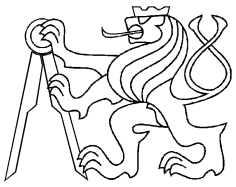


Czech Technical University in Prague
Faculty of electrical engineering
Department of cybernetics

Counting repetitions of exercises from body worn sensors

Diploma thesis

DRAFT VERSION

Michal Šustr
michal.sustr@gmail.com

Prague, 2015

1) Prostudujte existující zařízení a principy využívané při automatickém sledování pohybové aktivity při fyzickém tréninku s použitím váhy vlastního těla nebo činkami jako jsou dřepy, kliky, mrtvý tah, bicepsový zdvih a podobně.

2) S dodaným experimentálním HW (pohybové senzory) vykonejte na několika lidech měření průběhu zvolených cviků pro získání surových dat k dalšímu zpracování. Data rovněž poskytněte k obecnému použití.

3) S využitím metod jako Hidden Markov Models, Dynamic Time Warping, Support Vector Machines nebo jinými vhodnými metodami navrhněte postup pro robustné počítání opakování vybraných cviků.

4) Zhodnoťte nejvhodnější umístění těchto senzorů na těle člověka (ruce, nohy, pas, ...), dále nutný počet senzorů, jejich vzorkovací frekvenci a zejména úspěšnost počítání opakování zvolených cviků.

Abstrakt

Táto práca popisuje algoritmus pre počítanie počtu opakovaní cvičení pri použití senzorov merajúcich zrýchlenie a rotačnú rýchlosť umiestnených na tele človeka. Rozpoznáva sa pohyb pri cvičení s vlastnou váhou tela. Senzory sú umiestnené na rôznych častiach tela pre zistenie najvhodnejšej polohy pri ktorej algoritmy majú najvyššiu úspešnosť. Pri vývoji metódy je kladený dôraz na nízku časovú a pamäťovú náročnosť, aby algoritmy boli vhodné pre implementáciu do jednočipových procesorů. Algoritmy sú testované na nameranom dátovom sete pozostávajúcom z cvičení s rôznymi osobami.

Abstract

This thesis describes algorithm for counting number of repetition of exercises using body-worn sensors measuring acceleration and rotational speed. The movement is recognized for body-weight exercises. The sensors are placed at different parts of the body to find the most appropriate locations where the algorithms have the highest accuracy. While developing this method low time and memory constraint is considered, so that the algorithms are appropriate for implementation in embedded devices. Algorithms are tested on acquired dataset of workout with various people.

Acknowledgement

I would like to express my gratitude to my thesis advisor Petr Novák for his heart-warming approach and long discussions during lunch, Jan Sedivy for his project support in eClub, Marek Novak and Martin Borysek for help with the hardware and to my family and friends for their support throughout my studies.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 25. 4. 2016

Contents

Chapter 1

Introduction

1.1 Current status

During last few years there has been a big emergence of various fitness trackers / wearables with the purpose of tracking daily motion activity, fitness or life functions of their users (see Figure 1.1). They commonly use accelerometers, gyroscopes and heart-rate monitors to measure data. The number of producers is increasing, mainly because of the decrease of cost of electronic parts required for manufacture. Sensors are sometimes 100x cheaper than in the last decade because of the wide-spread of smartphones.

There are many companies on the market, such as Samsung, Apple, Fitbit, Mio, Sony, Garmin, TomTom, Nike, Asus and others. The available devices are unfortunately closed. Ordinary developer is not able to implement their own software and algorithms in these commercial devices, or just have the ability to gain raw data suitable for next processing, experiments or research. This fact makes the existing devices usable only for commercial purposes and not for scholar or experimental purposes.

Not only professional sportsmen, but also ordinary people are getting more used to using digital sport trainers while performing sport activities. It can be various special sport watches or bands or applications on smartphones, where they can set and track their progress towards desired goals like weight loss, muscle gain, competition preparation, etc.

However, none of these devices provide the benefit of tracking specific exercises while moving in a gym or a workout park. This would be useful for exact evaluation of workout progress of each individual as they try to follow their long-term workout plans to reach their desired goals.

This is the reason why I decided to create this thesis: to automatically recognize the number of repetitions of exercises from body worn sensors. There is some work done in this area (as in [32, 24, 10]) which served as inspiration for this thesis. My ultimate goal

is to be able to evaluate complicated motion directly on embedded devices (commonly called wearables). By complicated motion I mean exercises that are combined together in creative ways. I would like to be able to use it in other sports as well where the emphasis is in performing the motion in a beautiful smooth way, such as in parkour, freerunning or gymnastics. I would like to evaluate correctness, smoothness and the overall beauty of performance to provide very insightful advice for the user about how he should improve his technique. This advice would be difficult to acquire otherwise, because alternative recording on cameras is often obscured by various body parts. This thesis is a stepping stone towards this goal as I hopefully plan on further developing these algorithms.

1.2 Goals of this thesis

Main goals of this thesis are:

- Study the existing devices and priciples for automatic recognition of movement activity for physical training.
- Collect raw data of selected exercises of various people with provided measurement hardware from multiple positions on the body. Align all of the data into a single stream that has uniform sample frequency. Provide the collected data.
- Use appropriate methods to robustly count the number of repetitions of selected exercises, while keeping them computationally cheap enough for the use in embedded devices.
- Evaluate the best placement of these sensors on the body, their required number and the error rate of counting.
- Create simple implementation for embedded device.

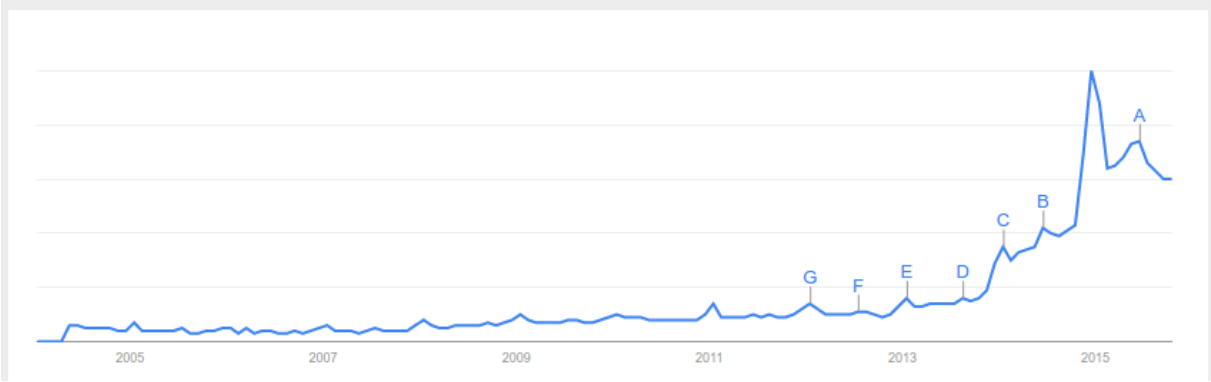


Figure 1.1: Search of words related to fitness tracking by Google Trends.
Original at http://jdem.cz/bt4kj3

This thesis is however not concerned about:

- The optimal construction of the measurement hardware and transmission of data (compression, encryption, safety etc.).
- Detailed documentation of the hardware.
- The automatic recognition of exercises without prior information. The counting of repetitions is in only one type of movement. The goal is not to create universal recognition of various kinds of movement, but counting of repetitions of one single type of exercise and differentiating it from other free motion.

The brief overview of history of the project is:

- I found the simplest way how to get acceleration data from *one* sensor. I had no knowledge or experience in this area, so I thought it would be a good start. For this I used breadboard that contained only the sensor, processor and bluetooth and I collected data via bluetooth to a phone. This helped to answer questions like if it is even possible to do this kind of recognition and if low frequency (50Hz) is good enough.
- Since the data looked reasonable, we created a *pair* of devices with a friend Martin Borýsek based on Arduino Dues that can log the data on SD card or transfer it to cell phone via bluetooth (see 3.2). The underlying assumption was that one device is not enough to recognize the movements complexly, but two devices placed on arm and leg might be enough for the task.
- It isn't immediately apparent what kind of information is needed for good recognition: do I need acceleration {and,or,xor} gyroscopic data? What is the best position on the body? Through eClub summer school I got hold of new special hardware that had *8 distinct sensors* that could be placed on the body.
- However there were some big problems with collecting data because of packet loss and I wasted a lot of time unfortunately. I had to repeat it again with improved system described in 7.2.
- I coded the algorithms and evaluation framework for the movement recognition and started to play with different approaches to the problem, using different settings etc.
- Once I have found good classifier I created implementation for embedded hardware.

