detail, providing explanations of noteworthy code snippets and proposed usages in a sample application, as well as in a commercial mobile-pocket app. Afterwards, two conducted empirical studies and their results were reported, demonstrating the CIT's usefulness and the options it offers. More specifically, this confirmed how the solution is able to help better understand real behavioural patterns during the interactions in both supervised and unsupervised usability studies, since the context information from the session containing usability friction enables to better classify the cause of usability issues.

Finally, the generated datasets were analyzed with the help of the programmed general visualization tool, positively answering the second stated research question. Moreover, the provided tool created the same evaluation possibility to the future application developers that would decide to include CIT in their application. Benefiting from the depicted contextual parameters on a timeline, they might estimate the distraction level of their users or participants in usability studies and link interface issues to different kinds of circumstances.

6.1 Limitations

As aimed, the used detection algorithms prioritize low battery overhead over high detection accuracy, where possible. While the average user conditions measured by the toolkit correspond with reality, each device contains sensors with different precision and there are many edge cases in the contextual conditions. For instance, in environmental detection there are, apart from interior and exterior states, also grey zones, such as semi-outdoor passages and semi-indoor terraces. When considering activity detection, a user can be moving very slowly and irregularly in a queue in a shop, or in case of handedness they might constantly exchange thumbs when scrolling. This must be accounted for in evaluation and the collected data should be always compared to average values from all sessions. Furthermore, despite of the low computational requirements, the calculations done by CIT in the background naturally add to the total demands on the device's processor and battery. Hence, the cost-benefit ratio needs to be considered, regarding when the implemented library should be active, mostly in cases the application itself is meant to run for longer period at a time.

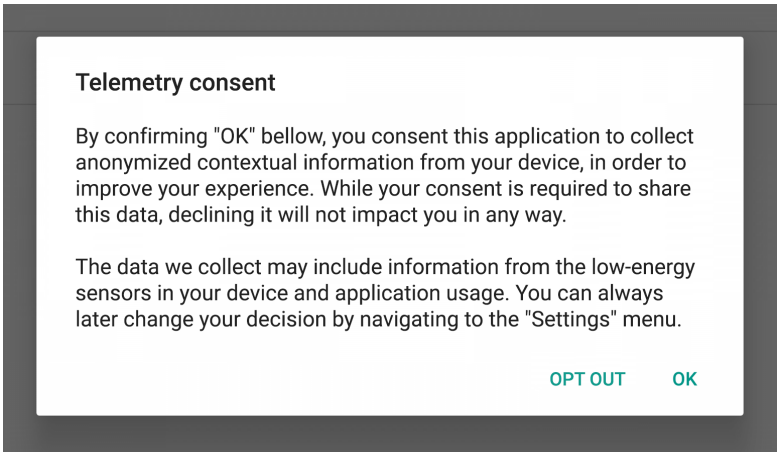


Figure 6.1: Example of a possible consent request dialog window in Android.

It has been shown what types of data CIT collects, what is more, since it is published open-source, this can be easily verified. In spite of that, the developers still need to make a decision, whether to limit the deployment to participants in user studies, or also profit from the datasets generated by the real users and in what scale. In the latter case, the users should be always informed about the statistical usage data collection by a telemetry warning (as in Figure 6.1) before the library is initialized, or before any data are sent to remote servers. Declining to give consent should disable the implemented library, while allowing for the application to continue running normally. Receiving information about the user base cheaply from the users themselves might be supported by various incentives and rewards.

6.2 Future Work

Even though the Context Information Toolkit supports the detection of five physical contextual types that were considered the most suitable also for their popularity in previous research, there are further identified context components, which might be interesting for the developers. Measuring all types of context would help to get an even bigger picture about users and may bring many novel implications, because as has been observed during this project, the contextual data are strongly interrelated.

Another step of future development could be detection accuracy improvement of the already included algorithms. Mostly in the field of machine learning many works promise better results than the simple decision trees. In case of successfully implementing a solution with low computational requirements during feature recognition, it might be lucrative alternative to the current implementation. Additionally, machine learning might bring new possibilities to easily recognize also higher level contexts, such as complex activities, environments, or ways of holding the device. It is argued that with sufficient advantages for the end-user and responsible approach from the developer, collecting only anonymous statistical data, this privacy trading would be still equally beneficial for both sides.

Similarly, measuring other usability metrics, such as long inactivity between actions could be added. Automatic friction pattern recognition is also a promising research direction that would enhance the developer experience. However, in this case the generality of the solution seems to be the biggest obstacle, considering the highly diverse application market.

For the applications that do not communicate with any remote servers it would be useful to provide a built-in feature of uploading the generated log files by the library to the cloud. This was out of the scope of this thesis, as excluding it significantly simplified the development, but in the future work it could be one of the most straightforward ways to improve the code's universality.

Besides the technical part of this project, future work could focus on further exploring the relationship of context and mobile devices. Although this work has referred to numerous research papers on the influence of context on users and their performance, there is currently still lack of research on its final impact on usability and countermeasures. Thus, for example a larger study comparing the two experiment conditions mentioned in this thesis should be conducted with a larger participant sample, in order to confirm some of the findings presented.

The provided visualization currently supports displaying only individual recorded sessions. Those are mainly interesting for UX teams, but by further preprocessing the data into statistically aggregated forms, the results could be more comprehensible for stakeholders. In the future work the visualization could get also other capabilities, such as zooming, panning, and scrolling to support its exploratory nature. Apart from that, an option could be added to manage the displayed source files and directly compare multiple separate sessions.

Finally, a major opportunity to extend the CIT's detection capabilities would be to incorporate signals from external devices, such as wearables. Fitness trackers, smartwatches, augmented reality glasses, or e-textiles with their built-in sensors could open new horizons for product usability improvement through physical context detection.

