Online Accelerometer Gesture Recognition using Dynamic Time Warping and K-Nearest Neighbors Clustering with Flawed Templates

Author:
R.M.W. Kluge
(S4388267)

Supervisors:
dr. F.A. Grootjen
Prof. P.W.M. Desain

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science
in
Artificial Intelligence

June, 2017
Radboud University

Abstract

Faculty of Social Sciences
Artificial Intelligence
Bachelor of Science

Online Accelerometer Gesture Recognition using Dynamic Time Warping and K-Nearest Neighbors Clustering with Flawed Templates

by R.M.W. Kluge

In this thesis we will discuss how we can conduct online accelerometer gesture recognition using Dynamic Time Warping (DTW) and K-Nearest Neighbors (KNN) Clustering. DTW works based on templates. These templates are autonomously extracted from the data streams. However, this algorithm can fail sometimes, resulting in extracted templates not conforming to the template standard. When a template deviates from this standard, it is considered a flawed template. This deviation could either happen by a miscalculated periodicity or wrong starting point. It is investigated if the performance of gesture recognition can be improved when we add flawed gesture templates to the training set. During online classification, it can happen that the template extraction goes faulty and tries to classify a gesture with a flawed template. If these flawed templates are already in the training set, this problem can be tackled and therefore, result in an increased performance. Unfortunately, the difference in mean performance seemed not significant enough ($p = 0.208$) to verify this hypothesis.
Contents

Abstract iii

1 Introduction 1

2 Materials and Methods 3
  2.1 Sensor-data acquisition 3
     2.1.1 Client side 3
        2.1.1.1 Viewer Page 3
        2.1.1.2 Record Page 4
        2.1.1.3 Classify Page 5
     2.1.2 Server side 6
        2.1.2.1 Website 6
        2.1.2.2 Websocket 6
        2.1.2.3 Machine Learner 6
  2.2 Continuous-gesture recognition 7
     2.2.1 Continuous gestures 7
     2.2.2 Preprocessing 8
        2.2.2.1 Combined Normalization 8
        2.2.2.2 Autocorrelation 9
     2.2.3 Periodicity Detection 9
     2.2.4 Classification 11
        2.2.4.1 Dynamic Time Warping 11
        2.2.4.2 K-Nearest Neighbors 14
        2.2.4.3 Implementation 14
        2.2.4.4 Online Classification 15
  2.3 Forming the datasets 15
     2.3.1 Original dataset 15
     2.3.2 Flawed dataset 16
     2.3.3 Types of flaws 17
        2.3.3.1 Multiple Periods 17
        2.3.3.2 Shifted Period 17
        2.3.3.3 Partial Period 18
  2.4 Experiment setup 21

3 Results 23
  3.1 Idealistic results 23
     3.1.1 Idealistic original performance 23
     3.1.2 Idealistic performance with flawed gestures solely in the training set 23
        3.1.2.1 Performance of the multiple-periods flaw dataset 24
        3.1.2.2 Performance of the shifted-period flaw dataset 24
        3.1.2.3 Performance of the partial-period flaw dataset 25
Chapter 1

Introduction

For our group project, we had the opportunity to create an application that transforms a static entertainment show into a more interactive experience for the crowd. In order to be able to work on this together at the same time, we divided the project into three different parts: user input, command recognition and command execution. The sub-project that I will be addressing focuses on user input and command recognition.

The Nintendo Wii is a prime example of a user-input device: the Wii controller uses an accelerometer (a device that is able to measure accelerations to detect and recognize user gestures, and converts them into controlled actions on screen. Studies have already been conducted regarding a Wii remote that was ‘hacked’, and is now being used for custom gesture recognition (Schlömer et al., 2008). Smartphones have the same capabilities as the Wii remote, and thus serve as an ideal alternative to the Wii remote. However, there are many different operating systems, and to get the application working on each device, it needs to be coded differently for each system. To address this time-consuming challenge, I found that HTML5 (HyperText Markup Language version 5) is able to detect the values of both the accelerometer and the gyroscope. Most modern web browsers on mobile devices have HTML5 by default. For our implementation we use HTML5 to capture accelerometer and gyroscope data. This means that we now have a platform-independent solution that runs on every smartphone with a modern browser, without the need to code it for different platforms. Unfortunately, the classification of gestures is still relatively CPU intensive; therefore, it was chosen not to classify on the web server or smartphone itself; instead, the heavy work was offloaded to a computer. Similar work was already done with phone-data collection and sending this data to a computer for processing. This involved accelerometer data and websockets (Remseth, 2015), which I used as inspiration. Thanks to websockets, we can set up an indirect connection between the smartphone’s accelerometer and the machine-learning computer, even when both are not on the same local network. The smartphone thus only functions as data collection device, and not for the processing of this data.

In order to recognize the performed gestures, a classification algorithm is needed. Dynamic Time Warping (DTW) in combination with K-Nearest Neighbors is picked to solve our classification issues. Dynamic Time Warping is a pattern-matching method that uses templates, and is used to find patterns in time-series data (Berndt and Clifford, 1994). DTW does this by using one-periodic templates to calculate similarity between one-another. A more detailed explanation of DTW is provided in chapter 2.2.4.1. DTW is used solely for calculation of similarity between two templates, while K-Nearest Neighbors is used to map and connect all these similarities. A similar study regarding accelerometer-based gesture recognition that uses DTW is uWave (Liu et al., 2009). uWave uses DTW in such a way, that each iteration of a
Chapter 1. Introduction

Figure 1.1: Shows the different steps of the template extraction algorithm.

Gesture will adapt the template database either positively or negatively. The downside of this is that over time, prime representations of a gesture have the possibility to be removed if they are old. uWave’s objective “is not to explore the most effective adaptation methods but to demonstrate that template adaptation can be easily implemented and effective in improving recognition accuracy over multiple days” (Liu et al., 2009).