1.3 Structure of this thesis

This thesis is structured in following way:

- Chapter 2 shows the most important knowledge of the domain to solve the task, how the physical reality of the problem affects the input data and where the data comes from.
- Chapter 3 introduces hardware that was used for capture of the data and implementation in embedded device.
- Chapter 4 presents theoretical approach for signal processing algorithms and their advantages and disadvantages.
- Chapter 5 outlines the design of the recognition system.
- Chapter 6 shows the most important information about the implementation that is available in the attached CD.
- Chapter 7 presents the results of evaluation on available dataset.
- Chapter 8 concludes the works of this thesis.

Chapter 2

Task analysis

In this chapter I present basic task analysis, how the physical reality of the problem affects the input data, where the data comes from and the domain knowledge of exercise that can be used in recognition model design.

2.1 Available data

The best information that one could have for evaluating person's motion is by using a grid of points that mark multiple (x,y,z) positions in space $\mathbf{x}$ of his body parts at each time step t. In this case we could create a model for physical activity using time series $\mathbf{x}(t)$ and probabilistically determine that the person is most likely to be sitting, standing, laying down, walking, etc., which is reffered to commonly as *activity recognition*. The methods that can be used to capture point locations include using commercially available motion capture systems (like in Figure 2.1), using RGBD camera[1] or ensemble or RGBD cameras (such as in Microsoft's Kinect). There are some recent works in pose estimation from visual inputs using deep learning [6].

However, acceleration can be measured much more easily than exact position in space, it doesn't require difficult processing of the camera data and there are very cheap sensors available. Acceleration is the second derivative of position. One could think of using integration to get back the positions in space, but that cannot be done well because of integration drift. Accelerometers coupled with gyroscope (which measures rotational speed) called commonly IMU are good sources of acceleration data and can be used for recognition successfully as well.

Finding location in space is important problem for flying UAVs[2] which also use IMUs. Most commonly they use GPS for localization, because it gives a relatively precise location

[1] RGBD - RGB camera which perceives scene depth (D)
[2] UAV - unmanned aerial vehicle

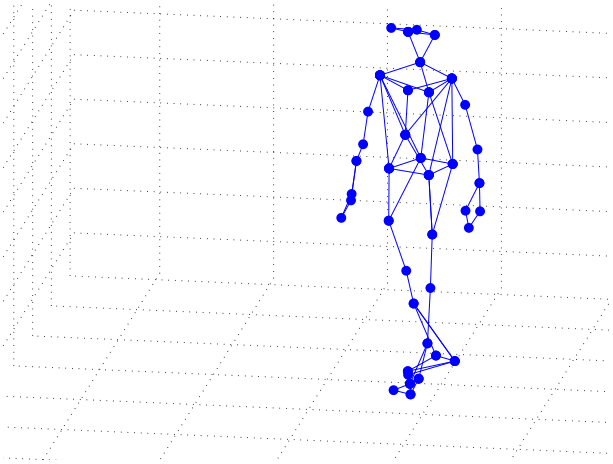


Figure 2.1: Grid of points that mark positions in space of various body parts captured with motion capture.

for a low cost and combine this data with IMU to determine angular deviation (yaw, pitch, roll) of the vehicle. There have been works in localizing using on board cameras [21]. However, none of these are applicable for my task for obvious reasons.

One could think of trying to solve the integration drift by using magnetometer data with some clever technique to combine it with acceleration and angular speed, like using Kalman filters. The magnetometer points at Earth's magnetic north and therefore it reduces the uncertainty of location in one axis. However this suffers from a problem - in a gym there is usually a lot of iron, the person might lift some dumbbells which would completely confuse the sensors.

Another idea might consist in trying to have some magnet on the body, or maybe externally. The magnet would provide a fixed orientation point and again with some clever filtering we could get some nice localization. But quick investigation shows this is not physically feasible. Look at Figure 2.2. Let's suppose that we want that under any condition the deviation α of the direction of magnetic vector from the magnet is less than 3° compared to Earth's magnetic field. The strength of magnetic field of Earth is somewhere between $\langle 25; 65 \rangle$ nano tesla[3]. Then the magnetic force at distance of $L = 1\text{m}$ must be $B_x = \frac{B_e}{\tan \alpha} = \frac{25nT}{\tan 3°} = 16\text{mT}$. Usual permanent magnets have "intrinsic induction" of about $B_i = 0.4T$. Let's use a small, practical permanent magnet of size $2 \times 3 \times 0.5$ cm. By modelling permanent magnet as a magnetic dipole moment[4] we can predict at what

[3] http://www.crystalinks.com/earthsmagneticfield.html

[4] Source of information and equations: https://tiggerntatie.github.io/emagnet/offaxis/mmoffaxis.htm

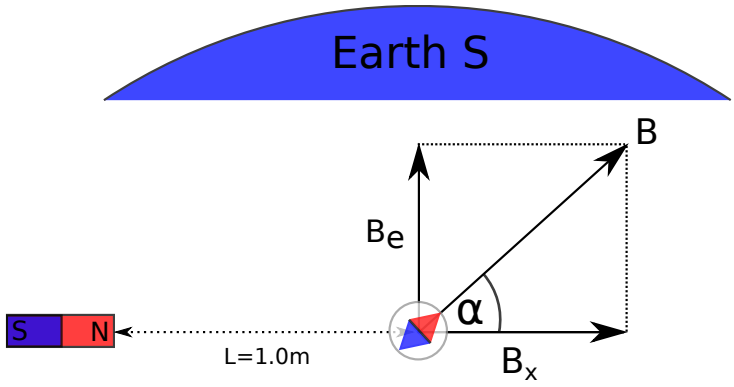


Figure 2.2:

distance we can actually generate the influence of 16mT:

$$B_x = \frac{\mu_0 M}{4\pi}\left[\frac{3\cos^2\alpha - 1}{L^3}\right] \tag{2.1}$$

$$M_{(p)} = \frac{B_i V}{\mu_0} \qquad \text{For } \textit{permanent} \text{ magnets, magnetized in a single direction} \tag{2.2}$$

$$M_{(e)} = NiA \qquad \text{For air core } \textit{electromagnets} \text{ (solenoid)} \tag{2.3}$$

$$B_x = \frac{B_i V}{4\pi}\left[\frac{3\cos^2\alpha - 1}{L^3}\right] \tag{2.4}$$

$$L = \left(\frac{B_i V(3\cos^2\alpha - 1)}{4\pi B_x}\right)^{1/3} = \left(\frac{0.4\mathrm{T}\cdot 3\cdot 10^{-6}\mathrm{m}^3(3\cos^2(3^\circ) - 1)}{4\pi 16\cdot 10^{-3}\mathrm{T}}\right)^{1/3} = 2.3 \text{ cm} \tag{2.5}$$

That is only 2.3 cm with ordinary magnet! The Earth's magnetic field is actually about 100x stronger at distance of just $L = 1\mathrm{m}$. Unfortunately, the field's influence decreases cubicly with distance, so this approach cannot be used.

Let's suppose we would use external magnets (although such system wouldn't be easy to use for a normal user). If we are in a room and we would want to satisfy the angular condition up to distance of 10 meters, then the magnetic dipole moment (using 2.1) must be:

$$M = \frac{4\pi B_x L^3}{\mu_0(3\cos^2\alpha - 1)} = \frac{4\pi\cdot 16\cdot 10^{-3}\mathrm{T} * (10\mathrm{m})^3}{1.256\cdot 10^{-6}\mathrm{Tm/A}(3\cos^2(3^\circ) - 1)} = 8.037\cdot 10^7\mathrm{Am}^2 \tag{2.6}$$

What could be a power consumption of such a magnet? If we take a coil with $N = 10^4$ turns and $A = 1\mathrm{dm}^2 = 10^{-2}\mathrm{m}^2$ area, then the current going through this coil has to be about

$$I = \frac{M_e}{NA} = \frac{8\cdot 10^7\mathrm{Am}^2}{10^4\cdot 10^{-2}\mathrm{m}^2} = 8\cdot 10^5 A \tag{2.7}$$

which is not realizable at all, not even for a shorter distance (L=3m is about just about one order less of current size) - there's no point to even consider power consumption - it would be huge!