Appendix A

Algorithm Flow Charts

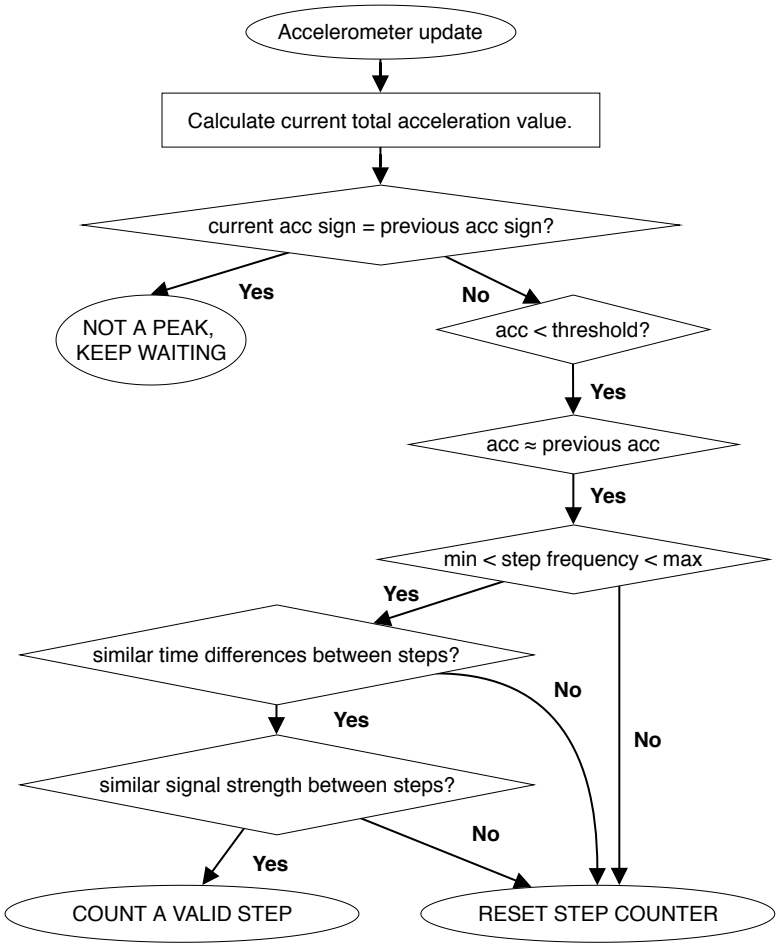


Figure A.1: Decision tree for step detection and counting. The algorithm is invoked by an update from the accelerometer sensor, the result given in the current number of identified steps. The shortcut 'acc' stands for the computed acceleration value.

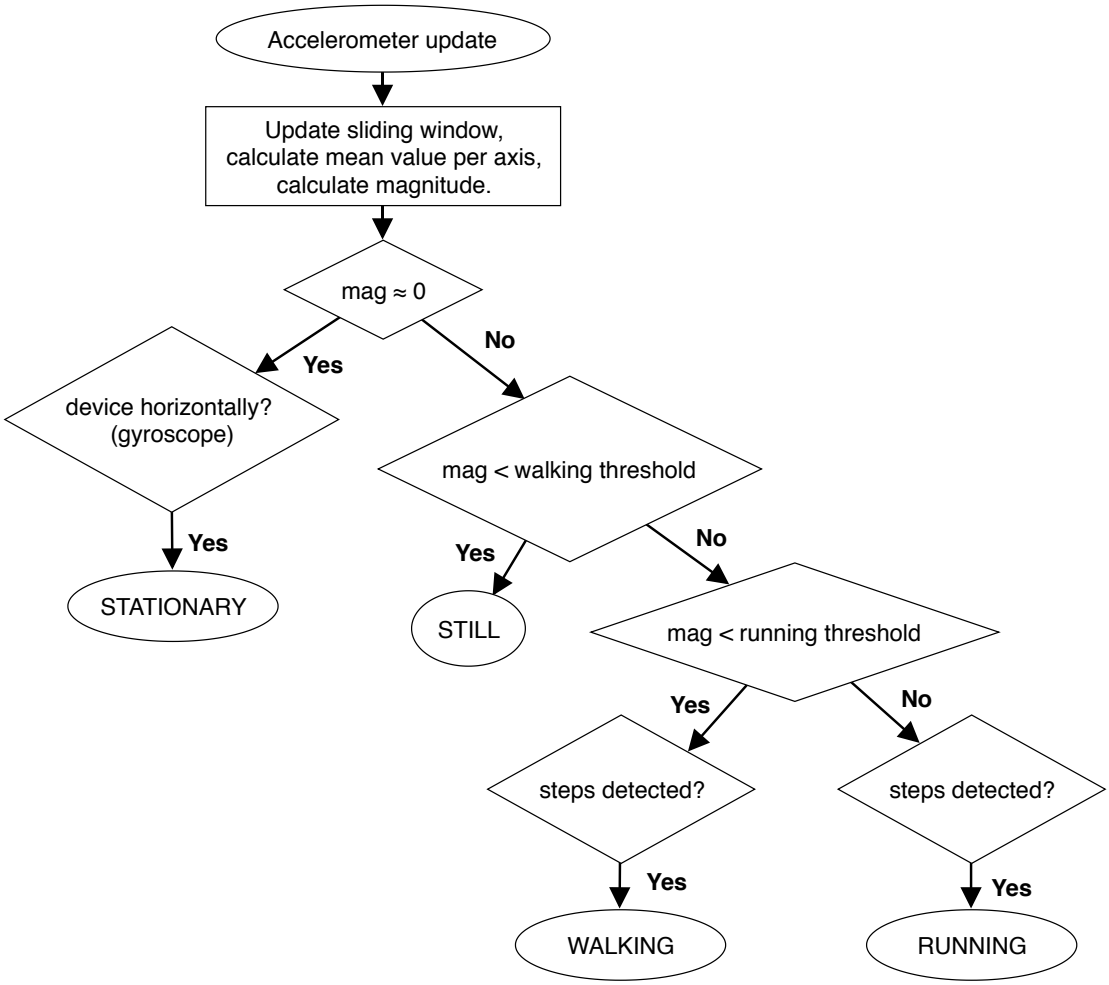


Figure A.2: Decision tree for activity detection. The algorithm is invoked by an update from the accelerometer sensor, the result is one of the activity states. The shortcut 'mag' stands for magnitude.