The research question of this paper does not ask what the most effective adaptation method is, but rather whether different templates affect the algorithm’s performance. Online gesture recognition requires an autonomous template extraction algorithm (see Figure 1.1). Our proposed autonomous template extraction algorithm is, however, not flawless and can cause the extraction of erroneous templates. When these flawed templates are solely tested, and do not occur in the training dataset, they have a higher chance to be misclassified, and therefore reduce the performance of classification remarkably. It is interesting to investigate how these flawed templates affect the algorithm’s performance when they are added to the training set beforehand, and what happens when we do not take the flawed templates into consideration. Of course, we could improve the periodicity extraction algorithm so we do not have to deal with the uncertainty of extracting flawed templates, but this might not always be possible. The main idea of this paper is to investigate whether flawed data templates can improve the performance of gesture recognition by adding them to the training set. The research question is as follows: “Can flawed DTW templates (caused by errors in the period-detection algorithm) improve classification performance by adding them to the training set?”
Chapter 2

Materials and Methods

In order to conduct the experiments for our research question, we need a system that is capable to do the following:

1. Collect sensor data (2.1)
2. Apply continuous-gesture recognition
3. Form the different datasets to test performances

First, the necessary accelerometer data are collected; this sensor-data acquisition happens through a web browser. The data are passed real-time to any connected machine learner for further processing. The machine learner is responsible for the recognition of the continuous gestures. After explaining how the machine learner uses DTW with K-Nearest Neighbors clustering, we can form the different datasets. An original (default) dataset is formed and flaw gestures are actively induced to generate flawed templates. The cause of flawed gestures will be analyzed and the different types of flawed gestures are added to their significant flawed dataset accordingly.

2.1 Sensor-data acquisition

Accelerometer sensor data are collected from an android smartphone through a web browser. This is done using HTML5 and a websocket connection to another server. Our system comprises four parts: a recorder client, an observer/trainer client, a web server and a server websocket. The accelerometer data that are being collected consist of the X, Y and Z axes, which represent the planes of an accelerated movement in a specific direction. The measurements occur at intervals of 0.05 seconds (20 Hz).

2.1.1 Client side

The client side is considered the part of the system that is responsible for interaction with the users. It is able to collect necessary user information and display back classified gestures originating from the machine learner.

2.1.1.1 Viewer Page

The viewer page is solely used by observers who want to view the classification data and the corresponding classification gesture. As illustrated in Figure 2.1, the raw data streams are visualized in a graph, while the gesture classification, including relevant data (tempo of the gestures), is represented in text below the graph. There is no limit to the number of observers that can connect to the websocket that provides
Chapter 2. Materials and Methods

2.1. Materials and Methodology

2.1.1 Data Collection

The data is collected using a web-based application that allows users to perform gestures and record sensor data. The application uses HTML5 for the page and web workers for real-time data processing. The data is sent to a server for analysis and storage.

2.1.1.2 Record Page

The recorder is an HTML5 web page with functionality to collect accelerometer sensor data (see Figure 2.2). It is not even necessary to ask for permissions to access the accelerometer in certain browsers, such as Chrome and Firefox, since it works out of the box. The prerequisite, though, is that the Chrome browser will only send this data if the connection is secure. To meet this security requirement, SSL certificates by Let’s Encrypt (Electronic Frontier Foundation, 2016) are installed, which make it possible to serve connections over HyperText Transfer Protocol Secure HTTPS. While recording this data, it is possible to differentiate between training and test cases by selecting the corresponding train/test toggle. This ‘test’ toggle would only be necessary if a user wants to record a particular movement that should explicitly occur under test conditions.
2.1. Sensor-data acquisition

not be included in the training set (for example, poorly executed movements that are only used to test an outcome in order to understand the classification). In later stages, this train/test toggle is deprecated because of the use of cross-validation. The same kind of toggle also applies to select different gestures for supervised classification. These labels are then sent in combination with the accelerometer data through the websocket, and are saved on the machine-learning computer, so that the different cases can be distinguished from each other.

Initially, out-of-the-box HTML5 functionality for accelerometer-data collection was used to obtain this data; however, it seemed to collect inconsistent data in comparison with other devices. To retrieve consistent input results, the different accelerometers from different devices should be normalized to a specific standard. The GyroNorm plugin (Eker, 2016) is implemented to achieve this. GyroNorm has the functionality to check whether the accelerometer data has already been normalized for gravity, and to apply normalization procedures accordingly per device.

2.1.1.3 Classify Page

The classify page is a separate HTML5 web page with almost the same functionality as the record page. The only difference is that this page does not offer users the option to select which movement they are performing, and whether it should be used for training or testing purposes.

Since classification can take longer than recording the data, after ‘start’ was pressed, the smartphone often switched to sleep mode. This is because most smartphones thought they were idling during the recording of data. When the device switches to sleep mode, the browser is automatically treated as a background process, which stops data transmission. To address this problem, the NoSleep.js plugin was implemented. This module is able to keep the smartphone awake in both Android and
iOS web browsers (Tibbett, 2015) by invisibly displaying a canvas that plays a video that loops forever. The web browser will identify this as a movie currently playing, resulting in the device remaining awake.

2.1.2 Server side

![Diagram of Websocket, Website, and Machine Learner](image)

**Figure 2.3**: Visualizes how data is transmitted through the system

2.1.2.1 Website

For the web app, I used yeoman’s ‘generator for web apps’ (Yeoman, 2017), which offered a quick developer workflow and enabled us to easily switch between developer and production environments. This web app is able to present the HTML5 page that contains the necessary scripts to acquire sensor data, and is used for both the collection of sensor data and the visualization of the classified gestures.

2.1.2.2 Websocket

For the ‘pass-through’ sensor traffic we need to be able to relay data from the smartphone to the machine learner computer and vice versa. All this data are being time-stamped and contains mostly accelerometer data. The machine learner should also be able to broadcast messages containing the predicted gesture. To achieve this, we needed a websocket that was able to manage traffic bi-directionally, and Socket.IO (Socket.IO) seemed to fulfill this requirement. Socket.IO is an open-source socket application that is implemented through a Node.JS module; it is also platform-independent and works with any type of data. Figure 2.3 illustrates how these data are transmitted throughout the system.

