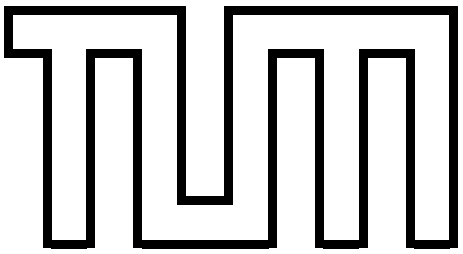

TUM SCHOOL OF MANAGEMENT

TECHNISCHE UNIVERSITÄT MÜNCHEN

MOTIUS GMBH

IDP report

Gesture Control with the Apple Watch Series 3

Gestensteuerung mit der Apple Watch Series 3

Author:	Michael Wang
Professor:	Prof. Dr. Nicola Breugst
Advisor:	Rieke Dibbern
Advisor (Motius):	Sören Gunia
Submission Date:	March 28th 2018

I hereby declare that this IDP report is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the report in digital form can be examined for the use of unauthorized aid and in order to determine whether the report as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future reports submitted. Further rights of reproduction and usage, however, are not granted here.

This report was not previously presented to another examination board and has not been published.

Munich, March 28th 2018 Michael Wang

Abstract

The research and development of robust systems for gesture recognition and control is expensive. Another problem is that companies often struggle to find good use cases for gesture control in their existing products. In conclusion, a cheap and yet effective way of evaluating such potential business ideas is crucial for R&D.

The purpose of this IDP report is to evaluate to which extent the *Apple Watch Series 3* can be used in prototyping ideas for gesture control.

We first select a use case from the automotive industry. The goal is for a user that is driving a car to accept or deny an incoming call with gesture control.

Afterwards, we develop a working prototype. For this purpose, we implement a way to record the raw sensor data of a gesture. A set of 734 gesture recordings is then created and this set is used to train a machine learning model.

Additionally, we evaluate this prototype in an empirical case study. In this study 500 random gesture attempts are recorded and evaluated. The total *False Rejection Rate (FRR)* is 12.6%. The *FRR* without *NoGesture* where gestures that were not recognized do not count is 2.2%. Furthermore, we observe a learning effect. The *FRR* for the *RightSwipe* gesture improves from 63.6% to 10.0% between the first 100 attempts and the last 100 attempts.

Finally, we rate how easy the integration of our prototype into an existing app is. On a subjective rating scale we rate this a full 10 out of 10.

In conclusion, we present a gesture recognition library that is reasonably accurate and that can be easily integrated into existing apps. We suggest that the *Apple Watch* can be used by companies to quickly and cheaply prototype their gesture recognition and control ideas.

Contents

1. Introduction

Human-computer interaction is a field that allows for a wide range of research. One such area is gesture recognition and control. Its applications are "manifold, ranging from sign language through medical rehabilitation to virtual reality" [10].

There are multiple ways to realize gesture recognition. Approaches include simple 2D-image recognition, 3D-capturing, and the use of hand-held controllers, such as the ones used in virtual reality systems. These approaches provide very accurate data if implemented well, but are also expensive to research and develop.

Wearable devices, such as smart watches or fitness armbands, can also be used for gesture recognition. Developing a prototype with these devices is cheaper compared to the previously named approaches. This key feature is crucial to this IDP: When a company wants to research which effect gesture control can have on their business, it is best to first build a cheap prototype, such as with wearable devices. A company could thus get a feeling for a future system that uses a more advanced form of gesture control. The success of such a cheap prototype can then be leveraged to invest more into researching better forms of gesture control.

1.1. Motivation

We research an easy and cheap way to quickly prototype business ideas for gesture recognition. This is done in two collaborations: First, with *Motius GmbH* [11], a R&D company. Secondly, with an automotive company.

In this project we explicitly do not focus on the market-ready variant of gesture recognition which could include multiple cameras costing millions of Euro to develop. What we are interested in instead are the new use cases that would be possible with such an advanced system. These new use cases should then be easy to implement into a first prototype of the advanced version, so that the idea can be evaluated.