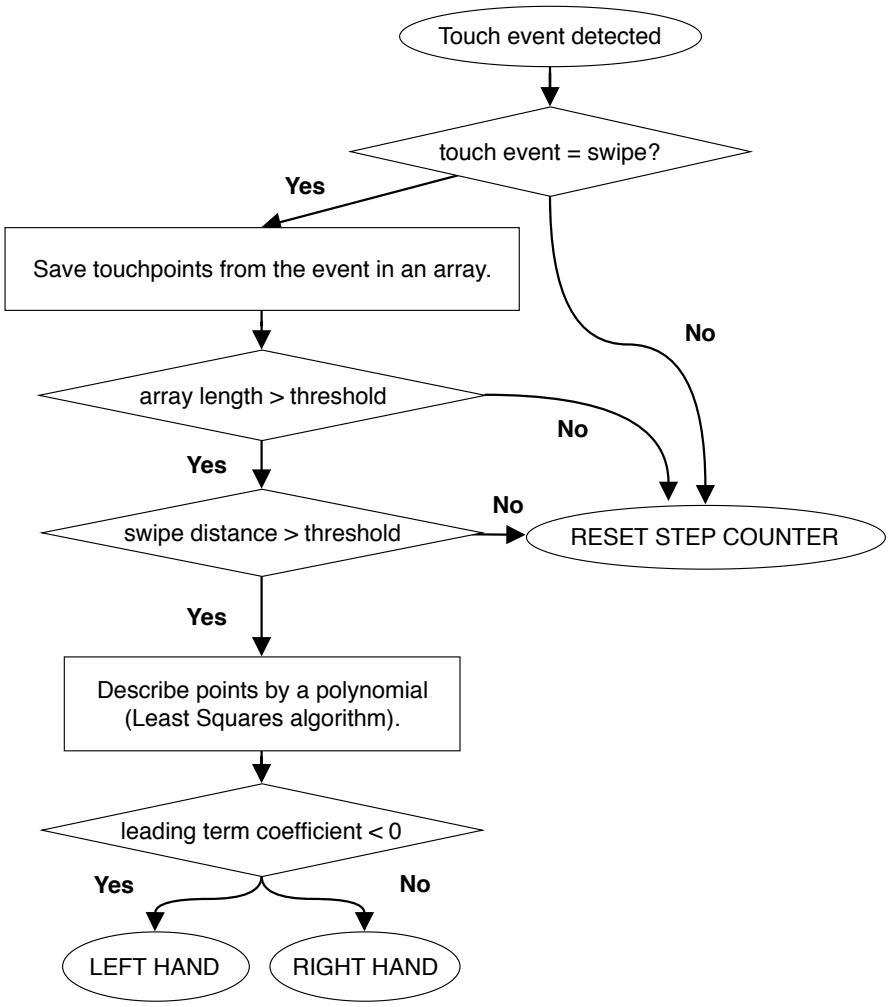


Figure A.3: Decision tree for hand detection. The algorithm is invoked every time a user touches the touchscreen and in case of a relevant swipe event results in a binary decision meaning either left or right hand.