There are many works that use only data from IMUs to recognize person's activity. Some are using sensors embedded in smartphones [24, 3, 5, 29, 10] or they use specialized hardware to do so [4].

One example of activity recognition could be detecting person's fall, which is interesting for example in applications for elderly people assistance (calling ambulance in the case of fainting [34]).

The IMU are good enough for the application of counting the number of repetition of exercises, since they are cheap and provide enough distinct data [24, 4]. It is actually not necessary to have full (x,y,z) positions in space.

Person's activity influences the heart rate (HR) based on the difficulty of activity. One can think of using this data to help with recognition. However, this idea of using another input has been discarded. Heart rate has low correlation with the actual start and end of movement[30] - once the person starts to exercise his HR goes up, but it has a long inertia until it comes back to rest and therefore it is not discriminative.

2.2 Measurement sensors

A microelectromechanical system (MEMS) is an embedded system that integrates electronic and mechanical components at a very small scale to create a sensor.

Accelerometers measure force that is exerted on a mechanical part of the system, and calculate acceleration using simply $a = m/F$. In some accelerometers, piezoelectric crystals such as quartz take advantage of piezoelectric effect. A crystal is attached to a mass and when the accelerometer moves, the mass squeezes the crystal and generates a tiny electric voltage which is measured, as illustrated in Figure 2.3. Another approach uses capacitor plates that move and measures the difference in capacity. These electromechanical parts can be arranged orthogonally to measure (x,y,z) components of acceleration. Since accelerometers measure overall acceleration, in stillness they measure the gravitational acceleration.

Gyroscopes measure angular velocity. MEMS gyroscopes use the Coriolis Effect to measure the angular rate, as shown in Figure 2.4(a).

When a mass m is moving in direction v and angular rotation velocity Ω is applied, then the mass will experience a force in the direction of the arrow as a result of the Coriolis force. And the resulting physical displacement caused by the Coriolis force is then read from a capacitive sensing structure.

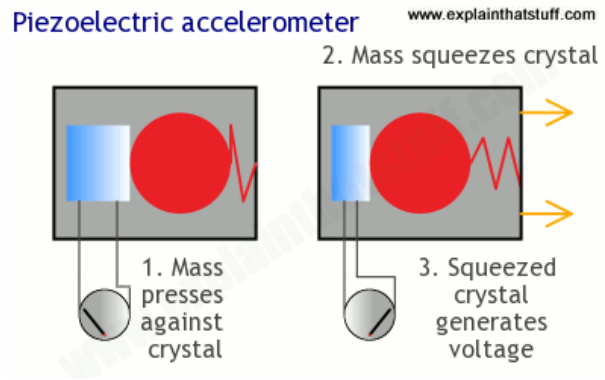
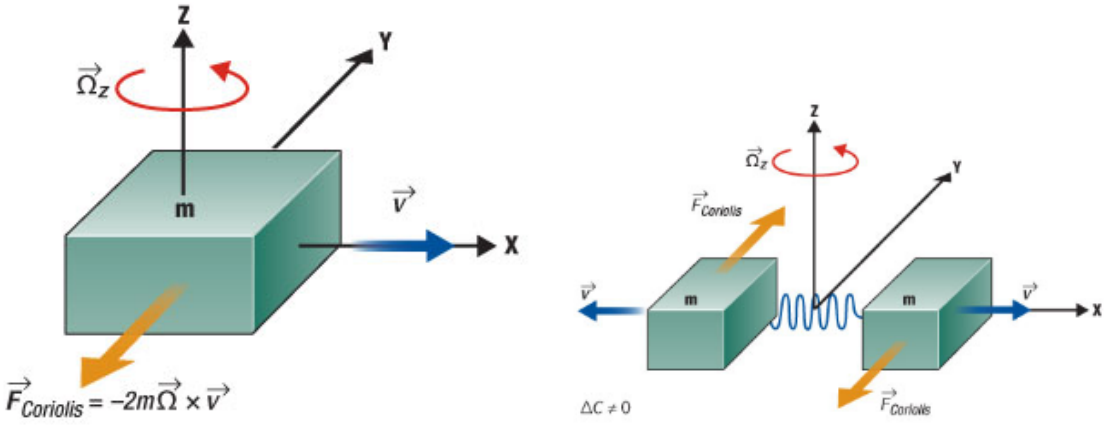


Figure 2.3: Peizoelectric accelerometer, image courtesy
http://www.explainthatstuff.com/accelerometers.html



(a) Coriolis effect (b) Angular velocity applied

Figure 2.4: Image and text courtesy
http://electroiq.com/blog/2010/11/introduction-to-mems-gyroscopes/

Most available MEMS gyroscopes use a tuning fork configuration. Two masses oscillate and move constantly in opposite directions (Figure 2.4(b)). When angular velocity is applied, the Coriolis force on each mass also acts in opposite directions, which result in capacitance change. This differential value in capacitance is proportional to the angular velocity Ω and is then converted into output voltage for analog gyroscopes or LSBs for digital gyroscopes.

When linear acceleration is applied to two masses, they move in the same direction. Therefore, there will be no capacitance difference detected. The gyroscope will output zero-rate level of voltage or LSBs, which shows that the MEMS gyroscopes are not sensitive to linear acceleration such as tilt, shock, or vibration.

Magnetometers are a common measurement sensor that comes with accelerometers and gyroscopes, it is a part of the used measurement sensor. They are used mostly to find orientation compared to Earth's magnetic poles. They are sensitive to presence of metal objects. Since there's usually a lot of metal objects in gyms, they would not give useful

values and their usage has been discarded.

2.3 Domain knowledge of exercise

Repetition of exercise is the motion that corresponds to the person's movement starting at the inital position, passing through all the motion that is required for the given exercise, coming back to the initial position.

Across all the collected data there is one phenomena that can be observed: the repetitions that are in the beginng or end of exercise set have a different shape than the repetitions in the middle. This can be explained that the person has to gain some movement inertia from the starting state of stillness. An example is illustrated in Figure 2.5. The user often does some special movement before the exercise starts, such as laying down on the ground before doing pushups, or jumping upwards to get hold of the pullup bar before doing pull-ups. These facts can be used to help recognize when the exercise set starts or stops, but also makes segmenting the first and last repetitions more difficult.

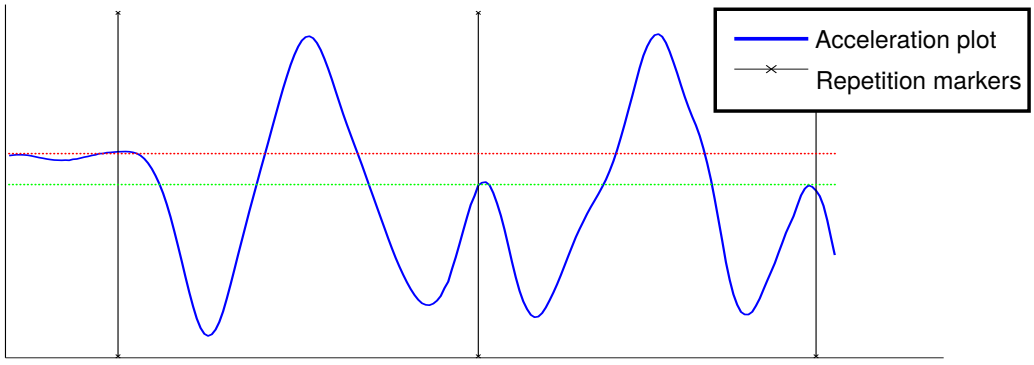


Figure 2.5: Very simplified illustration of acceleration inertia. The acceleration is different in the beginning repetition from the one following right after.

The movements subjected to recognition do not contain any sudden twitches or very quick motion. The fastest exercise that I can think of is jumping with jumping rope, for which the world record averages on 0.18 seconds per one jump[5]. Normal person will take at least double of that time, which leaves us with about 0.4 sec per one exercise repetition. The measurement frequency of the sensors should be therefore high enough to capture a motion like this with enough of samples to perform recognition on. Setting measurement frequency to 50Hz should provide 20 samples for 0.4 sec long repetition, which seems reasonable.