2.1.2.3 Machine Learner

The machine learner is the brain of our system. By providing data with a gesture label attached to it, it is able to distinguish between patterns (also called ‘supervised learning’). This part is installed on the computer where the machine learning should occur, and the accelerometer data input is obtained through a connection with the websocket. When a smartphone is transmitting data to this websocket, the data are instantly relayed to the machine-learning computer. The training and testing of the model happens offline (it can be done online, but for future work), however,
classification takes place online. Data retrieval and websocket connection are done using Node.js, while machine-learning is done using python. When the accelerometer-data files are being received through the websocket, the Node.js script saves them locally on the computer. The machine-learner can differentiate between incoming training/testing data and classification data. When classification is triggered, the machine-learner automatically splits up the continuous data stream into chunks of 3 seconds of data per file. For each generated file, the Python script automatically attempts to classify the gesture. When classification is successful, the newly classified gesture is sent back through the websocket as feedback for the observers.

### 2.2 Continuous-gesture recognition

**Figure 2.4:** Clockwise movement with stopping after 1 period (left) versus clockwise movement without stopping after 1 period (right).

#### 2.2.1 Continuous gestures

For this project, continuous gestures are used. A continuous gesture is a gesture that requires no pauses between each performance and can be performed in continuous fashion. With continuous gestures, it is possible to detect periodicity and form templates for DTW. The extraction of periods from the raw datafile is explained later.

For implementation, decided was to use four different continuous gestures for classification: left-right, up-down, rotate-clockwise and a rest class. It is arguable that it is easy to distinguish left-right gestures from up-down gestures, as they contain different types of data on the X, Y and Z axes. However, upon further investigation, this is proven to be false. For example: during recording of up-down movements, not only the dominating Z-axis is active, the Y-axis shows periodic activity too. The anatomy of the human arm offers a useful metaphor for explaining this: the arm is connected to a joint, and it is therefore difficult to perform a natural movement that goes straight up and down while the arm is rotating around the joint. Another arguable point is the rest class. Resting is not a gesture that can be actively performed, and there is no periodicity involved. However, it is useful to have a rest class during online classification.

It should be noted that all the movements were recorded in a continuous fashion, which means that the person who was performing the movement continued the same movement even when one period had already passed. It was observed that
the last piece of data is different when the movement was stopped after one period, when compared to data that was recorded in a continuous fashion. This is evident in Figure 2.4. On the ‘accZ’ streams, between points 20-30 and 25-30, respectively; a peak is missing in the right figure. This can be explained by the fact that the user is slowing down to idle or no movement as the movement period is about to end.

2.2.2 Preprocessing

Each set of data is different, and therefore so are the accelerometer data, especially when using different devices during classification. Since most of the accelerometers are different, it is necessary to modify the sensor values to converge to a unit norm. Two methods that help us get towards this goal are being implemented: combined normalization and autocorrelation.

2.2.2.1 Combined Normalization

Normalization of accelerometer data is necessary to scale large and small performed gestures to a unit norm. The normalization procedure that was used was the Scikit’s (Pedregosa et al., 2011) ‘normalize’ method, with parameters to use L2 (or Euclidean) normalization, which is the square root of the sum of squares of the data (see Formula 2.1), and which scales all the data to a valid unit norm.

\[
\sqrt{x_1^2 + \ldots + x_n^2}
\]  

(2.1)

Because we use three different accelerometer streams (X, Y and Z) from the same measurement device, it is illogical to treat each stream as an individual. For various processes like normalization it is illogical to treat each stream independent of each other. There are large differences observed when treating streams individually versus treating them as a whole. My proposed solution is to combine the normalization and treat it as one stream. We first bundle all three streams together into a large array, then we normalize this large array of combined stream data, and finally split it back into its significant X, Y and Z streams.

When applying individual regular normalization, however, causes irrelevant streams with mostly noise to be amplified, and overactive streams to be eased out. This makes it more difficult for the classifier to distinguish between different gestures.

---

**Figure 2.5**: These figures depict the left-right movement gesture: the graph on the left illustrates individual stream normalization, and the graph on the right illustrates combined normalization.
2.2. Continuous-gesture recognition

When performing the left-right gesture, the Y-axis data stream consists mostly of small movements and noise. Figure 2.5 illustrates that when we normalize each stream individually, the Y-axis ‘accY’ stream is amplified to be as strong as the dominating X-axis ‘accX’ data stream. In the figure on the right, this problem is solved with our proposed combined normalization.

2.2.2.2 Autocorrelation

Autocorrelation (or serial correlation) is used to find periodicity in data that contains large amounts of noise. Autocorrelation is performed by cross-correlating the data with a time-shifted version of itself “in order to detect non-randomness or to find repeating patterns” (Sántha and Hermann, 2014) (see Formula 2.2; Y represents the data points, and h is the time-shift lag). We can only calculate autocorrelation by estimation, because our accelerometer data is discrete and not continuous.

\[
C_h = \frac{1}{(N - h)\sigma^2} \sum_{i=1}^{N-h} (Y_i - \bar{Y})(Y_{i+h} - \bar{Y}) \tag{2.2}
\]

Autocorrelation also helps us to find the dominating frequency of our accelerometer data, and to form consistent, clear templates for the DTW. Figure 2.6 illustrates how the raw data differs from the auto-correlated data, and shows how much clearer this signal is.

![Figure 2.6: 10 seconds of the (normalized) raw versus auto-correlated Gamma data streams. The autocorrelated stream shows a streamlined signal. The code for this plot is a contribution of Regan, M. (Regan, 2014).](image)

But how does autocorrelation deal with a stream that contains only noise, and therefore should not be considered as a signal? For example, when analyzing the up-down movement, we can reason that the X-axis of the accelerometer is not used because there is no hand movement from left to right and vice versa. This X-axis should thus contain only noise. It can be reasoned that noisy streams also result in a noisy autocorrelation, and Figure 2.7 (‘accX’ graph) seems to confirm this reasoning. This figure also demonstrates that autocorrelation is able to magnify the periodicity occurring on the Y-axis (‘accY’), that, upon first glance, is not visible to the human eye.