Use Case

In collaboration with both partners the following use case was selected:

Michael is driving his car. The car has the steering wheel on the left-hand side. Michael also owns an *Apple iPhone* and is wearing an *Apple Watch* on his right hand. The watch is paired with the phone and the phone is connected to the car. Michael gets a call on his phone via the internet, for example with *Skype*. Michael can now do a gesture with his right hand to accept or to deny the call.

Goal

One of our goals is to evaluate this use case. For example, we could evaluate which gesture works best or how the use of gesture recognition feels inside a car.

The focus of this IDP is however on a higher level of abstraction: We evaluate whether we can improve the use case evaluation itself. For example, if we use an Apple Watch as a cheap prototype solution, can we still reliably recognize gestures?

1.2. Approach

With this goal in mind, we implement a gesture recognition prototype for our use case. For this purpose we need to implement a way to record the raw data of a gesture from the Apple Watch, gather a set of training data, and then train a machine learning model. Afterwards, this prototype is evaluated in an empirical case study. In this study we answer whether the prototype was implemented well, for example if it recognizes the gestures it is supposed to. We also answer to which extent our prototype fulfills our goal.

1.3. Report Outline

In this report we present a library for gesture control with the Apple Watch Series 3. In Chapters 1 and 2 we explain why this is of interest and explain some background knowledge. We then implement a prototype in Chapter 3. This prototype is evaluated in an empirical case study in Chapter 4. Finally, in Chapter 5, we summarize the learnings of this IDP and present future work.

2. Fundamentals

This chapter introduces background knowledge for subsequent chapters. We begin with an overview on the *Apple Watch Series 3*. Afterwards, we introduce the *Core ML* framework.

2.1. Apple Watch Overview

The *Apple Watch* is a line of smartwatches. Its current third-generation model, the *Apple Watch Series 3*, was released in Germany in September 2017 starting at 399 Euro.

In combination with an *iPhone*, the watch allows for fitness and health tracking. For this purpose it is equipped with various sensors such as: GPS, barometric altimeter, heart rate sensor, accelerometer, gyroscope, and ambient light sensor [7].

In this report we use an *Apple Watch Series 3 42mm (GPS)*, model *MQL12ZD/A*.

2.2. Core ML Framework

Core ML is a framework that allows us to integrate a trained machine learning model into our app. Apple describes this as follows: "A trained model is the result of applying a machine learning algorithm to a set of training data. The model makes predictions based on new input data. For example, a model that's been trained on a region's historical house prices may be able to predict a house's price when given the number of bedrooms and bathrooms." [4]

Similarly, our goal is to recognize a user's individual gesture. A model must first be trained on a specific gesture. Afterwards, the trained model should then be able to recognize that specific gesture.

Core ML was officially released in September 2017 and is only available for the following platforms: *iOS 11.0+*, *macOS 10.13+*, *tvOS 11.0+*, and *watchOS 4.0+* [4].

3. Implementation

In this chapter we describe the implementation of a Watch app that can recognize gestures. We first explain how to record gestures with an Apple Watch. Afterwards, we define the gestures to be recognized. These gestures are then recorded to create a set of training data. Finally, this training data is used to train a machine learning model and the result is integrated into the app.

3.1. Gesture Recording

This section explains how to record a gesture with an Apple Watch. We first describe how to get access to the raw sensor data when the watch is active. We then present a workaround on how to get *continuous* access even when the watch is inactive.

Accessing Sensor Data

The first step in this project was to read out sensor data from the watch. In Section 2.1 we already described the sensors in the watch. For gesture recognition, we are especially interested in the data provided by the accelerometer and the gyrometer.

The class `CMMotionManager` [2] allows us to "start the services that report movement detected by the device's onboard sensors" [2]. Implementing this class allows us to receive both `CMAcceleration` and `CMRotationRate` objects. These contain the raw x, y, and z sensor values. This is how we receive raw sensor data.

Continuous Access: Workout Mode

There is another problem to tackle: For battery-saving purposes Watch apps "run only while the user interacts with one of their interfaces" [1]. This means that our app can only receive sensor data when the watch screen is active, i.e. when the user

has raised his wrist in such a position where he can see the watch display. Conversely, we can not receive sensor data when the user lowers his wrist as the watch screen will turn off. This is known as *foreground execution mode*.