The data has annotated *reference segmentation marks* that show where the repetition

[5] Calculatedfromvideoinhttps://www.youtube.com/watch?v=reJ45Z3HU9s

starts and ends. The data is also annotated into these 4 categories:

- `OP` - opening: movement that precedes the actual exercise motion, such as laying down on the ground before doing pushups. It is defined as 50 samples (1 second) before the first repetition.
- `EX` - exercise: the actual exercise motion
- `CL` - closing: movement that follows the actual exercise motion, such as getting up from the ground after doing pushups. It is again defined as 50 samples (1 second) after the last repetition.
- `FM` - free motion: all motion besides the previously mentioned.

2.4 Finding the best placement

It is not immediately apparent what is the best number of sensors and their placement on the body to get the most appropriate data. If only one sensor is used there are various exercises that would be impossible to recognize. Example: for the placement on the wrist we can find exercise where the arm will not move much (leg raises) or similarly for ankle (bicep curl ups). Therefore we can formulate an assumption that it is needed to use at least 2 distinct sensors, supported by [4].

The positions that have been used for capturing data are marked in Figure 2.6. There were 8 sensors available for capturing data, so I decided to use 2 for each arm, 1 for the waist and the rest on one leg, because I wanted to find out the influence of (anti)symmetry of placement.

The best positions for a given number of sensors will be selected based on the results of the entire algorithm.

2.5 Similar problems

Many fitness trackers on the market today use pedometers as basic movement recognition. Mechanical pedometers have been constructed as early as of 18th century: they work a bit like pendulum clocks (the ones with a swinging bar powered by a slowly falling weight). As the pendulum rocks back and forth, a kind of see-saw lever called an *escapement* flicks up and down and a gear wheel inside the clock (which counts seconds) advances by one position. So a pendulum clock is really a mechanism that counts seconds. The original pedometers used a swinging pendulum to count steps and displayed the count with a pointer moving round a dial (a bit like an analog watch). They are fixed on the waist

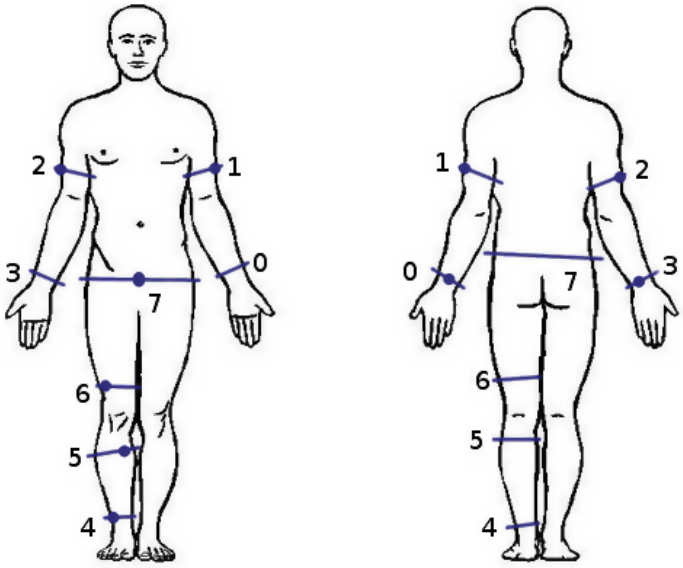


Figure 2.6: Placement of sensors on the body, large dots indicate the positions. Sometimes the placement is 0-indexed or 1-indexed, depends on the context.

and every time a step is taken the pendulum swings to one side then back again, causing a gear to advance one position and moving the hand around the dial.[6]

There is a patent for pedometers that use digital accelerometers[7] that is based on a trivial algorithm: it captures the data via AD converter, filters them, integrates the square of values and if they pass a certain threshold it increments the steps count. This type of thresholding is not precise at all and it will falsely label also other kinds of movement as steps.

Unfortunately I found it difficult to find other approaches of pedometer implementation because the internet is flooded with papers that compare the efficiency of all kinds of commercially available pedometers *without looking at how they actually work inside* (which I find as a very unrigorous approach). However, more accurate approaches can be based on the same methods as the described algorithm for counting reps.

The signal processing that is required out of a recognition system like the one that I'd like to create is similar to speech recognition. The algorithms that I am going to describe in Chapter 4 are former state-of-the art for speech recognition applications. The difference is that the input signal is much simpler, with lower frequencies, but it comes from more sources (audio is 1D signal, while the motion recognition deals with multidimensional signals).

[6]Explanation by http://www.explainthatstuff.com/how-pedometers-work.html
[7]https://www.google.com/patents/US7725289

Chapter 3

Hardware

Two different types of hardware were used throughout this thesis. They will be referred to as *measurement hardware* and *implementation hardware*. Both of these will be very briefly described in this chapter.

3.1 Measurement hardware

. It consists of 8 devices that can be placed on the body that act as *transmitters*. They measure the acceleration and gyroscopic data and send it wirelessly to a *receiver*. Because of problems of collecting data described in Chapter 7 two receivers had to be used that combine the received data. Following are short descriptions of the transmitters and receivers.

Transmitter

Input

- Barometer Freescale MPL3115
- 9-axis IMU sensor Bosch BMX055
- Configurative 4 pin connector Molex PicoBlade™
- 2x buttons
- Programming via TC2030-MCP-NL

Output

- 1x yellow LED
- 1x red LED
- 2.4 Ghz output of data in 32B packets at the frequencies 2420 - 2440 Mhz with GFSK modulation
- Expanding 8 pin connector Molex PicoBlade™

Receiver

Input

- 3x SMA antennas 2.4 Ghz
- 1x temperature sensor
- 2x buttons

Output

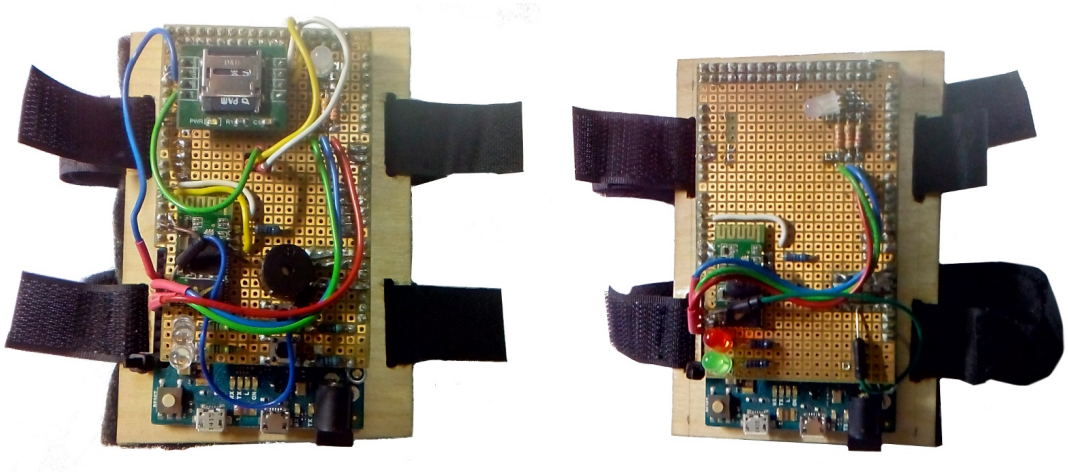
- 1x yellow and 1x red LED for each transmitter
- 4x virtual USB-COM port - 3x receiver, 1x configuration
- 8 connectors for connecting 8 transmitters and their time sync
- 1x USB connector

Figure 3.1: Photograph of one receiver and 4 transmitters.

3.2 Implementation hardware

Implementation hardware consists from 2 embedded devices. Arduino Dues[1] have been used for implementation in embedded hardware. The boards have been upgraded by accelerometer ADXL335[2] (with no gyro unfortunately), bluetooth module HC05, SD card reader and other components, such as LED lights, buttons or beeper. They are powered by external battery via microUSB connector.

The two devices act as a slave and a master. The slave only trasmits measured data via bluetooth to the master, which records all the data on SD card and evaluates the whole recognition pipeline.



(a) Master (b) Slave

Figure 3.2: Photographs of Arduino prototypes

[1] https://www.arduino.cc/en/Main/ArduinoBoardDue
[2] https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf

Chapter 4

Theory and analysis of algorithms

In this chapter I will introduce theoretical aspects of various algorithms used for signal processing, their advantages and disadvantages.