2.2.3 Periodicity Detection

Before we discuss forming the templates for DTW and K-Nearest Neighbors (KNN) classification, we need to determine the number of periods there are in a data file.
We somehow need to be able to extract one period of these and need to fit it into the DTW algorithm. For continuous gestures, each gesture contains a periodic recurrent signal. It is possible to determine the duration of a period if we do not examine the time domain, but rather the frequency domain by transposing the data into frequency domain using Fourier transformation instead. When the data is transposed to the frequency domain instead of the time domain, it is possible to see which frequency dominates the data.

Because our data has a sample rate of 0.05 seconds between each sample, the data cannot be considered a continuous signal. This is problematic because we cannot use continuous Fourier transform, but are forced to use discrete-time Fourier transform instead. The SciPy library for Python containing the Fast Fourier Transform (FFT) function (Jones; Oliphant, and Peterson, 2014) was used to implement this discrete Fourier transform. After applying it, the frequency with the highest intensity is most likely to be the dominating frequency with the duration of one period of a specific gesture. With peak detection, we can determine which frequency this is. Duarte’s peak detection in Python (Duarte, 2015) was utilized, and found most of the possible peaks. These peaks were then used to determine the length of one period. For example, we attempted to extract one period from an up-down gesture. Figure 2.8 visually confirms that one period was extracted.

It should be noted that it is necessary to capture at least a few time-periods of data in order for the Fourier-transform method to work, since the dominating frequency only stands out when multiple periods are inside a data file. This works best when short or quick gestures are used (when the time between the start and end of a gesture is minimal), since more gestures are able to fit into the same time frame. When gestures are longer, more time must be recorded to detect the dominating frequency, otherwise the dominating frequency could be cloaked by other frequencies that are higher in intensity in the Fourier transform than the correct period frequency. In order to improve the chances that the correct periodicity frequency will stand out and rise above the others, a large training file is needed with more performed gestures per file. Figure 2.9 illustrates how autocorrelation helps to improve the quality of the Fast Fourier Transformed frequency graph.

The next problem is selecting the right data stream. Since we are dealing with three streams (X, Y and Z), we also have three possible predictions for the actual dominating frequency. Only one of these streams should be used to predict the actual dominating frequency, so we need to select one of those three. As a measurement to determine the best data-stream with the highest signal-to-noise ratio, we
2.2 Continuous-gesture recognition

Figure 2.8: One period, starting from the peak, derived by peak detection and the periodicity algorithm. The extracted period is displayed in red.

attempted to differentiate the stream quality by using a test for normal distribution of the Fourier frequency transformed data. Unfortunately, this was not a satisfactory test for stream quality, and any related literature on this is still to be found. As a provisional solution, we chose to use a simpler and more straightforward statistical approach: taking the median of all the predicted peak frequencies.

2.2.4 Classification

The classification of gestures happens using supervised learning. Data are given class labels and are being clustered by K-Nearest Neighbors (KNN). When clustering data, a similarity function is needed to see how a template relates to other templates. Dynamic Time Warping (DTW) is implemented to mathematically define this relation between templates using distances. DTW is computationally expensive, so an approximation algorithm is used to speed up classification.

2.2.4.1 Dynamic Time Warping

Dynamic Time Warping (DTW) is a function to calculate the distance between two temporal streams of data. It uses “a template-matching recognition method based around a dynamic programming algorithm. As such, DTW is a pattern-matching method only, meaning that the templates must be generated externally, either by hand or using a discovery algorithm” (Mitchell, 1995). In our system, this means that it is necessary to automatically extract one period from the data stream and feed it as a DTW template.

As for the similarity function inside DTW, Akl suggests computing the similarity cost between two gestures as follows: $DTW(G_i, G_j) = \sqrt{D_{n,m}^2(x) + D_{n,m}^2(y) + D_{n,m}^2(z)}$
FIGURE 2.9: Fourier transformed signal of a raw data stream (first graph) versus an auto-correlated data stream (second graph).
2.2. Continuous-gesture recognition

(Akl; Feng, and Valaee, 2011). Akl was also concerned about the speed of the training process, since this process is needed to compute the DTW value for each stream independently (Akl; Feng, and Valaee, 2011). Furthermore, DTW is computationally expensive (quadratic in both space and time); therefore, it would take even longer when we consider three streams. A solution for this is to use a decent approximation algorithm for DTW, instead of computing the perfect solution.

A recently developed approximation algorithm named FastDTW (Salvador and Chan, 2007) was therefore implemented. FastDTW is an estimator for the exact DTW (that is quadratically complex in both time and space). It performs this approximation by reducing the complexity in a number of different ways, one of which is by putting global constraints on the warping path, for example, the use of the Sakoe-Chiba Band and the Itakura Parallelogram (Keogh, 2002), which are the most popular global constraints (see Figure 2.10). These constraints restrict the algorithm to find a path in the filled area only, thus minimizing the number of possible paths (and therefore minimizes computation).

FastDTW does not use these illustrated bands; instead, it uses this constraint optimization in another way: to further reduce the number of options, FastDTW reduces the number of cells that the signal must pass through by scaling down in resolution to determine the shortest path. Finally, as the last step, FastDTW uses this path in the lower resolution as a constraint (just like the Sakoe-Chiba Band) to find the final path in a higher dimension. This process is illustrated in Figure 2.11.

When normalizing for DTW, not only are the amplitudes of the data streams taken into consideration, but also the length of each template series, which could be normalized to unit standards. Henniger already experimented with this, and concluded that considering normalizing the length of template series decreases the performance of DTW, unless the time length is considered as a separate feature (Henniger and Muller, 2007).

![Figure 2.10: Sakoe-Chiba band (Sakoe and Chiba, 1978) and the Itakura Parallelogram (Rabiner and Juang, 1993) — figures extracted from (Keogh, 2002).](image)
2.2.4.2 K-Nearest Neighbors

K-Nearest Neighbors is a clustering algorithm that classifies input data according to the distances between other data points. The distance or similarity function that we utilized is the previous FastDTW implementation. When using a distance-based classifier, such as FastDTW, Mitsa suggests the use of a K-nearest neighbors classifier where \( k = 1 \) (giving us 1-nearest neighbors or 1NN), as this is considered “the best” (Mitsa, 2010) (Eruhimov; Martyanov, and Tuv, 2007) and “1NN with DTW is exceptionally hard to beat” (Xi et al., 2006).