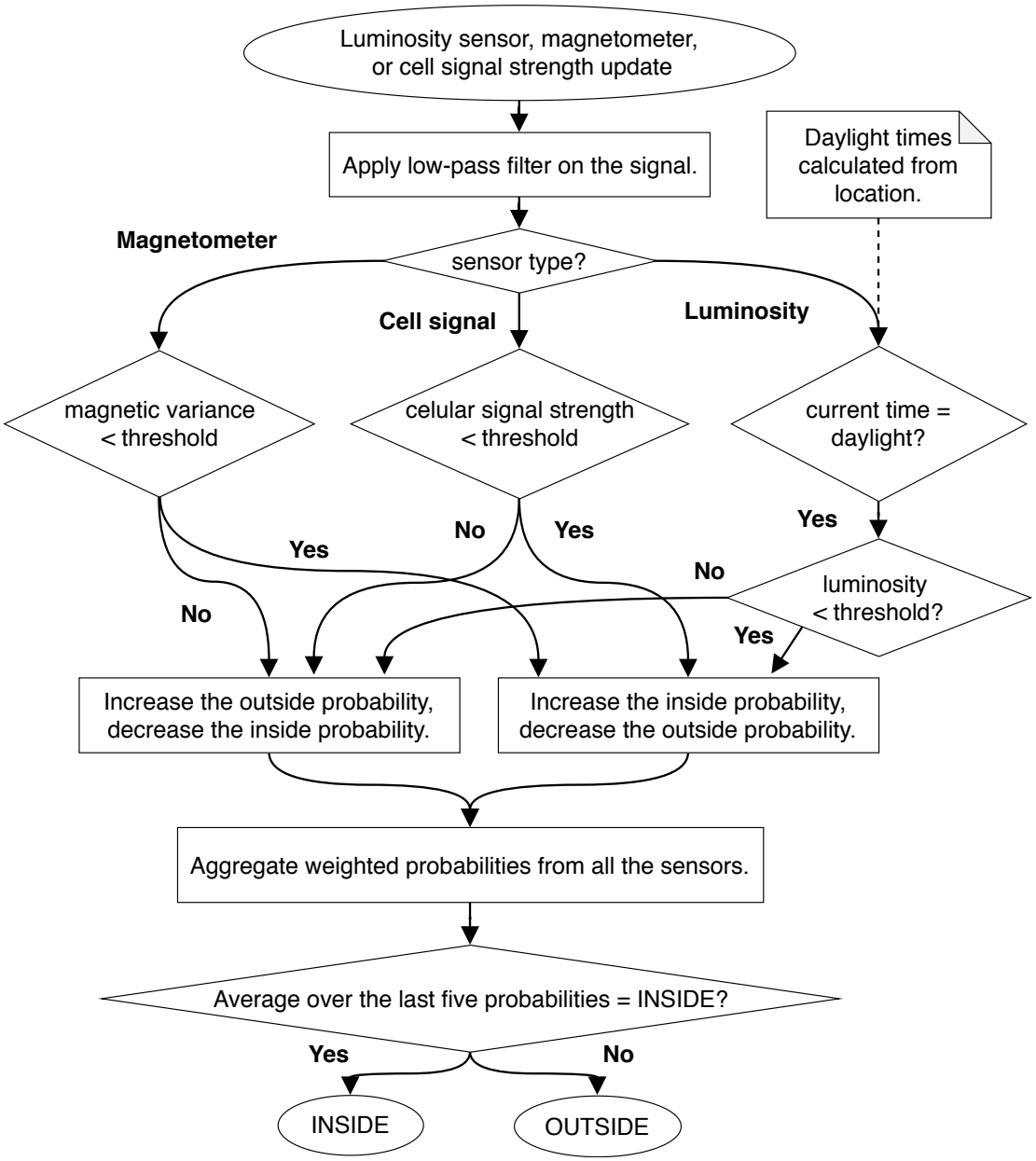


Figure A.4: Decision tree for environment detection. The algorithm is invoked every time one of the relevant sensors provides an update and results in a binary decision meaning either inside or outside condition.

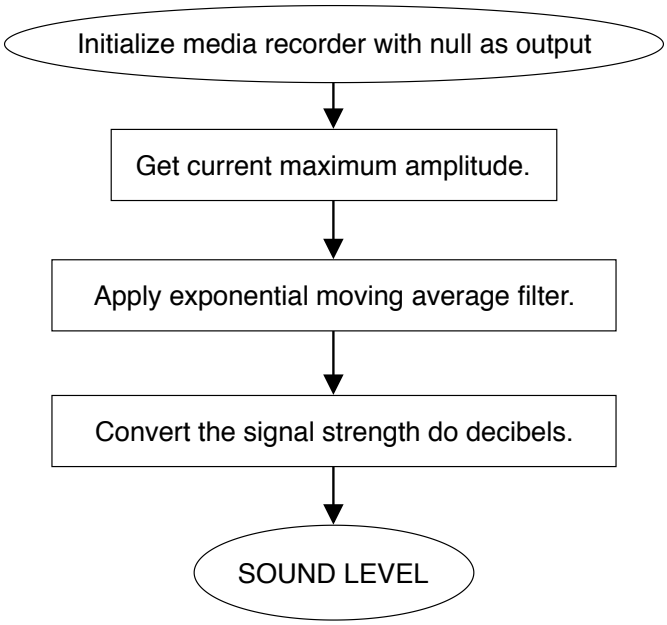


Figure A.5: Sound detection algorithm. The media recorder is polled once per second when active.

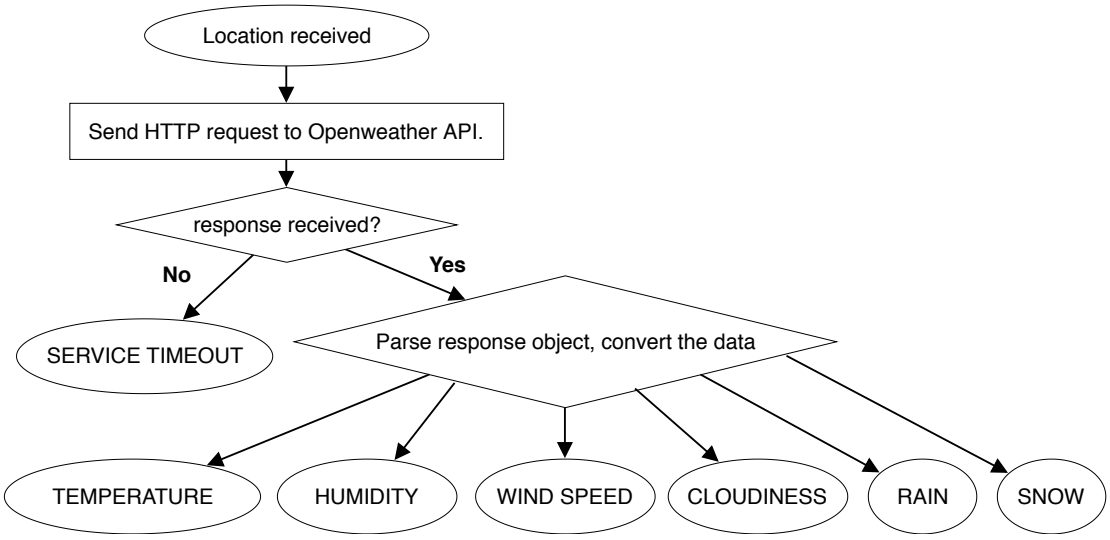


Figure A.6: Weather detection algorithm. The invocation happens after receiving the access to the user's location and results in a list of weather parameters.

Appendix B

CD-ROM Contents

B.1 Context Information Toolkit

```
/
└── Context Information Toolkit
    ├── Java source
    └── cit.aar ..... compiled Android library
```

This folder is comprised of the Android Studio project from the library development and a compiled version of CIT that can be implemented to any other Android application.

B.2 Demo Application

```
/
└── Demo application
    ├── Java source
    └── cit-demo.apk ..... compiled Android demo application
```

The compiled demo application developed to showcase the CIT's functionality and its source Android Studio project are included in this folder.

B.3 Datasets

```
/
└── Datasets
    ├── Pilot study datasets ..... csv data per session
    └── Experiment datasets
        ├── Sitting ..... csv data from sitting scenario per session
        └── Walking ..... csv data from walking scenario per session
```

In this Datasets folder are the generated experiment data in CSV file format. These are openable in a spreadsheet software, such as Excel, or in the produced visualization.