In order to run the Watch app in *background execution mode*, we can abuse one exemption to the general rule that Watch apps are considered foreground apps: The class `HKWorkoutSession` [5]. It is intended for when a user wants to do a workout, i.e. goes jogging, cycling, and so on. Apps designed to track such a workout would use a *Workout Mode* provided by `HKWorkoutSession` to "continue to access data from Apple Watch's sensors in the background" [5]. While our gesture control app has nothing to do with workouts, we can use this exemption as a workaround. This incurs several drawbacks, such as increased battery drain. Furthermore, our app is no longer compliant to the Apple AppStore, so we would not be able to use this workaround in practice. For a prototype however, this workaround is sufficient to continuously read out sensor data.

Conclusion

Using `CMMotionManager` we read out raw sensor data and write both accelerometer and gyrometer data to a file. We can record these values continuously using the *Workout Mode* provided by `HKWorkoutSession`. In conclusion, we are now able to record the sensor data of individual gestures.

3.2. Training Data

The purpose of this section is to show how we gathered training data. This is necessary to teach a machine learning model to recognize gestures. We start by describing and examining the individual gestures. Afterwards, we evaluate these gestures and select a final list of gestures to recognize. Finally, we record these selected gestures to create a set of training data.

Gesture Examination

In Section 3.1 we explained how to record the data needed to recognize gestures. The next step in the project is to examine which gestures we want to actually recognize.

Whether a gesture makes sense is not only dependent on how well we can recognize it, but also on whether it makes sense in the context of the use case. In a brainstorming session with colleagues, we defined the following gestures to be examined:

SingleFlick Flick your wrist once by 180° degrees.

DoubleFlick Do *SingleFlick* twice in quick succession.

SingleTap Tap hardly (or clap lightly) on your thigh, one time only.

DoubleTap Do *SingleTap* twice in quick succession.

LeftSwipe Swipe your wrist in the air to the left, along a horizontal axis.

RightSwipe Swipe your wrist in the air to the right, along a horizontal axis.

Phone Raise your wrist towards your ear and make a "Call-Me" gesture.

Evaluation

With these seven gestures defined, we proceed to evaluate them. For this evaluation we recorded each gesture and used that data as input for a machine learning model. This was an iterative process, starting with a few recordings for each gesture. Later, we increased the input data to about twenty to thirty recordings for each gesture, thereby increasing the accuracy of our machine learning model. In this process we discarded three gestures and added one:

SingleFlick was discarded for being too inaccurate. It was for example very easy to accidentally flick your wrist more than once. Alternatively it was too easy to over- or under-flick, i.e. not reach 180° degrees. In conclusion, it seems that there was not enough distinguishable data for a *SingleFlick* from the baseline or from other gestures in our simple model.

SingleTap was discarded for similar reasons. We conclude that there was also not enough distinguishable data in our prototype model.

Phone was discarded because it was not a good fit with our use case. This gesture

teaches the user to move their wrist powerfully in an upwards vertical direction. In our practical evaluation, users would sometimes instinctively move the steering wheel with their other hand, in an counterbalancing downwards direction. This is clearly dangerous and therefore the *Phone* gesture was discarded. With this in mind, a similar argument could be made for the *DoubleTap* gesture. However, in practice, that gesture does not seem to induce dangerous behaviour.

NoGesture was added as a baseline for movements we do not want to recognize, such as random vibrations or small wrist movements. This increases the detection accuracy for movements we actually want to recognize as gestures.

In-Car Recording

Five gestures remain to be recorded and to train our machine learning model: *DoubleFlick*, *DoubleTap*, *LeftSwipe*, *RightSwipe*, and *NoGesture*. For consistency, we recorded all gestures in a real car that was parked. Additionally, the watch was to be worn on the right wrist and both hands were to be placed on the steering wheel. The exact starting position of the right hand was allowed to be varied, but only between 12 o'clock and 6 o'clock on the steering wheel. In total, we recorded 734 gestures with the distribution presented in Table 1.