2.2.4.3 Implementation

The final stage of implementation involves combining all the described technologies to use within our algorithm. This process begins with recording and saving the gestures, and then constructing templates of them. As a last step, we lay out all the templates and form the clusters. Note that the data underwent preprocessing before the extraction of templates, and not vice versa!

2.2.4.3.1 Determining a period’s time window

The first stage of implementation involves determining the time window that a period should contain. This is determined by using the previously described period-detection algorithm, which will provide us with the dominating frequency of the gesture. Using this dominating frequency, we can determine the size (length) of the time window, which is calculated by dividing the sample rate by the dominating frequency. This gives us the amount of data points that we should use for one period of data. If an incorrect dominating frequency is selected, we obtain a flawed template that contains more (or less) than one period’s worth of data.

2.2.4.3.2 Determining a period’s starting point

We decided to use a peak as a reference point for the start of a period. This reference point is important for obtaining a consistent position of where each period starts. However, a known problem is that period detection is not always correct. Therefore, to obtain a solid peak for most of the times, we increase the already determined time window on both sides by 20%. This adjustment is necessary to compensate for the miscalculated periodicity that could occur, and it means that even if the period is slightly off, the starting point could still be correct. Nevertheless, it is not possible to increase the time window at the start or end of a dataset, as there is simply no data to expand the window to.
For the next iteration of a period, we need to redetermine the time window. The initial window of this next run starts at the index of the ending of the previous period. The possibility that periods could overlap is likely (which could be a good thing), since we will increase the time window again by 20% on each side.

2.2.4.3.3 **Determining the template for a gesture** Now that there is defined a time window and a dominating frequency, we can form the template for our DTW. A template should be exactly one period of data, starting from the correct starting point (the peak), and contains the data of all three streams. Each gesture has its own unique template with its characteristics. It is up to DTW to calculate the similarity or distance between one another.

However, there is one problem: we do not have just one starting point and time window, but three. This is due to the fact that, instead of one, we have three streams to choose both a starting point and time window from. We need to pick the ‘best’ data stream from the X, Y and Z streams to use for periodicity and starting point (peak) detection. As suggested earlier, the algorithm takes the mean of all calculated period frequencies. The data stream that matches this median is considered the ‘best’ data stream, and is also used to determine the starting point.

2.2.4.4 **Online Classification**

The Node.js script automatically splits the incoming classification data stream into files that consist of 4 seconds of data, and our algorithm uses this data to form these one-period templates. For faster classification, we considered smaller classification files containing less seconds of data, however, a shorter timespan resulted in the period-detection algorithm no longer being able to detect the correct dominating frequency. As mentioned earlier, we need at least a couple of gestures per file, and four seconds seems to be enough to obtain them. When the algorithm produces a proposed gesture, the proposed gesture is sent back through the websocket to be displayed to the viewers. For this online classification, there is no human interference nor are there checks for the validity of templates. Therefore, period extraction could go faulty, resulting in the misclassification of gestures. To investigate this, we will analyze how the performance changes if flawed gestures are added to the training dataset (in realistic scenarios).

2.3 **Forming the datasets**

There are two types of datasets being formed: the ‘original’ dataset and the ‘flawed’ dataset, both of which are discussed in this section. It should be noted that all the data are recorded from one participant who is performing gestures using one smartphone (OnePlus 3).

2.3.1 **Original dataset**

The original dataset consists only of data that has clean gesture templates. Because we are unsure about the performance of our periodicity detection algorithm, we must manually verify each template to be sure that we have an initial clean dataset. The manual verification of templates is time consuming, and because of limited time, we only have a limited dataset of 53 templates (13 for left-right, 15 for up-down, 14 for rotate-clockwise and 11 for rest).
Chapter 2. Materials and Methods

We have to be cautious regarding contamination of the original dataset. Contamination between training and test data can occur when multiple extracted periods originating from the same data file are used. To ensure this does not happen, a maximum of only one period of data is extracted from each data file. This will take more time to record; however, it will prevent us from testing data while another slice of data from the same data file is also present in the training set. If we do not take this into consideration, and contamination does occur during cross-validation, then the algorithm will always test this sample correctly. This happens because a very similar template is also present in the current training set.

2.3.2 Flawed dataset

The flawed dataset consists only of data that has flawed gesture templates. Flawed gesture templates are templates that the period-detection and/or starting point algorithm have inaccurately extracted before they are sent for classification. For the ‘generation’ of flawed gesture templates, we record the data separately. This data is solely used to attempt to extract flawed gesture templates, and the correctly extracted files will not end up in the original dataset (for prevention of contamination). Again, we need to manually verify which data templates are flawed, and in which category these flaws belong. To ensure that manual verification is successful, and that we do not make the same mistake as the algorithm does, we record 10 periods of continuous gestures in each training file. When more than 10 periods of data can be extracted, we know that the verification of that file went corrupt and should be reconsidered. If everything goes well, from the 10 available periods we will only look at the first 9 (from the 10 available periods, the Continuous Gestures chapter 2.2.1 explains why). Also in this data we prevent data contamination, as this dataset is also being used for training/test splits. If multiple periods in this data file are labeled as flawed, there is only one template extracted.

To intentionally generate flawed templates, we need to know why the periodicity algorithm fails and outputs a flawed gesture. Flawed gestures can occur for the following reasons:

1. The user changes the speed/tempo of the gesture drastically during classification; this can happen either by speeding up or slowing down during recording.

2. The user pauses during classification, and then resumes. While this delay between gestures is already addressed by performing peak detection to detect the next period, it can occur that the user pauses for the length of a period, resulting in incorrect peak detection, and a flawed template that contains just noise.

3. Period detection misclassifies the periodicity.

4. Peak detection captures a faulty peak from an incorrect (noisy) stream.

The next step is to acquire these flawed gestures. This process involves actively attempting to exploit one of the reasons for the occurrence of the flawed gestures, and requires the speeding up and slowing down of the movement of the gestures as well as taking pauses during recording. The most flawed gestures were acquired during slowing down and speeding up while recording. This resulted in the period detection to miscalculate and autocorrelation to cause the signal to ease out over the rest of the dataset.
These eased out signals, which occurred at the beginning or end of the data file (depending on whether the gesture was sped up or slowed down), are so weak that their low amplitudes have the same value range as a rest gesture (see Figure 2.12). This similarity can cause a legitimate gesture to be classified as a rest class. The rest class is a gesture that is different from the rest of the gestures, as this gesture does not display periodicity or peaks. Therefore, there is no definition whether a rest gesture is considered flawed or not. The flawed datasets thus contain no rest gestures.