These are definitions of various terminilogies that will be used:

1. *Sequence* is an ordered collection of values.
2. *Input space* I is space as defined by Equation 4.1
3. *Input data* are $|I|$ sequences of the same length that may or may not contain repetitions.
4. *Repetition* is a motion described in Section 2.3, which are $|I|$ sequences of the same length that are a subset of the input data. Their beginning and end is marked in input data by indices called *segmentation marks*.

4.1 Input data

Data that comes from the sensors are 12 bit signed values in range of $V = \langle -2048; 2047 \rangle, V \subset \mathbb{N}$. A single sensor produces 6 ordered sequences of the same length (3 x,y,z values for accelerometer and 3 x,y,z for gyroscope). A selected combination of x sensors that are located at different parts of the body creates an input space

$$I \subseteq V^{x \cdot 6}. \tag{4.1}$$

So for combination of two bracelets, $x = 2$ and the $I \subset V^{12}$.

4.2 Simple types of analysis of signal

In this section we will look at various simple approaches for signal analysis such as thresholding, correlation and dynamic time warping. As it will be shown, these approaches are not sufficient for finding the number of repetitions and more complex approaches are needed to solve the recognition task.

4.2.1 Thresholding

A very simple way of counting reps is to use thresholding on the signal values. Such a approach may be useful for finding an occurenece of a large acceleration difference, for example tapping detection at the accelerometer. But after a single glance at the data (see Figure 4.1) it is immediately apparent that this is not going to work in this case, since the range of values of exercise can be very similar to those of free motion. This applies also for the use use of integration and then thresholding (algorithm already described in pedometer section 2.5 and illustrated in Figure 4.1).

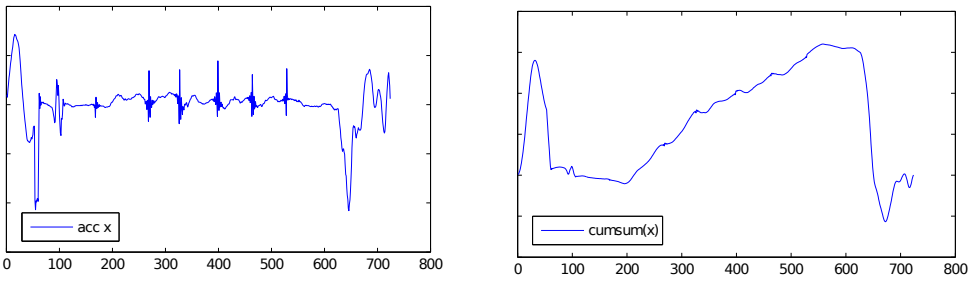


Figure 4.1: Thresholding of signal is not an appropriate method, even if using more combinations or decision trees for multiple signals.

4.2.2 Correlation

In signal processing, cross-correlation is a measure of similarity of two series as a function of the lag of one relative to the other. This is also known as a sliding dot product or sliding inner-product. It is commonly used for searching a long signal for a shorter, known feature.

For discrete functions f and g, the cross-correlation is defined as:

$$(f \star g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\ g[m+n].$$

Cross correlation is not appropriate for signals that can be warped in time, as is apparent in Figure 4.2 and therefore it is not a suitable method.

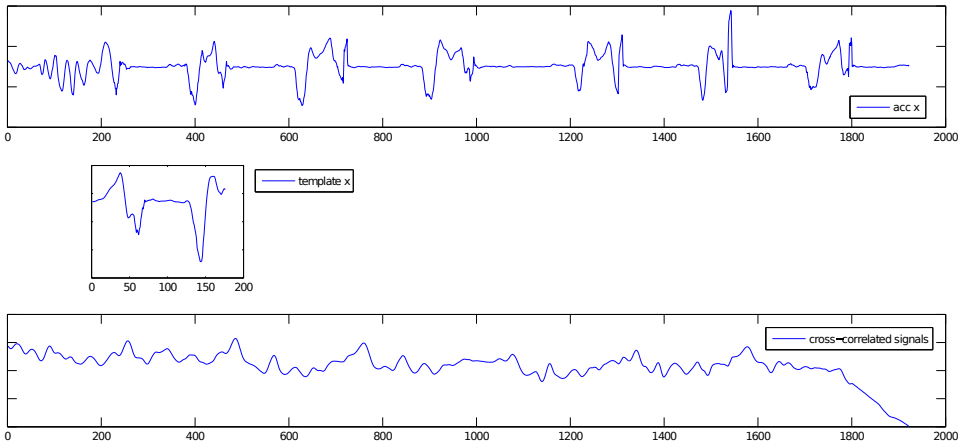


Figure 4.2: Cross correlating signals with exercise template does not produce any local maxima that have large difference to the rest of the signal and could be later thresholded.

4.2.3 Dynamic time warping

Dynamic time warping (DTW) is an algorithm for calculating similarity between two temporal sequences which may vary in time or speed. In general, DTW is a method that calculates an optimal match between two given sequences (e.g. time series) with certain restrictions. The sequences are "warped" non-linearly in the time dimension to determine a measure of their similarity independent of certain non-linear variations in the time dimension. This sequence alignment method is often used in time series classification. DTW can be easily implemented as a dynamic programming described in Algorithm 4.2.3.

The distance function $dist(a, b)$ can be arbitrarily defined, for signal alignment it is usually square of euclidean norm $dist(a, b) = \|a - b\|^2$. Time complexity of DTW is $\mathcal{O}(tr)$ for warping two sequences of length t and r. There are several ways that can be used to speed up the alignment, such as limiting the size of the warping window (so that the sequences do not contain large gaps, as shown in Figure 4.4) or reducing the input data.

One trouble is that DTW doesn't align sequences locally - it aligns across the whole sequence. If we'd like to use DTW to find a repetition as a close match to some exercise template it is necessary to know the starting and ending positions of tested subsequence.

DTW is used in [24] as the main algorithm for evaluating repetitions. However, their approach is not robust for more complicated motion, because they rely on threshold segmentation of acceleration to determine the starting and ending positions of tested subsequence. It is also neccessary to show the system when exercise started to occur, which I wanted to avoid completely. However, if it is known how to segment the exercise

itself from the motion before and after it, the DTW can produce *useful features* that can be used for a very simple classifier based on logisitic regression with good results [24]. The DTW is a method that will be used for improving accuracy of the recognition as described in Section 4.4.

DTW distance is not the only feature to use in classification. A distance of the warped path to the diagonal path has shown to be useful as well. For a warping window of size $t \times r$ (which are the lengths of the warped sequences) a diagonal path is a line that goes from point (1,1) to (t,r). The vertical (i) and horizontal (j) coordinates of the line can be calculated as

$$i(x) = \lceil (t/r)x \rceil \quad 1 \leq x \leq N \tag{4.2}$$

$$j(x) = \lceil (r/t)x \rceil \quad 1 \leq x \leq M \tag{4.3}$$

which "pixelates" the line.

The horizontal distance d_x is the sum of differences of the warped path's horizontal coordinates to the horizontal coordinates of the diagonal of the warping window, and vertical distance d_y is computed similarly.