	Recordings
DoubleFlick	142
DoubleTap	147
LeftSwipe	173
RightSwipe	186
NoGesture	86
Sum	734

Table 1: Distribution of Recordings

Conclusion

We first examined seven possible gestures. We then evaluated these and discarded three of those seven. We also added a *NoGesture* gesture. Of these five remaining gestures, we then took 734 recordings. In conclusion, we have now recorded a set of gestures which we can use to train a machine learning model.

3.3. Machine Learning

This sections shows how we integrate a trained machine learning model into our app. First, we apply a machine learning algorithm to our set of training data obtained in Section 3.2. This is done with the *Keras* framework. Afterwards, we convert the resulting model into the *Core ML* model format for integration into our app.

Keras

Before we can recognize gestures, we need to train a machine learning model first. For this we selected the neural network library *Keras*. Our selection in this decision was limited to frameworks supported by *Core ML*.

Figure 1 shows the implemented network architecture. This was conceived in many iterative rounds of feedback, with continuous improvement on the accuracy of detection, and with the help of a colleague. Describing this entire process is unfortunately out of scope for this report.

We now have a trained machine learning model that can detect gestures. How accurate this detection is will be evaluated in Chapter 4.

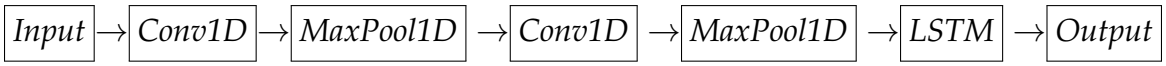

Figure 1: *Keras* network architecture

CoreML

Finally, we need to integrate the trained machine learning model into the Watch app. Therefore, we must first convert our *Keras* model to the *Core ML* model format `.coreml`. This can be achieved with various converters [3]. Afterwards, integration into the app is trivial [6]. We selected a detection threshold of 70%.

Conclusion

We first used *Keras* to train a machine learning model. Afterwards, we converted the trained model to the *Core ML* model format and integrated it into the app. The app can now input live sensor data into the model and receive the ID of a gesture.

4. Evaluation

This chapter evaluates the prototype implemented in Chapter 3. We first explain the design of this study and formulate research questions. We then describe the study procedure. Afterwards, we answer the research questions and present the study results. Finally, we discuss these results.

4.1. Study Design

In this section we describe the design of this empirical case study, which is based on the Goal-Question-Metric approach [8].

Goal

The goal is to evaluate the implementation described in Chapter 3. First, we determine whether our prototype was implemented well enough to detect the gestures it is supposed to recognize. Secondly, we want to know if our prototype is sufficient to fulfill our use case.

Questions

In order to achieve these goals, we formulate the following research questions (RQ):

RQ1 **How accurate is the implemented gesture recognition system?**
This question evaluates the implementation quality of the prototype.

The optimum would be to have perfect gesture recognition, i.e. gestures must be recognized and non-gestures must not be recognized. Unfortunately, the implementation of such a precise system is unrealistic so we want to evaluate to which extent our prototype can approximate a perfect system.

RQ2 How easy is the integration into an existing app?
This answer to this question analyzes how well our prototype can be used in practice.

If the integration of our prototype into Watch apps was rather easy, then existing apps could easily be enhanced with gesture recognition capabilities. This means that our prototype would not be specific to a single industry, but could be applied in many.

Metrics

The following metrics are used to answer the research questions:

RQ1 How accurate is the implemented gesture recognition system?
We measure the *False Rejection Rate (FRR)* of our implemented system.

Possible values range between 0% and 100%, calculated by the ratio of correct gestures that were not recognized to the total amount of correct gestures. A lower ratio implies a more robust system. A higher ratio implies comfort issues, as users would for example have to do a gesture twice before it is recognized.

The *FRR* does not have security implications since we do not measure whether incorrect gestures get accidentally recognized - this would be the *False Acceptance Rate* which is out of scope for this IDP.

RQ2 How easy is the integration into an existing app?
We rate this on a subjective rating scale between 0 and 10.