![Figure 2.12: Low amplitude left-right gesture that is predicted as a rest class.](image)

2.3.3 Types of flaws

Upon investigation of the period-extraction algorithm, we stumbled upon some flaws of the algorithm. There are a number of different flaws that occur often enough to be used to test for significant differences in performance, and to determine whether these flaws could somehow improve the dataset.

2.3.3.1 Multiple Periods

Multiple period templates are templates that contain more than one period. The extraction of multiple periods occurs when the periodicity algorithm misjudges the dominating frequency. This error, which Figure 2.13 illustrates, occurred for faulty periods to be extracted between 1.2 and 2 period sizes. This error occurred for all gestures except rest.

2.3.3.2 Shifted Period

Shifted period templates are templates that are extracted half a period too early or too late. A shifted period template occurs when peak detection uses the wrong data.
stream, which is a stream that does not represent the dominating stream. The peak
detection rather picks another stream that also shows support in periodicity, but
with less amplitude and that is shifted in time. This occurs only during left-right
and up-down gesture extraction; the reason this does not happen to the rotation
gesture is that each auto-correlated stream in this gesture has a peak at the same
time (Figure 2.14). Figure 2.15 presents a correct left-right template, and Figure 2.16
illustrates how the ‘accX’ stream has shifted by half a period.

2.3.3.3 Partial Period

A partial period is a template that does not exactly represent one period, but only
part of a period. A partial-period template occurs when periodicity detection is off,
and mostly occurs when the user is speeding up or slowing down drastically during
the movement. It did not take place during recordings with pauses. Partial period
errors occur in every type of gesture, excluding the rest class. Figure 2.17 illustrates
a flawed partial period, where less than one period is depicted, but the right starting
point is chosen.
2.3. Forming the datasets

**Figure 2.14:** The rotate-clockwise gesture, and peaks resulting at the same time, independent of stream type.

**Figure 2.15:** What a good left-right template looks like.
Chapter 2. Materials and Methods

Figure 2.16: Shifted period — Left-right gesture template shifted by half a period.

Figure 2.17: Partial up-down template. Represents a template that contains less than one period.
2.4 Experiment setup

Around 30% of the extracted periods by the periodicity algorithm are estimated to be flawed. This is an estimation and is not investigated thoroughly. In order to incorporate this percentage in our dataset to match the population, we also need to have 30% of our total dataset to be flaws. Because we have relatively many samples (50% of all recorded templates) in comparison to the original dataset (see table 2.1), we need to reduce this amount to represent the 30%. For each iteration, we randomly pick 60% of all the flawed gestures in order to shrink the flawed dataset towards the total dataset to 30%. It is important to notice that these flawed gestures are not tossed away, but randomly picked during each iteration. After this selection is done, we can treat the data like the original dataset, and make the same cross-validation splits.

The experiments we are conducting consists of several sub-experiments:

1. Original (idealistic) performance. Here it is tested how the algorithm performs with the original dataset only. This is considered the ideal situation, where the periodicity detection works flawless. The results of this experiment are our baseline for idealistic scenarios.

2. Idealistic performance with flawed gestures solely in the training set. Here we test how particular types of flawed gestures influence the idealistic original performance. This scenario happens when flawed gestures are added to the training set, but the periodicity detection works flawlessly and only extracting perfect gestures in the test set.

3. Original realistic performance. Here, the flawed gestures are solely added to the test set. This is considered the realistic situation, as the periodicity detection is faulty sometimes. No flawed gestures are added to the training set yet. The results of this experiment are our baseline results for realistic scenarios.

4. Realistic performance with flawed gestures in the training and test set. In this case it is tested whether flawed gestures in the training set can improve the realistic performance of the test set as well.

For each experiment conducted, cross-validation is applied. With cross-validation, the predetermined datasets are split randomly into training and test sets using a given ratio. This enables us to perform tests with different training and test splits using the same data to give more reliable results. We then compare the means of the calculated performances using statistical tests.

For experiments 3 and 4, separate cross-validation happens to the flawed gesture dataset. In experiment 3, only the test split is appended to the test set, and in experiment 4 both the training and test splits are added to the dataset.
<table>
<thead>
<tr>
<th>Datatype</th>
<th>Good (amount)</th>
<th>Flawed (amount)</th>
<th>Percentage Flawed / Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-right</td>
<td>13</td>
<td>10</td>
<td>43%</td>
</tr>
<tr>
<td>Up-down</td>
<td>15</td>
<td>9</td>
<td>37.5%</td>
</tr>
<tr>
<td>Rotate</td>
<td>14</td>
<td>7</td>
<td>33.3%</td>
</tr>
<tr>
<td>Rest</td>
<td>11</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>53</strong></td>
<td><strong>27</strong></td>
<td><strong>32.9%</strong></td>
</tr>
</tbody>
</table>

**Table 2.1:** Illustrates how many good and flawed gestures are used for our experiments.
Chapter 3

Results

To test the performance of the dataset, 400 cross-validations with a test-set size of 30% were done. This cross-validation creates a random training/test set from the original dataset. Furthermore, pseudo-randomness, which causes the exact same ‘random’ sequence of training/test splits in each run, is used to ensure that each algorithm is tested against the same splits. Through pseudo-randomness, we can compare the performance of different datasets with the same training/test splits using paired statistical tests.

To demonstrate that different training-gesture sets are statistically different, Demšar proposes that non-parametric tests, such as Wilcoxon, are “safer than parametric tests, since they do not assume normal distributions or homogeneity of variance. As such, they can be applied to classification accuracies, error ratios or any other measure for evaluation of classifiers” (Demšar, 2006). The samples can be considered as paired because each performance sample is calculated against the same training/test split. This pairing allows us to use the Wilcoxon signed-rank test (Woolson, 2008). All statistic tests are done using R and their toolkits (R Core Team, 2013). The null hypothesis is rejected when the p-value < 0.05.