Algorithm 1 Dynamic time warping

```
1:  procedure DTWDistance(r: array [1..n], t: array [1..m])
2:      DTW ← array [0..n, 0..m]
3:      for i = 1 .. n do                                   ▷ Initialization
4:          DTW[i, 0] ← ∞
5:      end for
6:      for j = 1 .. m do
7:          DTW[0, j] ← ∞
8:      end for
9:      DTW[0, 0] ← 0
10:
11:     for i = 1 .. n do                                   ▷ Compute table
12:         for i = 1 .. n do
13:             cost ← dist(r[i], t[j])
14:             ins ← DTW[i − 1, j]
15:             del ← DTW[i, j − 1]
16:             match ← DTW[i − 1, j − 1]
17:             DTW[i, j] ← cost + min(ins, del, match)
18:         end for
19:     end for
20:     return DTW[n, m]
21: end procedure
```

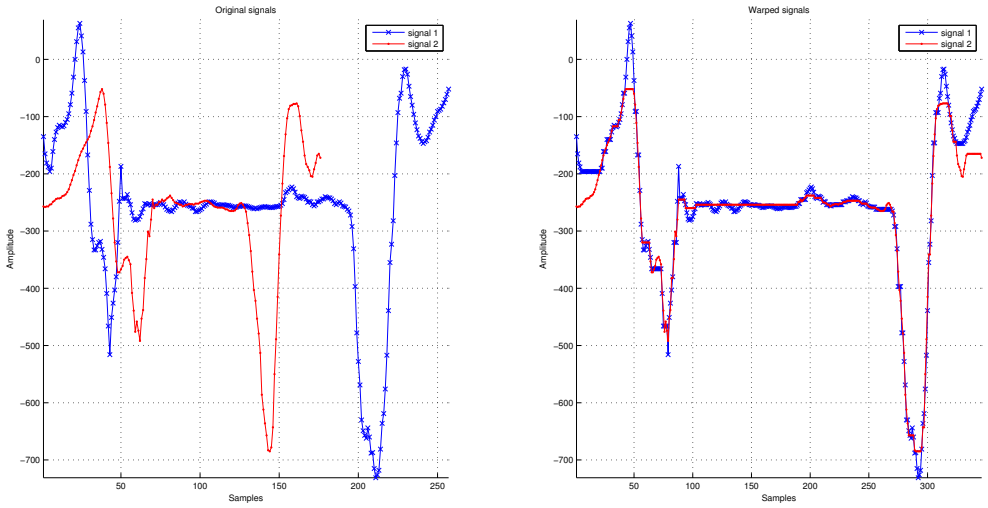


Figure 4.3: A burpee aligned to its template using one accelaration axis.

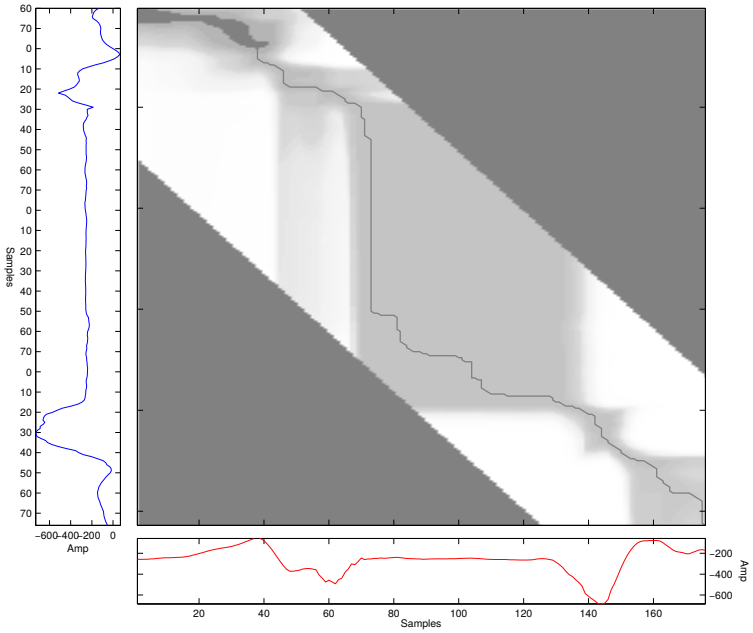


Figure 4.4: Warping window with restricted size from the diagonal.

4.3 Hidden Markov models

As the previously described simple types of analysis of signal are insufficient to reach the project goals, this section will introduce description of algorithms for training and evaluating Hidden Markov Models (abbreviated as HMM). In essence it is a probabilistic method which main aim is to infer what is the most probable hidden state of a system based on observations that the system produces. HMMs have been used in variaty of applications for recognition of patterns, such as gestures[8, 32], spoken words [15] or handwritten characteres [23].

A class of HMM, discrete HMM is of particular interest, because of its good performance, low complexity, and low computational demands that are a necessary requirement for implementation in embedded devices. It is an appropriate method to use in the case of motion recognition, because measured data from sensors can be used to infer what is the most likely real state of a person performing exercise.

One important aspect of using discrete HMM is vector quantization that creates observation sequences based on which HMM infers hidden state. Vector quantization will be introduced in the first subsection, and the description of HMM and its training will follow.

The training methods will be presented in general terms and later explained how they are going to be used for the motion recognition. The content of this section is based on the classic paper by [26] and thesis of [32, 22].

4.3.1 Vector quantization

A *vector quantizer* (VQ) is technically a method for compressing large input data spaces into its finite disjunct subsets. These subsets are characterized by subset "centers" called *prototype vectors* or *centroids*. The set of prototype vectors is called a *codebook*. The purpose of using vector quantization is in producing observation sequences that serve as input for discrete hidden Markov models.

Contrary to compression the main concern is not a compression ratio, which is usually high (the number of prototype vectors is typically low compared to the size of the input space), but the data loss that is suffered by their application.

Formally, a vector quantizer is a function

$$Q : I \to C,\ I \subset \mathbb{R}^n, C \subset \mathbb{R}^n, |C| = M \tag{4.4}$$

where I is original input space and

$$C = \{\mathbf{c_1}, \ldots, \mathbf{c_M}\} \tag{4.5}$$

is called a codebook of size M. In practice a set of measurements $X \subset I$ is available, and quantization is done using this set.

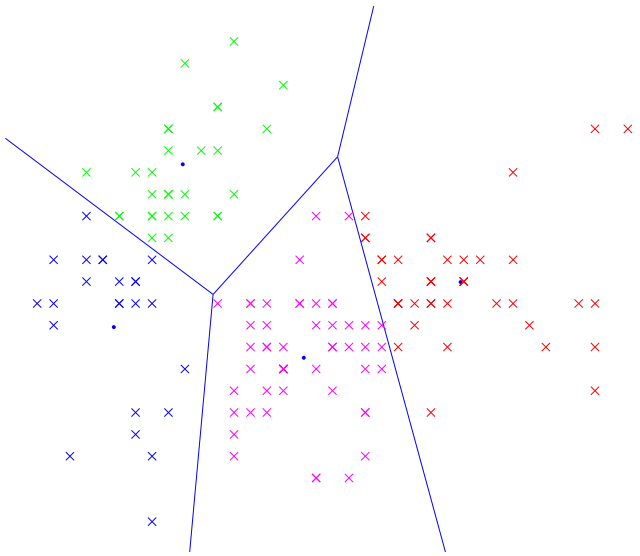


Figure 4.5: Voronoi diagram with 4 disjunct regions based on protope vectors.

Such assignment of each vector to its prototype vector basically splits the input space into disjunct regions R_i, and it is most interesting to know what is the index of assignment $i \in \{1 \dots M\}$ to region. One way of telling which region it is is by finding the closest protype vector

$$V(\mathbf{x}) = \arg\min_i \|\mathbf{x} - \mathbf{c_i}\|. \tag{4.6}$$

Such separation can be visualized using Voronoi diagrams (Figure 4.5).

To evaluate how well is the space separated we can define a quantization error over one vector $x \in X$

$$e(x|Q) = \|x - Q(x)\|, \tag{4.7}$$

and over the whole set of measurements X as

$$e(Q) = \mathbb{E}[e(X|Q(X))] = \int_X \|\mathbf{x}, Q(\mathbf{x})\| p(\mathbf{x}) dx, \tag{4.8}$$

where $p(\mathbf{x})$ is probability distribution of $\mathbf{x}$ in set X.

The most widely used method for vector quantization is algorithm k-means (also referred to as Lloyd's algorithm [19]), which locally iteratively minimizes the sum of distances of each vector from the disjunct subset $\mathbf{x} \in R_i$ to its prototype vector $\mathbf{c_i}$ (or in other words it minimizes the within-cluster sum of squares):

$$\arg\min_C \sum_{\mathbf{c_i} \in C} \sum_{\mathbf{x} \in X, \mathbf{x} \in R_{V(\mathbf{x})}} \|\mathbf{x} - \mathbf{c_i}\|^2 \tag{4.9}$$

The algorithm consists of several steps:

1. **Initialization**
   Find centroid seeds by random, or use a heuristic such as K-means++ [2].

2. **Assignment step**

$$R_i^{(t)} = \left\{ \mathbf{x_p} : \left\| \mathbf{x_p} - \mathbf{c_i}^{(t)} \right\|^2 \leq \left\| \mathbf{x_p} - \mathbf{c_j^{(t)}} \right\|^2, \quad \forall i, j \in \{1 \dots M\} \right\},$$

   where $\mathbf{x_p}$ is assigned exactly to one $R_i^{(t)}$, even if it could be assigned to more of them.