The lowest end of this scale (0) represents a gesture recognition system that was implemented for a specific project and can not be applied to other projects or industries.

The highest end of this scale (10) represents a gesture recognition system that is highly compartmentalized and can be used as a library in any other project or industry.

4.2. Study Procedure

In this section we explain the study procedure for both RQ1 and RQ2.

RQ1: How accurate is the implemented gesture recognition system?

RQ1 evaluates the *False Rejection Rate (FRR)* of our system. For this purpose we conducted a systematic trial of 500 attempts. Each attempt was conducted according to the following three steps:

First of all, one of our four gestures was randomly selected and communicated to the tester. This person is knowledgeable about the system and has been trained on the use of the system.

Secondly, the tester then did his best to execute the randomly chosen attempt. If a tester did this gesture incorrectly it would not count as a correct attempt. This is because we only want to measure the *FRR*. We are not interested in what happens if incorrect gestures are applied to the system, which would be out of scope.

Finally, the tester recorded the gesture recognized by the system. If nothing happened, he recorded that no gesture was recognized.

In conclusion, 500 attempts with two data points were recorded, which is shown in Appendix A. The first data point is the gesture that was executed. The second is the gesture that was actually recognized. The goal is to match these two data points in as many attempts as possible.

RQ2: How easy is the integration into an existing app?

RQ2 analyzes how well our prototype can be used in practice and is rated on a subjective rating scale between 0 and 10. For this purpose we simply assess this value manually.

Ideally multiple researches would also assess this question, separate from each other. We could then calculate Cohen's kappa coefficient [9] to measure the inter-rater agreement between these raters. If ratings were inconsistent, the differences could have been discussed and an agreement on the correct result could have been reached. Unfortunately this was also out of scope.

4.3. Results

This section presents the results of the study which will be discussed in Section 4.3.

RQ1: How accurate is the implemented gesture recognition system?

We first aggregate all 500 attempts in Table 2. We then break down the data set further and detail the first 100 attempts in Table 3 and the last 100 attempts in Table 4. In these tables, we define FRR_N as the *FRR* without *NoGesture*.

		Recognized Gesture							
		Flick	**Tap**	**Left**	**Right**	**None**	**Sum**	**FRR**	**FRR_N**
Intended Gesture	**Flick**	109	1	0	0	21	131	16.8%	1.0%
	Tap	0	112	0	0	17	129	13.2%	0.0%
	Left	0	1	118	0	4	123	4.1%	0.8%
	Right	0	0	8	98	11	117	16.2%	7.5%
							500	12.6%	2.2%

Table 2: Distribution of all 500 attempts

		Recognized Gesture							
		Flick	**Tap**	**Left**	**Right**	**None**	**Sum**	**FRR**	**FRR_N**
Intended Gesture	**Flick**	21	0	0	0	0	21	0.0%	0.0%
	Tap	0	30	0	0	2	32	6.4%	0.0%
	Left	0	1	21	0	3	25	26.0%	4.5%
	Right	0	0	8	8	6	22	63.6%	50.0%
							100	20.0%	10.1%

Table 3: Distribution of first 100 attempts

		Recognized Gesture							
		Flick	**Tap**	**Left**	**Right**	**None**	**Sum**	**FRR**	**FRR_N**
Intended Gesture	**Flick**	21	1	0	0	7	29	27.6%	4.1%
	Tap	0	23	0	0	4	27	14.8%	0.0%
	Left	0	0	22	0	0	22	0.0%	0.0%
	Right	0	0	0	20	2	22	9.1%	0.0%
							100	14.0%	1.1%

Table 4: Distribution of last 100 attempts

RQ2: How easy is the integration into an existing app?

We rate this metric a full 10. Our reasoning is as follows: Given a trained machine learning model, integration into Watch apps is trivial, as seen in Section 3.3. It follows that our implementation can be very easily integrated in any Watch app regardless of the project or the industry. Since all of the input is provided by the Watch, the app has to only decide what to do with the recognized gesture. In summary, this implementation is an easily integratable gesture recognition library.