3.1 Idealistic results

3.1.1 Idealistic original performance

If our periodicity detection is perfectly accurate (idealistic scenario), we will not have to deal with flawed gestures, and can just use the original dataset. The original dataset consists of 53 templates originating from 53 different recording files (1 template of each file), and because 30% of the templates was used for testing, in each cross-validation iteration, the training set and test set consists of 37 templates and 16 templates, respectively. This original dataset without flawed gestures involved demonstrates a mean performance of 0.788, and a standard deviation of 0.108. Figure 3.1 depicts the performance of each predicted gesture. In this figure, 100% of the left-right gestures are classified correctly, whereas only 69% of the rotate-clockwise gestures are classified correctly.

3.1.2 Idealistic performance with flawed gestures solely in the training set

The next experiments will show how performance changes when flawed gestures are added to the training set. This will show the performance in situations where the periodicity detection works perfectly and thus no flawed gestures are classified or tested (idealistic scenario). It is tested how different types of flawed gestures in the training set affect the performance on the original test set.
Chapter 3. Results

Figure 3.1: Confusion matrix of the original dataset’s performance.

After the training/test splits are generated from the original dataset, a category of flawed gestures is appended to the training set. Ensured is that these flawed gestures could never occur in the test set. The Wilcoxon signed-rank test with continuity correction is used to determine whether there is a significant difference between the original and the different original-plus-flawed dataset (two-sided test). For each ranked Wilcoxon test that is performed for each flawed gesture dataset, we assume the following hypothesis:

\[ H_0 : \mu_{\text{original}} = \mu_{\text{flawed}} \]

\[ H_a : \mu_{\text{original}} \neq \mu_{\text{flawed}} \]

3.1.2.1 Performance of the multiple-periods flaw dataset

The multiple-periods flaw dataset demonstrated a performance of 0.773, and a standard deviation of 0.102. It was observed (see Figure 3.2) that the classification of the rest class worsens in comparison with the original dataset (61% and 52% respectively). The p-value is 0.000307, which rejects the null hypothesis ($p < 0.05$). This means that the original dataset demonstrated a significant difference in performance in comparison to the dataset that contains the multiple-periods flaw.

3.1.2.2 Performance of the shifted-period flaw dataset

The shifted-period dataset reached a performance of 0.744, and a standard deviation of 0.109. The additional flawed data consisted of only left-right and up-down data; however, this data seemed to affect the classification of the other gestures as well. The confusion matrix (Figure 3.3) illustrates that gesture classification decreased for both the up-down gesture and the rest class. The up-down gesture was wrongly predicted to be a rotate-clockwise gesture more often (11% in the original dataset...
3.1. Idealistic results

25

Figure 3.2: Confusion matrix of the multiple-periods flaw dataset performance.

versus 21% in the shifted-period flaw dataset), and the rest class was wrongly predicted to be a left-right gesture more often (29% in the original dataset versus 44% in the shifted-period flaw dataset). It is evident that this shifted-period dataset influences the classifier’s performance. This significance is confirmed by the Wilcoxon test, which results in a p-value of practically 0 ($2.2e^{-16}$). The null hypothesis is rejected; the mean performance of the shifted-period dataset does not equal the mean performance of the original dataset.

3.1.2.3 Performance of the partial-period flaw dataset

The partial-period dataset results in a performance of 0.789, and a standard deviation of 0.101. The classification performance of the rotate-clockwise gesture improved (69% in the original dataset compared to 74% in the partial-period flaw dataset). However, the performance of the rest class decreased from 65% to 56% (see Figure 3.4).

The overall performance of the partial-period flaw dataset was slightly better than that of the original dataset. The Wilcoxon test revealed a p-value of 0.813, which retains the null hypothesis. This means that the partial-period dataset does not differentiate significantly from the original dataset.

3.1.2.4 Performance of the partial-periods and multiple-periods flaw dataset

Tests were also conducted to determine whether multiple combined flaws change the overall performance in relation to the original dataset. Partial-period flaws and multiple-period flaws were added to the training set and run for 400 iterations. The mean performance is 0.765, with a standard deviation of 0.105. The mean performance of both these flaws is worse, compared to the flaws used separately. The
Chapter 3. Results

Figure 3.3: Confusion matrix of the performance of the shifted-period flaw dataset.

Wilcoxon test also revealed to reject the null hypothesis ($p < 0.05$), with $p \approx 0$ ($p = 5.402e^{-7}$). Figure 3.5 illustrates the confusion matrix for this experiment.

Figure 3.4: Confusion matrix of the partial-period flaw dataset.
3.2 Realistic results

Because our periodicity detection is not perfect, we should assume that mistakes will be made during template extraction in the online classification phase. To test how our algorithm performs when classifying flawed gesture templates, flawed templates are added to the test set by default.

As described earlier, 60% of all flawed gestures are picked at random to represent the 30% of flawed gestures in the population. Cross-validation occurs with 70% training and 30% test data for splits, respectively. In one condition, the flawed templates are solely added to the test set. This is considered the baseline condition, and are compared with our next condition, the experimental condition.

In the experimental condition, flawed templates are added to both the training and the test set. We hypothesized that the realistic performance with flawed gesture templates in the training and test set has a higher mean performance than the realistic original performance with only flawed gesture templates in the test set. Statistical hypothesis that is retained:
\[
H_0 : \mu_{\text{original}} \geq \mu_{\text{flawed}} \\
H_a : \mu_{\text{original}} < \mu_{\text{flawed}}
\]

3.2.1 Realistic original performance (flawed gestures in the test set)

As for the realistic original performance, cross-validation of flawed templates utilizes 70% and 30% for the training and test set respectively, but in this experiment, the training split is discarded. Only the test split is being added to the current test set.
Chapter 3. Results

This baseline experiment has a mean performance of 0.659, with a standard deviation of 0.103. In comparison with the idealistic original performance (0.788), the overall performance decreases by 12.9%. This decrease in performance can be explained by the fact that there are templates being tested that the classifier has never seen before. When comparing Figure 3.6 of this experiment with Figure 3.1 of the idealistic original performance, it is observed that the performance of every gesture decreases, with the exception of the rest class (the rest class performs about the same). This equal performance of the rest class can be explained by the fact that there are no rest gestures in the flawed dataset, because this rest class has no flawed gestures (as mentioned earlier in 2.3.2). Hence, the test splits of the flawed gesture dataset do not contain additional tests for the rest class, and the tests for the rest class during the realistic original performance are exactly the same as during the idealistic original performance.