3. **Update step**

$$\mathbf{c_i}^{(t+1)} = \frac{1}{|R_i^{(t)}|} \sum_{\mathbf{x_j} \in R_i^{(t)}} \mathbf{x_j}$$

4. **Termination**
   Terminate if the assignment to regions $V(\mathbf{x})$ does not change or after specified number of iterations.

Because of its iterative nature, K-means converges to local optimum and may not find the global one, therefore it is recommended to re-run the algorithm several times and choose the quantization with minimum error.

Finding the optimal solution to the k-means clustering problem for observations in d dimensions is NP-hard in space d even for 2 clusters [1] and NP-hard for a general number of clusters k even in the plane [20]. The run time of k-means is $\mathcal{O}(nkdi)$, where n is the number of d-dimensional vectors, k the number of clusters and i the number of iterations needed until convergence. Using centroid seeds by k-means++ can speed up the algorithm and is that is $\mathcal{O}(\log k)$-competitive with the optimal clustering [2].

4.3.2 Discrete hidden Markov models

This section is an introduction to the theory of hidden Markov models. The practicalities of training the models is described in next section.

Hidden Markov Models are an extension of ordinary discrete Markov models. A discrete Markov model describes a system as a collection of distinct states that have quantified probabilities of transitions from one state to another. The states can be indexed with integers $1, 2, \dots N$ which refer to modelled phenomena. Let's denote the current state of the system in a time instant t as q_t.

Discrete Markov models are independent on history of system, a fundamental property called Markov Property which can be stated as

$$P(q_t|q_{t-1}) = P(q_t|q_{t-1}, q_{t-2}, ..., q_2, q_1). \tag{4.10}$$

The Markov model can be generalized in such a way that the probability of the current state can depend on more than one previous state; it can depend on n previous states. In this case we speak of *nth-order* Markov models, the model just described is *discrete-time first-order Markov model*.

The state transition probabilities a_{ij} may be defined as

$$a_{ij} = P(q_t = j | q_t = i) \quad i \in \{1, \dots, N\}, j \in \{1, \dots, N\} \tag{4.11}$$

with following properties:

$$0 \leq a_{ij} \leq 1 \quad \forall i, j \in \{1, \dots, N\}, \tag{4.12}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i, j \in \{1, \dots, N\} \tag{4.13}$$

The probabilities a_{ij} can be represented by a square *transition probability matrix* of size N. The initial state distribution is represented by probability vector

$$\pi = \begin{bmatrix} P(q_1 = 1) \\ P(q_1 = 2) \\ \vdots \\ P(q_1 = N) \end{bmatrix} \tag{4.14}$$

that sums up to 1 as well. The model can be described by a pair of parameters $\lambda = (A, \pi)$.

An extension to this Markov model is hidden Markov model. The extension is done in such a way that set of observations $O = (o_1, o_2, \dots)$ are are produced by each individual state q according to an associated probability function. As a result of the extension, hidden Markov models describe two-stage stochastic processes.

The first stage is an ordinary Markov model. It is this part that gave HMM its name because now the states q are not observed ? they are ?hidden?. The second stage is a collection of observation (or emission) probability distributions associated with each individual (hidden) state, i.e., each state produces (emits, or generates) an observation according to its distribution.

Consider a probabilistic function associated with the state i defined as:

$$b_i(k) = P(o_t = k | q_t = i) \tag{4.15}$$

where $P(o_t = k | q_t = i)$ denotes the probability that at time t the state i generates observation symbol k. The function $b_i(k)$ is called *emission probability* of state i with observation k.

The important property of the associated densities is that they are assumed to be independent of any previous state and generated observations, an extension of Markov property (4.10) i.e.:

$$P(o_t | q_t = i) = P(o_t | o_{t-1}, o_{t-2}, \dots, o_1, q_t, q_{t-1}, \dots, q_1) \tag{4.16}$$

It needs to be mentioned that (4.15) is based on discrete probability densities. But similarly it can be defined using continuous probability densities. They are usually modeled using gaussian mixture models and this type of model is often referred to as GMM-HMM.

The observation set O can be quite large – in the case of exercise recognition it is $2^{12} \cdot 6 \cdot (\#$ of used bracelets), which makes $4.8 \cdot 10^6$ different observations for two bracelets. We would like to reduce it to a limited number of representations, since such an ineffable large size would make the implementation of HMM unattainable.

This is where the vector quantization comes in. The codebook (4.5) can be used to limit the observation set, by creating a quantized observation set $O', |O'| = M$. Quantized observation o'_t is determined by finding closest prototype vector c to each observation o_t using (4.6) as

$$o'_t = V(o_t). \tag{4.17}$$

Now the emission probabilities can be represented by *emission matrix* B, where

$$b_{ik} = P(o'_t = k | q_t = i). \tag{4.18}$$

The index k represents the assignment to prototype vector c_k from codebook C.

The rows of B satisfy similar restrictions as in matrix A:

$$0 \leq b_{ik} \leq 1 \quad \forall i \in \{1, \ldots, N\}, \forall k \in \{1, \ldots, M\}, \tag{4.19}$$

$$\sum_{k=1}^{M} b_{ik} = 1 \quad \forall i \in \{1, \ldots, N\}, \forall k \in \{1, \ldots, M\}. \tag{4.20}$$

A fully defined discrete-time hidden Markov model is

$$\lambda = (A, B, \pi). \tag{4.21}$$

An illustrative example of using HMM for weather prediction can be found in [32], an example of urn and ball model in [26], or the occasionally dishonest casino in [16].

4.3.3 Structure of hidden Markov model

The structure of a Markov model is determined by the appearance of the matrix A as it defines the probability of transition between states. Transitions with $a_{ij} = 0$ mean that it is impossible to go from state i to j.

- The first type of structure is **fully connected** (Figure 4.6). It is a model in which it is possible to go from every state to every other state in a single step. This structure

is described by a matrix A for which holds:

$$0 < a_{ij} < 1 \tag{4.22}$$

$$1 \leq i, j \leq N \tag{4.23}$$

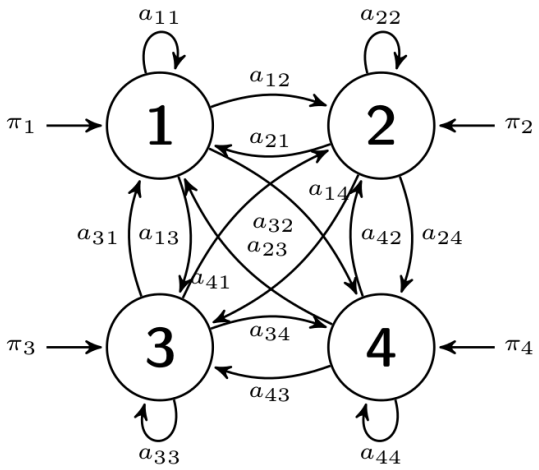


Figure 4.6: Fully connected HMM model.

- The second type of structure is called **left-to-right** or **linear** (Figure 4.7), and is based on assumption that the modelled system itself exhibits a chronological or linear structure. Therefore, it does not make much sense to allow every possible transition between states of the model. Typically these structures are used in the field of automatic speech and handwriting recognition [26, 9], but are applied for acclerometer based recognition with big successes as well [25, **?**].

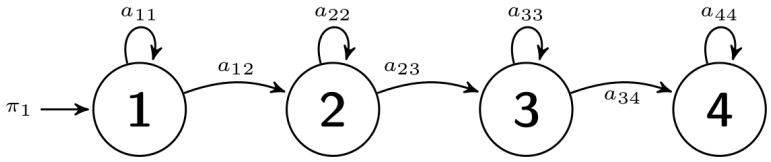


Figure 4.7: Left-to-right HMM model

The state transition matrix A of a left-to-right model have the following property:

$$a_{ij} = 0, j < i \tag{4.24}$$

The transition probabilities from states with higher index to lower index are set to 0. This implies that the model does not return to a state that has been already left. The self transitions are used to allow the model to capture the variations of a described pattern in time. This structure can be easily modified to allow skipping of individual states, which is reflected by the following constraint for elements of A:

$$a_{ij} = 0, j > i + \Delta \tag{4.25}$$

where Δ denotes the "length" of the skip. Left-to-right models with such transitions are called Bakis models. The Bakis model with $\Delta = 1$ such as the one depicted in 4.7 is the simplest one, and is also referred to as linear. The linear model has the following transition matrix A for size $N = 4$:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \tag{4.26}$$

Need to say, that the Bakis models nowadays are very much used in practical applications for their simplicity and tractability. This model has less parameters, which have to be estimated and calculated with, respectively, and it is also very flexible in representing a pattern.