B.4 Visualization

/
- **Visualization**
  - **JavaScript source** ..... executable in a web browser

This folder contains the D3.js visualization source files, that can be opened in a web browser, either using a local server or by hosting the project online.

B.5 Thesis

/
- **Thesis**
  - LaTeX **source**
  - **thesis.pdf**

A digital copy of this thesis, as well as the source code written in LaTeX.

References

Literature

[1] Brady A. Thrift. "The Effects of Distraction on Usability Testing Results in a Laboratory Environment". Master Thesis. Guelph, Ontario, Canada: The University of Guelph, 2012 (cit. on p. 29).

[2] Gregory Abowd et al. "Towards a Better Understanding of Context and Context-Awareness". In: *Handheld and Ubiquitous Computing*. Ed. by Hans-W Gellersen. Vol. 1707. Lecture notes in computer science. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 1999, pp. 304–307 (cit. on p. 5).

[3] Mohsen Ali, Tamer ElBatt, and Moustafa Youssef. "SenseIO: Realistic Ubiquitous Indoor Outdoor Detection System Using Smartphones". In: *IEEE Sensors Journal*. Vol. 18, pp. 3684–3693 (cit. on p. 26).

[4] Theodoros Anagnostopoulos et al. "Environmental exposure assessment using indoor/outdoor detection on smartphones". *Personal and Ubiquitous Computing* 21.4 (2017), pp. 761–773 (cit. on p. 26).

[5] Alan Baddeley. "Working memory," *Applied Cognitive Psychology* 2.2 (1988), pp. 166–168 (cit. on p. 6).

[6] M. A. Baker and D. H. Holding. "The effects of noise and speech on cognitive task performance". *The Journal of general psychology* 120.3 (1993), pp. 339–355 (cit. on p. 6).

[7] Simon Banbury and Dianne C. Berry. "Disruption of office-related tasks by speech and office noise". *British Journal of Psychology* 89.3 (1998), pp. 499–517 (cit. on p. 29).

[8] Leon Barnard et al. "Capturing the effects of context on human performance in mobile computing systems". *Personal and Ubiquitous Computing* 11.2 (2007), pp. 81–96 (cit. on pp. 5, 6, 16, 18).

[9] Stefan Berti and Erich Schröger. "A comparison of auditory and visual distraction effects: behavioral and event-related indices". *Cognitive Brain Research* 10.3 (2001), pp. 265–273 (cit. on p. 29).

[10] Nigel Bevan and Miles Macleod. "Usability measurement in context". *Behaviour & Information Technology* 13.1-2 (1994), pp. 132–145 (cit. on p. 6).

[11] Igor Bisio, Alessandro Delfino, and Fabio Lavagetto. “Poster: Detecting if a Smartphone is Indoors or Outdoors with Ultrasounds”. In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys ’15*. Ed. by Gaetano Borriello et al. New York, USA: ACM Press, 2015, pp. 475–475 (cit. on pp. 2, 26).

[12] Agata Brajdic and Robert Harle. “Walk detection and step counting on unconstrained smartphones”. In: *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp ’13*. Ed. by Friedemann Mattern et al. New York, USA: ACM Press, 2013, pp. 225–234 (cit. on p. 20).

[13] Stephen Brewster. “Overcoming the Lack of Screen Space on Mobile Computers”. *Personal and Ubiquitous Computing* 6.3 (2002), pp. 188–205 (cit. on p. 6).

[14] Gianna Cassidy and Raymond A.R. MacDonald. “The effect of background music and background noise on the task performance of introverts and extraverts”. *Psychology of Music* 35.3 (2007), pp. 517–537 (cit. on p. 29).

[15] Doruk Coskun, Ozlem Durmaz Incel, and Atay Ozgovde. “Phone position/placement detection using accelerometer: Impact on activity recognition”. In: *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. IEEE, 2015, pp. 1–6 (cit. on p. 19).

[16] Constantinos K. Coursaris et al. “The impact of distractions on the usability and intention to use mobile devices for wireless data services”. *Computers in Human Behavior* 28.4 (2012), pp. 1439–1449 (cit. on pp. 6, 18).

[17] Jackson Feijó Filho, Wilson Prata, and Juan Oliveira. “Where-How-What Am I Feeling: User Context Logging in Automated Usability Tests for Mobile Software”. In: *Design, User Experience, and Usability: Technological Contexts*. Ed. by Aaron Marcus. Vol. 9748. Lecture notes in computer science. Cham: Springer International Publishing, 2016, pp. 14–23 (cit. on p. 11).

[18] Denzil Ferreira, Vassilis Kostakos, and Anind K. Dey. “AWARE: Mobile Context Instrumentation Framework”. *Front. ICT* 2 (2015), pp. 1–9 (cit. on pp. 9, 29).

[19] Gabriel Filios, Sotiris Nikoletseas, and Christina Pavlopoulou. “Efficient Parameterized Methods for Physical Activity Detection using only Smartphone Sensors”. In: *Proceedings of the 13th ACM International Symposium on Mobility Management and Wireless Access - MobiWac ’15*. Ed. by Mirela Sechi M.A Notare, Ángel Cuevas Rumín, and Miguel Lopez-Guerrero. New York, USA: ACM Press, 2015, pp. 97–104 (cit. on pp. 2, 19).

[20] Mayank Goel, Leah Findlater, and Jacob Wobbrock. “WalkType: Using Accelerometer Data to Accommodate Situational Impairments in Mobile Touch Screen Text Entry”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI 2012*. New York, USA: Association for Computing Machinery, 2012, pp. 2687–2696 (cit. on p. 18).

[21] Mayank Goel, Jacob Wobbrock, and Shwetak Patel. "GripSense: Using Built-In Sensors to Detect Hand Posture and Pressure on Commodity Mobile Phones". In: *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*. Ed. by Rob Miller. New York, USA: ACM, 2012, pp. 545–554 (cit. on p. 2).