4.4. Discussion

In this section we discuss the results obtained in Section 4.3.

RQ1 shows a total *FRR* of 12.6%, i.e. 437 of 500 gestures were recognized correctly. Furthermore, the total FRR_N, i.e. the *FRR* without *NoGesture*, is 2.2%! When our prototype made an error, it was much more common for it to simply not recognize any gesture than to identify the wrong gesture.

Out of all gestures, the *RightSwipe* gesture was the most problematic. In the first 100 attempts, a 63.6% were recognized wrongly. In the last 100 attempts, the error rate was dramatically reduced to 10.0%. This is due to learning effects. The *RightSwipe* is the gesture with the most freedom of movement inside a car and it takes a while for the tester to learn it. This learning effect can have multiple implications: First, it suggests a way of improvement for the accuracy of the entire system as for example a self-reinforcing learning system of the user's habits would likely increase the accuracy. Secondly, it means that for the current implementation user's have to be taught on how to use this system in a short tutorial.

In general, we are very pleased with the accuracy of the system given the boundaries of this evaluation.

RQ2 was rated a full 10 out of 10. We are very satisfied with Apple's *Core ML* framework and the possibilities of integration. We could integrate the implemented gesture recognition library into any Watch app and it would simply work.

In conclusion, our implementation fulfills the goals of Section 4.1. It is accurate enough to recognize the gestures we defined, as its *FRR* is 12.6% and its FRR_N is 2.2%. It can also fulfill the use case well, because integration into this and other projects is trivial.

5. Learnings and Future Work

The goal of the IDP was to evaluate the extent to which the Apple Watch can be used for gesture recognition prototyping. We will first summarize the learnings of this project. Afterwards, we present future work.

5.1. Learnings

The learnings of this project consist of three main results.

First of all, we learned how to implement gesture recognition with the Apple Watch in Chapter 3. We, for example, had to overcame technical problems like the *foreground execution mode* challenge faced in Section 3.1. We also took our first steps with the newly released *Core ML* framework, such as how to convert a trained machine learning model into the *Core ML* format and how to integrate the result into the Watch app.

Secondly, we learned that our prototype was accurate enough for our use case, as described in the empirical case study of Chapter 4. For this purpose 500 random attempts were recorded and evaluated. In conclusion, the total *False Rejection Rate (FRR)* was 12.6%. The *FRR* without *NoGesture* was 2.2%, so when our prototype made an error, it was much more common for it to simply not recognize any gesture than to identify the wrong gesture. Furthermore, we observed a learning effect. The *FRR* for the *RightSwipe* gesture improved from 63.6% to 10.0% between the first 100 attempts and the last 100 attempts. We discussed the implications of this in Section 4.4.

Thirdly, we learned that our prototype can be easily integrated into any Watch app. On a subjective rating scale, we rated it a full 10 out of 10. Since all input is provided by the watch, an app has to only decide what to do with the recognized gesture. In summary, we implemented an easily integratable gesture recognition library.

5.2. Future Work

Future work could be to improve the implementation by enhancing the machine learning algorithms. The implementation could also be improved with an increased training set of people, gestures, and more cars.

Furthermore one could improve the evaluation by adding more test case attempts or by testing more metrics, such as the *False Acceptance Rate*. If one were to test multiple detection thresholds, one could also evaluate the *Equal Error Rate*.

A. Evaluation Raw Data

This is a list of 500 attempts XY, where X is the gesture to be recognized and Y is the gesture that was actually recognized. X and Y can be:

F: DoubleFlick, **T**: DoubleTap, **L**: LeftSwipe, **R**: RightSwipe, **N**: NoGesture