![Confusion matrix](image)

**Figure 3.6**: Confusion matrix of the realistic original performance.

3.2.2 Realistic performance with flawed gestures in the training and test set

This experiment answers the research question of this thesis. In comparison to the previous baseline experiment, the flawed training split is now added to the dataset. The same pseudo random splits are used for the experiments to guarantee the equal results when we re-run. This experiment has a mean performance of 0.667, with a standard deviation of 0.081. In comparison to the baseline experiment ($\mu = 0.659$), this is a slight increase in performance. When looking at the individual performances of gestures, weirdly enough only the up-down class seems to observe an increased performance. The left-right class and rest class both observed a decrease in performance, while the rotate-clockwise class remained roughly the same.
A surprising observation is the classification of the rest class. As illustrated in Figure 3.7 and Figure 3.6 of our baseline experiment, the rest class performs 12% worse in the experimental condition (55% performance) versus the baseline condition (67% performance). Beforehand, the rest class is never (0%) falsely predicted to be a rotate-clockwise gesture. However, in this experiment this misclassification occurs 17% of the time. This misclassification could be explained by the fact that some flawed rotate-clockwise gestures have such a weak amplitude, that they resemble the rest class.

The same goes for classification of the left-right class; observed is a decrease in performance by 16% (79% resp. 63%). The up-down class, however, shows an increase in performance by 13% (73% resp. 86%). The performance of the left-right gesture decreases by 16% (79% resp. 63%). This has to do with the fact that the left-right gesture is suddenly more often predicted to be a up-down gesture.

The only gesture that showed increased performance was the up-down gesture, which increased by 13% (73% resp. 86%).

![Confusion matrix. 400 iterations, test ratio: 0.3](image)

**Figure 3.7**: Confusion matrix of the realistic experimental performance.

### 3.2.2.1 Significance

The Wilcoxon test shows a P value of 0.208. This P-value does not satisfy $p < 0.05$, so we have to retain the null hypothesis. This statistical result concludes that the baseline’s mean performance is higher than or equal to our experimental’s mean performance. This statistical test matches our observations in Figure 3.8. This figure shows no significant shift in means, but it does show that the experimental condition (‘Flawed’) is more normally distributed than the baseline (‘Original’) experiment.
FIGURE 3.8: Histogram of the performances of the original dataset and the histogram of the performances of the experimental ‘flawed’ experiments.
Chapter 4

Conclusion and Discussion

4.1 Idealistic scenario

It was hypothesized that the performance with flawed gestures in the training set does not change, because only good templates are tested, and should not interfere with the flawed templates in the training set. However, this hypothesis is found incorrect. The performance of the idealistic scenario is negatively affected by the flawed gestures. In idealistic scenarios, where the template extraction is perfect, the achieved performance is 0.788. Of all the flawed gesture types that were added to the training set, only the flawed gesture types multiple-periods ($\mu = 0.773$) and shifted-period ($\mu = 0.744$) significantly changed the performance of the proposed DTW and KNN algorithm ($p = 0.000307$ and $p = 2.2e^{-16}$ respectively). The partial-period flaw seems not to change the performance of the algorithm ($\mu 0.789$, with $p = 0.813$).

4.2 Realistic scenario

It was hypothesized that the realistic original performance will decrease in comparison to the idealistic original performance, because flawed odd templates are being tested that do not occur in the training set. The baseline experiment results show a decrease in performance by 12.9%, which confirms this hypothesis.

It was also hypothesized that the performance of the baseline condition drastically decreases in comparison to the idealistic scenario, but that this performance would be restored in the experimental condition, where we also add flawed gestures to the training set (our research question). The results show that the addition of flawed gesture templates to the training set increase the mean performance ($\mu = 0.659$ versus $\mu = 0.667$), but that this difference is not significant ($p = 0.208$). The null hypothesis is thus false; when template extraction has flaws, adding flawed gesture templates to the training set does not significantly increase classification performance.

4.3 Discussion

During experiments, new data had to be recorded because the dataset was contaminated. This contamination happened because multiple templates were extracted from the same file. It happened that some of these templates were found in both the training and test set simultaneously. This caused us to have a smaller dataset than we would have liked to test with.

Not only the dataset size is an issue, also the ratio flawed/good gestures in the population should be considered. The flawed gestures should represent the same percentage of occurrence in our experiments as in the population. It has not been
tested thoroughly what the occurrence rate of specific flaws are, and what the success rate of the template extraction algorithm is. Table 2.1 shows the distribution of the type of (flawed) gestures in our dataset. These occurrence rates should be determined and reflected back in the datasets.

It has not been tested thoroughly what the exact error rate of the template extraction algorithm is. In this thesis we assume an error rate of 30%. If this error rate is discovered to be lower, it is to be questioned if it is even needed to test our research question, as the error rate is very low.

Another point for discussion is that the algorithm is only trained on one participant and device. It is not tested whether the same results will be acquired when multiple participants are involved. Therefore, it is not legitimate to generalize the current results to the population.
Chapter 5

Future Work

For future work, as previously noticed, these experiments should be re-run with more data, more participants and more devices. Another expansion will be to implement and test with more types of continuous gestures.

Improvements need to be made when selecting the right datastream for determination of periodicity and determining peaks. This part is far from correct which causes a lot of flawed gestures. We currently assume that the datastream which displays the median of all the calculated periodicity frequencies is the datastream with the highest signal to noise ratio.

Another improvement to the algorithm could be by checking the number of periods that exists in a to be classified file. For example, when there are six periods, then try to classify all six of these periods. If four of six periods are classified as up-down, we are more certain the true gesture is the up-down gesture.


Socket.IO. URL: https://socket.io/.