[22] Morten Hertzum. "User Testing in Industry: A Case Study of Laboratory, Workshop, and Field Tests". In: *User interfaces for all*. Ed. by Alfred Kobsa and Constantine Stephanidis. Vol. GMD–74. GMD Report. Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 1998, pp. 59–72 (cit. on p. 6).

[23] Karin A. Hummel, Andrea Hess, and Thomas Grill. "Environmental context sensing for usability evaluation in mobile HCI by means of small wireless sensor networks". In: *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*. Ed. by Gabriele Kotsis. New York, USA: ACM Press, 2008, pp. 302–306 (cit. on p. 5).

[24] Karin Anna Hummel, Thomas Grill, and Andrea Hess. "On Context-Sensitive Usability Evaluation in Mobile HCI". *Journal of Mobile Multimedia* 5.4 (2009), pp. 351–370 (cit. on pp. 1, 29, 31).

[25] Sinisa Husnjak et al. "Identification and Prediction of User Behavior Depending on the Context of the Use of Smart Mobile Devices". In: *Proceedings of the 26th International DAAAM Symposium 2016*. Ed. by Branko Katalinic and Branko Katalinic. DAAAM International Vienna, 2016, pp. 462–469 (cit. on pp. 1, 2).

[26] Andrey Ignatov et al. "AI Benchmark: Running Deep Neural Networks on Android Smartphones". In: *Computer Vision – ECCV 2018 workshops*. Ed. by Laura Leal-Taixé and Stefan Roth. Vol. 11133. Cham, Switzerland: Springer, 2019, pp. 288–314 (cit. on p. 16).

[27] International Organization for Standardization, Geneva, Switzerland. *Ergonomics of Human-System Interaction – Part 210: Human-Centred Design for Interactive Systems*. 2010. URL: https://www.iso.org/standard/52075.html (cit. on pp. 1, 5).

[28] Dylan Jones. "Recent advances in the study of human performance in noise". *Environment International* 16.4-6 (1990), pp. 447–458 (cit. on p. 29).

[29] Satu Jumisko-Pyykkö and Teija Vainio. "Framing the Context of Use for Mobile HCI". *International Journal of Mobile Human Computer Interaction* 2.4 (2010), pp. 1–28 (cit. on pp. 5, 8).

[30] Eija Kaasinen. *User acceptance of mobile services: Value, ease of use, trust and ease of adoption*. Vol. 566. VTT Publications. Espoo: VTT, 2005, pp. 11–12 (cit. on p. 6).

[31] Xiaomin Kang, Baoqi Huang, and Guodong Qi. "A Novel Walking Detection and Step Counting Algorithm Using Unconstrained Smartphones". *Sensors* 18.1 (2018), pp. 1–15 (cit. on pp. 2, 20).

[32] Hoyoung Kim et al. "An empirical study of the use contexts and usability problems in mobile Internet". In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. Ed. by Ralph H. Sprague. 2002, pp. 1767–1776 (cit. on p. 5).

[33] Artur H. Kronbauer, Celso A. S. Santos, and Vaninha Vieira. “Smartphone Applications Usability Evaluation: A Hybrid Model and Its Implementation”. In: *Human-Centered Software Engineering*. Ed. by Marco Winckler, Peter Forbrig, and Regina Bernhaupt. Vol. 7623. LNCS sublibrary. SL 2, Programming and software engineering. Heidelberg: Springer, 2012, pp. 146–163 (cit. on pp. 1, 11).

[34] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. “Activity Recognition Using Cell Phone Accelerometers”. *ACM SIGKDD Explorations Newsletter* 12.2 (2011), pp. 74–82 (cit. on p. 19).

[35] Jakob Eg Larsen and Kristian Jensen. “Mobile Context Toolbox: An Extensible Context Framework for S60 Mobile Phones”. In: *Smart Sensing and Context*. Ed. by Payam Barnaghi. Vol. 5741. Lecture notes in computer science. Berlin: Springer, 2009, pp. 193–206 (cit. on p. 11).

[36] Jakob Eg Larsen et al. “Observing the context of use of a media player on mobile phones using embedded and virtual sensors”. In: *NordiCHI 2010: Extending Boundaries : Workshop on Observing the Mobile User Experience*. Ed. by Benjamin Poppinga et al. 2010, pp. 33–36 (cit. on pp. 1, 6).

[37] Florian Lettner and Clemens Holzmann. “Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications”. In: *MoMM2012*. Ed. by Ismail Khalil et al. ICPS: ACM international conference proceeding series. New York, USA: ACM, 2012, pp. 118–127 (cit. on p. 6).

[38] Florian Lettner and Clemens Holzmann. “Usability Evaluation Framework: Automated Interface Analysis for Android Applications”. In: *Computer Aided Systems Theory - EUROCAST 2011*. Ed. by Roberto Moreno-Dîaz, Franz Pichler, and Alexis Quesada-Arencibia. Vol. 6928. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 560–567 (cit. on p. 7).

[39] Min Lin et al. “How do people tap when walking? An empirical investigation of nomadic data entry”. *International Journal of Human-Computer Studies* 65.9 (2007), pp. 759–769 (cit. on p. 1).

[40] Jessica Ljungberg, Gregory Neely, and Ronnie Lundström. “Cognitive performance and subjective experience during combined exposures to whole-body vibration and noise”. In: *International Archives of Occupational and Environmental Health*. Vol. 77. 2004, pp. 217–221 (cit. on p. 18).

[41] Markus Löchtefeld et al. “Detecting users handedness for ergonomic adaptation of mobile user interfaces”. In: *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia - MUM ’15*. Ed. by Clemens Holzmann and René Mayrhofer. New York, USA: ACM Press, 2015, pp. 245–249 (cit. on p. 2).

[42] Yonggang Lu et al. “Towards unsupervised physical activity recognition using smartphone accelerometers”. *Multimedia Tools and Applications* 76.8 (2017), pp. 10701–10719 (cit. on p. 19).