Raw Data

TT LN FF TT TN TT FF TT LL TT FF RL FF TT TT TT LL LL RL TT TT TT LL LL LL LL RR LL TT LL LL RR LL TT FF TN FF LL LL TT TT RR LL LN RL LL TT TT FF RN LL RN FF FF RN FF FF LL RN RN TT RL TT FF FF FF RR RL TT TT FF RL RL RR LT TT RR FF FF FF RR FF LN TT TT TT TT TT TT RN LL RR RL LL FF TT LL TT LL FF TT FF TT FF FF FF LL LL FF RN TN LL FF TT FF TT FF LL FF LL TT RR TT FF RR FF TT RR LL RR RR RR RR LL TT RR RR FF RR TT RR FF FN FN RR RR LL LL FF TN LL LL FF RR TT FF TN RR TT LL LL TT RR TT LL RR RR LL TT TT TT LL TT TT RR RR TT RR LL LL TT LL LL FF LL FF LL LL FF FF FF FF RR FN LL LL RR LL RR RR LL LL TT LL RN TT LL TN RR FF TT FF LL RR RN TT FF FF TT TT RR TN FF TT LN FN RR FF RR LL TN RR FF LL LL TT TT FF FF TT FF LL LL FF RR FN FF TT TT RR RR FF FF TT TT TT FF TN LL FF TT TN LL FF LL TT RR RR LL FF TN LL FF LL FF FN LL LL RR LL RR LL FF FF RR FF TT FF LL LL RR RR RR RR LL RR FN TT LL LL LL TT LL LL FF FN RR RR RR RR TT TT FF FF FN RR TT FF RR RR RR RR LL RR FN FN LL RR TT RR TT LL FN LL TT FF TT FF TT TT RR LL TT LL RR FF TT RR RR RR TT LL FF TT TT FF FF FF LL LL LL RR LL TT FF FN RR TT RR FF LL LL TT LL TT TT TN RR FF FF FF FF FF LL FF LL RR RR LL LL LL FN RR RR TN RR LL LL FF FF FN TN FF LL RR RR RR TT FN FF FF FF TT LL LL FF RR TN LL FF FF RN RN TN TT TT LL TT TT TT TT TT TT TT TT FN LL RR LL RR TT LL FF FF FN RR FF TT FF RR LL RR FF FF RR TT FF TT RR FF FF LL RR TT RR FF TT TT LL LL RR TN LL LL LL LL LL FT RR RR FF FN RR LL LL TT RR FN TT FF LL LL RR LL RR TT FF FN FF TT

Bibliography

[1] Apple. *App Programming Guide for watchOS: Leveraging iOS Technologies*. 2018. URL: https://developer.apple.com/library/content/documentation/General/Conceptual/WatchKitProgrammingGuide/iOSSupport.html (visited on 03/28/2018).

[2] Apple. *Apple Developer Documentation: CMMotionManager*. 2018. URL: https://developer.apple.com/documentation/coremotion/cmmotionmanager (visited on 03/28/2018).

[3] Apple. *Apple Developer Documentation: Converting Trained Models to Core ML*. 2018. URL: https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml (visited on 03/28/2018).

[4] Apple. *Apple Developer Documentation: CoreML*. 2018. URL: https://developer.apple.com/documentation/coreml (visited on 03/28/2018).

[5] Apple. *Apple Developer Documentation: HKWorkoutSession*. 2018. URL: https://developer.apple.com/documentation/healthkit/hkworkoutsession (visited on 03/28/2018).

[6] Apple. *Apple Developer Documentation: Integrating a Core ML Model into Your App*. 2018. URL: https://developer.apple.com/documentation/coreml/integrating_a_core_ml_model_into_your_app (visited on 03/28/2018).

[7] Apple. *Apple Watch Series 3: Technical Specifications*. 2018. URL: https://support.apple.com/kb/SP766?locale=en_US (visited on 03/28/2018).

[8] V. R. Basili, G. Caldiera, and H. D. Rombach. "The Goal Question Metric Approach." In: *Encyclopedia of Software Engineering*. Wiley, 1994.

[9] J. Cohen. "A Coefficient of Agreement for Nominal Scales." In: *Educational and Psychological Measurement* 20.1 (1960), pp. 37–46.

[10] S. Mitra and T. Acharya. "Gesture Recognition: A Survey." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3 (May 2007), pp. 311–324.

[11] Motius. *Motius GmbH Website*. 2018. URL: `https://www.motius.de/en/` (visited on 03/28/2018).