[43] Neeraj Mathur, Sai Anirudh Karre, and Y. Raghu Reddy. "Usability Evaluation Framework for Mobile Apps using Code Analysis". In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018 - EASE'18*. Ed. by Austen Rainer, Stephen G. MacDonell, and Jacky Keung. New York, USA: ACM Press, 2018, pp. 187–192 (cit. on p. 2).

[44] E. M. Milic and D. Stojanovic. "Egosense: A Framework For Context-Aware Mobile Applications Development". *Engineering, Technology & Applied Science Research* 7.4 (2017), pp. 1791–1796 (cit. on p. 10).

[45] Emiliano Miluzzo et al. "CenceMe – Injecting Sensing Presence into Social Networking Applications". In: *Smart Sensing and Context*. Ed. by Gerd Kortuem. Vol. 4793. LNCS sublibrary. SL 5, Computer communication networks and telecommunications. Berlin and New York: Springer, 2007, pp. 1–28 (cit. on p. 10).

[46] Sachi Mizobuchi, Mark Chignell, and David Newton. "Mobile text entry: Relationship between Walking Speed and Text Input Task Difficulty". In: *MobileHCI 05*. Ed. by Manfred Tscheligi, Regina Bernhaupt, and Kristijan Mihalic. ACM International conference proceeding series. New York, USA: The Association for Computing Machinery, 2005, pp. 122–128 (cit. on p. 18).

[47] W. Moreno, O. Yurur, and C.-H. Liu. "Unsupervised posture detection by smartphone accelerometer". *Electronics Letters* 49.8 (2013), pp. 562–564 (cit. on p. 20).

[48] Kriti Nelavelli and Thomas Ploetz. "Adaptive App Design by Detecting Handedness" (2018), pp. 1–10 (cit. on p. 23).

[49] D. B. Nicholson et al. "Using Distraction-Conflict Theory to Measure the Effects of Distractions on Individual Task Performance in a Wireless Mobile Environment". In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. IEEE, 2005, pp. 1–9 (cit. on p. 6).

[50] Christian Monrad Nielsen et al. "It's worth the hassle!" In: *Mobile Human-Computer Interaction - MobileHCI 2004*. Ed. by Stephen Brewster and Mark Dunlop. Lecture notes in computer science, 0302-9743. Berlin: Springer, 2004, pp. 272–280 (cit. on p. 1).

[51] Marija Nikolic and Michel Bierlaire. "Review of transportation mode detection approaches based on smartphone data". In: *17th Swiss Transport Research Conference*. Ascona, Switzerland, 2017, pp. 1–18 (cit. on p. 19).

[52] Sonja Pedell et al. "Mobile Evaluation: What the Data and the Metadata Told Us". In: *Proceedings of the Australian Conference on Computer-Human Interaction - OZCHI '03*. 2003, pp. 96–105 (cit. on p. 6).

[53] Pengfei Zhou et al. "IODetector: A generic service for indoor outdoor detection". In: *Proceedings of the 10th ACM Conferenceon Embedded Network Sensor Systems - SenSys 2012*. New York, USA: Association for Computing Machinery, 2012, pp. 113–126 (cit. on p. 26).

[54] Valentin Radu et al. “A semi-supervised learning approach for robust indoor-outdoor detection with smartphones”. In: *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems - SenSys ’14*. Ed. by Ákos Lédecz, Prabal Dutta, and Chenyang Lu. New York, USA: ACM Press, 2014, pp. 280–294 (cit. on p. 26).

[55] Xukan Ran et al. “DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics”. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. Piscataway, NJ: IEEE, 2018, pp. 1421–1429 (cit. on p. 16).

[56] Sasank Reddy et al. “Determining transportation mode on mobile phones”. In: *ISWC 2008*. Piscataway NJ: IEEE Computer Society and IEEE, 2008, pp. 25–28 (cit. on p. 19).

[57] Virpi Roto. *Web browsing on mobile phones: Characteristics of user experience*. Vol. 49. TKK Dissertations. Espoo: Helsinki University of Technology, 2006 (cit. on p. 5).

[58] Jason Ryder et al. “Ambulation: A Tool for Monitoring Mobility Patterns over Time Using Mobile Phones”. In: *2009 International Conference on Computational Science and Engineering*. IEEE, 2009, pp. 927–931 (cit. on p. 19).

[59] Zhanna Sarsenbayeva et al. “Challenges of situational impairments during interaction with mobile devices”. In: *Proceedings of the 29th Australian Conference on Computer-Human Interaction - OZCHI ’17*. Ed. by Alessandro Soro et al. New York, USA: ACM Press, 2017, pp. 477–481 (cit. on p. 1).

[60] Zhanna Sarsenbayeva et al. “Effect of Distinct Ambient Noise Types on Mobile Interaction”. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2.2 (2018), pp. 1–23 (cit. on pp. 18, 29, 51).

[61] Nadav Savio and Jared Braiterman. “Design Sketch: The Context of Mobile Interaction”. *International Journal of Mobile Marketing* 2.1 (2007), pp. 66–68 (cit. on p. 5).

[62] Bastian Schildbach and Enrico Rukzio. “Investigating selection and reading performance on a mobile phone while walking”. In: *MobileHCI ’10 Proceedings of the 12th International Conference on Human computer Interaction with Mobile Devices and Services*. Ed. by Marco de Sá, Luís M. Carriço, and N. Correia. New York, USA: Association for Computing Machinery, 2010, pp. 93–102 (cit. on p. 51).

[63] B. N. Schilit and M. M. Theimer. “Disseminating active map information to mobile hosts”. *IEEE Network* 8.5 (1994), pp. 22–32 (cit. on p. 6).

[64] Albrecht Schmidt et al. “Advanced Interaction in Context”. In: *Handheld and Ubiquitous Computing*. Ed. by Hans-W Gellersen. Vol. 1707. Lecture notes in computer science. Berlin and London: Springer, 1999, pp. 89–101 (cit. on p. 20).

[65] Svenja Schröder, Jakob Hirschl, and Peter Reichl. “CoConUT- Context Collection for Non-Stationary User Testing”. In: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct - MobileHCI ’16*. Ed. by Fabio Paternò and Kaisa Väänänen. New York, USA: ACM Press, 2016, pp. 924–929 (cit. on p. 10).

[66] Andrew Sears et al. “When Computers Fade: Pervasive Computing and Situationally-Induced Impairments and Disabilities”. In: *Universal Access in HCI*. Ed. by Constantine Stephanidis. Human factors and ergonomics. Mahwah, N.J. and London: Lawrence Erlbaum, 2003, pp. 1298–1302 (cit. on p. 1).

[67] Muhammad Shoaib et al. “Resource consumption analysis of online activity recognition on mobile phones and smartwatches”. In: *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. Piscataway, NJ: IEEE, 2017 (cit. on pp. 16, 19).

[68] Brandon T. Taylor and V. Michael Bove. “Graspables: Grasp-Recognition as a User Interface”. In: *The 27th annual CHI conference on Human Factors in Computing Systems*. Ed. by Saul Greenberg. New York, USA: Association for Computing Machinery, 2009, pp. 917–925 (cit. on p. 23).

[69] Bettina Thurnher et al. “Exploiting Context-Awareness for Usability Evaluation in Mobile HCI”. In: *Proceedings of Usability Day IV*. Pabst Science, 2006, pp. 109–113 (cit. on pp. 2, 5).

[70] Alexandros S. Tsiaousis and George M. Giaglis. “An Empirical Assessment of Environmental Factors that Influence the Usability of a Mobile Website”. In: *Ninth International Conference on Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR), 2010*. Piscataway, NJ: IEEE, pp. 161–167 (cit. on pp. 1, 5, 29).

[71] Alexandros S. Tsiaousis and George M. Giaglis. “Evaluating the Effects of the Environmental Context-of-Use on Mobile Website Usability”. In: *Proceedings of 7th International Conference on Mobile Business (ICMB 2008)*. Barcelona, Spain, pp. 314–322 (cit. on p. 5).

[72] Heli Väätäjä. “Characterizing the Context of Use in Mobile Work”. In: *Human work interaction design*. Ed. by Jose Abdelnour-Nocera et al. Vol. 468. IFIP Advances in Information and Communication Technology, 1868-4238. Cham: Springer, 2015, pp. 97–113 (cit. on pp. 5, 8).

[73] K. van Laerhoven and O. Cakmakci. “What shall we teach our pants?” In: *The fourth international symposium on wearable computers*. 2000, pp. 77–83 (cit. on p. 19).

[74] Weiping Wang et al. “Indoor-Outdoor Detection Using a Smart Phone Sensor”. *Sensors* 16.10 (2016), pp. 1–15 (cit. on p. 26).

[75] Jacob O. Wobbrock, Brad A. Myers, and Htet Htet Aung. “The performance of hand postures in front- and back-of-device interaction for mobile computing”. *International Journal of Human-Computer Studies* 66.12 (2008), pp. 857–875 (cit. on p. 23).

[76] Zhibin Xiao et al. “Identifying Different Transportation Modes from Trajectory Data Using Tree-Based Ensemble Classifiers”. *ISPRS International Journal of Geo-Information* 6.2 (2017), pp. 57–79 (cit. on p. 19).

[77] Ji Soo Yi et al. “Context awareness via a single device-attached accelerometer during mobile computing”. In: *MobileHCI 05*. Ed. by Manfred Tscheligi, Regina Bernhaupt, and Kristijan Mihalic. ACM International conference proceeding series. New York, USA: The Association for Computing Machinery, 2005, pp. 303–306 (cit. on p. 29).

[78] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. “Deep Learning in Mobile and Wireless Networking: A Survey”. *IEEE Communications Surveys & Tutorials* (2019), pp. 1–67 (cit. on p. 16).

Online Sources

[79] Wikipedia contributors. *Dipole model of the Earth's magnetic field — Wikipedia, The Free Encyclopedia*. 2018. URL: https://en.wikipedia.org/wiki/Dipole_model_of_the_Earth%27s_magnetic_field (visited on 06/16/2019) (cit. on p. 27).

[80] Wikipedia contributors. *Low-pass filter — Wikipedia, The Free Encyclopedia*. 2019. URL: https://en.wikipedia.org/wiki/Low-pass_filter#Simple_infinite_impulse_response_filter (visited on 06/16/2019) (cit. on p. 27).

[81] Denzil Ferreira. *AWARE: OpenWeather plugin*. 2019. URL: https://github.com/denzilferreira/com.aware.plugin.openweather (visited on 05/10/2019) (cit. on p. 31).

[82] Interaction Design Foundation. *What is Cognitive Friction?* 2019. URL: https://www.interaction-design.org/literature/topics/cognitive-friction (visited on 05/07/2019) (cit. on p. 7).

[83] Google. *The Android Open Source Project: TwilightService.java*. 2019. URL: https://android.googlesource.com/platform/frameworks/base/+/kitkat-release/services/java/com/android/server/TwilightService.java (visited on 06/16/2019) (cit. on p. 27).

[84] James McCracken (https://stackoverflow.com/users/868492/james-mccracken). *Android 6.0 multiple permissions*. 2019. URL: https://stackoverflow.com/questions/34342816/android-6-0-multiple-permissions (visited on 05/21/2019) (cit. on p. 36).

[85] StatCounter. *Desktop vs Mobile Market Share Worldwide*. 2019. URL: http://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/#monthly-200901-201904 (visited on 05/03/2019) (cit. on p. 1).

[86] Research Group Mobile Interactive Systems. *automate toolkit*. 2019. URL: https://mint.fh-hagenberg.at/?page_id=579 (visited on 05/10/2019) (cit. on pp. 9, 17, 33).

[87] Karlsruhe Institute of Technology. *Privacy Friendly Schrittzähler App*. 2019. URL: https://secuso.aifb.kit.edu/Schrittzaehler.php (visited on 06/04/2019) (cit. on p. 20).