It should be noted in advance that the structure of hidden Markov models does not influence (in any way) the algorithms that work with them, which is very convenient.

4.3.4 Training the model

The Baum-Welch algorithm represents one of the most commonly used optimization algorithms for hidden Markov models. It is one of the variants of expectation-maximization (EM) algorithms. This section contains the description of equations used in the the algorithm. Detailed derivation of these equations can be found in [32, 22]. Please note that this section serves only as a very quick review.

The Baum-Welch algorithm iteratively reestimates the parameters of a given model $\lambda = (A, B, \pi)$ on a certain training observation sequence O of length T such that at each iteration for the reestimated model $\hat{\lambda}$ holds

$$P(O|\hat{\lambda}) \geq P(O|\lambda). \tag{4.27}$$

Consider following definitions:

- *Forward variable* $\alpha_t(i)$, which represents how likely the given model λ describes the partial observation sequence $(o_1, o_2, \ldots, o_t)$ when the system is found in the state i at the time instant t:

$$\alpha_t(i) = P(o_1, o_2, ..., o_t, q_t = i|\lambda) \tag{4.28}$$

The forward variables can be calculated using a dynamic programming type of algorithm, called Forward algorithm:

$$\alpha_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N \tag{4.29}$$

$$\alpha_t(i) = \sum_{j=1}^{N} \alpha_{t-1}(j) a_{ji} b_i(o_t), \quad 1 \leq i \leq N, 2 \leq t \leq T \tag{4.30}$$

- *Backward variable* $\beta_t(i)$, which represents how likely the Markov model λ describes the partial observation sequence $(o_{t+1}, o_{t+2}, \ldots, o_T)$ given the system is in the state i at the time instant t:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, ..., o_T | q_t = i, \lambda) \tag{4.31}$$

The backward variables can be calculated similarly as forward variables using Backwards algorithm:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N \tag{4.32}$$

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_j(o_{t+1}), \quad 1 \leq i \leq N, 2 \leq t \leq T \tag{4.33}$$

$\beta_T(i)$ has been set arbitrarily to 1.

The algorithms presented involve computation with probabilities and this give rise to a problem of representation of a significantly small values on the computer. This is especially important in respect to using the algorithm in embedded device, since it is possible to use only single floating point number representations with range of values $\langle 1.17 \cdot 10^{-38}; 3.40 \cdot 10^{38} \rangle$, though scaling concerns mostly the training Baum-Welch algorithm which is performed offline on a normal computer, where double precision can be used with range of values $\langle 2.22 \cdot 10^{-308}; 1.79 \cdot 10^{308} \rangle$ (by IEEE 754 standard). The same argument applies however.

This can be solved with scaling the variables, introducing *scaled forward* $\hat{\alpha}$ and *backward variables* $\hat{\beta}$ that can be calculated using scaled forward and backward algorithms. The variables $\tilde{\alpha}$ and $\tilde{\beta}$ correspond to local unscaled forward variables.

Scaled forward algorithm

1. **Initialization**

$$\begin{aligned} \tilde{\alpha}_1(i) &= \pi_i b_i(o_i) && 1 \leq i \leq N \\ c_1 &= \frac{1}{\sum_{k=1}^{N} \tilde{\alpha}_1(k)} \\ \hat{\alpha}_1(i) &= c_1 \tilde{\alpha}_1(i) && 1 \leq i \leq N \end{aligned}$$

2. **Induction**

$$\begin{aligned} \tilde{\alpha}_t(i) &= \sum_{j=1}^{N} \hat{\alpha}_{t-1}(j) a_{ji} b_i(o_t) && 1 \leq i \leq N, 2 \leq t \leq T \\ c_t &= \frac{1}{\sum_{k=1}^{N} \tilde{\alpha}_t(k)} && 2 \leq t \leq T \\ \hat{\alpha}_t(i) &= c_t \tilde{\alpha}_t(i) && 1 \leq i \leq N, 2 \leq t \leq T \end{aligned}$$

Scaled backward algorithm

1. **Initialization**

$$\tilde{\beta}_T(i) = 1 \qquad 1 \leq i \leq N$$
$$\hat{\beta}_T(i) = c_T \tilde{\beta}_T(i) \qquad 1 \leq i \leq N$$

2. **Induction**

$$\tilde{\beta}_t(i) = \sum_{j=1}^{N} \hat{\beta}_{t+1}(j) a_{ij} b_j(o_{t+1}) \qquad 1 \leq i \leq N, 2 \leq t \leq T$$
$$\hat{\beta}_t(i) = c_t \tilde{\beta}_t(i) \qquad 1 \leq i \leq N, 2 \leq t \leq T$$

Finally, given all these formulas the pseudocode of the Baum-Welch algorithm can be presented (for derivation please look at cited material):

1. **Initialization**

$\lambda = (A, B, \pi)$ the initial model to be optimized
O the training sequence
tol the minimum value by which every model's parameter has to change
$plogP = 1$ *a*

2. **Optimization**

$logP = \log P(O|\lambda)$ computed by the scaled forward algorithm
$\hat{\alpha}_t(i)$ computed during the previous step
$\hat{\beta}_t(i)$ computed during by the scaled backward algorithm

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) a_{ij} b_j(o_{t+1}) \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{c_t}} \quad \text{reestimate } \hat{A}$$

$$\hat{b}_t(o_k) = \frac{\sum_{t:o_t=o_k}^{T} \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{c_t}}{\sum_{t=1}^{T} \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{c_t}} \quad \text{reestimage } \hat{B}$$

$$\hat{\pi}_i = \frac{\hat{\alpha}_1(i)\hat{\beta}_1(i)}{\sum_{i=1}^{N} \hat{\alpha}_1(i)} \quad \text{reestimate } \hat{\pi}$$

$\hat{\lambda} = (\hat{A}, \hat{B}, \hat{\pi})$

3. **Termination**
If $\frac{|plogP - \log P|}{1+|plogP|} \geq tol$ OR $\max_{1 \leq i,j \leq N} |\hat{\alpha}_{ij} - a_{ij}| \geq tol$ OR $\max_{1 \leq i \leq N, 1 \leq k \leq M} |\hat{b}_i(k) - b_i(k)| \geq tol$ OR $\max_{1 \leq i \leq N} |\hat{\pi}_i - \pi_i| \geq tol$ then continue optimization, otherwise terminate.

4.3.5 Finding hidden states

The Viterbi algorithm is an efficient method of finding an optimal state sequence q^* for a particular observation sequence O, and model λ. It is another dynamic programming type of algorithm very similar to the forward algorithm, and it uses the following criterion to determine the optimality of a state sequence:

$$q^* = \underset{q \in \{1,2,\dots N\}}{\arg\max}\ P(O, q|\lambda) \tag{4.34}$$

Consider the following iteration variable:

$$\delta_t(i) = \max_{1 \le q1, q2, \dots, q_{t-1} \le N} P(o_1, o_2, \dots, o_t, q_1, q_2, \dots, q_t = i|\lambda) \tag{4.35}$$

or in other words the probability of observing a partial sequence $(o_1, o_2, \dots, o_t)$ produced along the state sequence $(q_1, q2, \dots, q_t)$ such that the sequence ends in state i at the time t. It can be verified by induction that the following holds for $\delta_t(i)$:

$$\delta_t(i) = \max_{j \in \{1,2,\dots N\}} \delta_{t-1}(j) a_{ji} b_i(o_t) \tag{4.36}$$

Generally, the Viterbi algorithm iteratively computes $\delta_t(i)$ according to 4.36, and stops after reaching $\delta_T(i)$. In order to find the optimal state sequence, it has to memorize the argument that maximizes $\delta_t(i)$ for each state i and time t. Then, after the algorithm finishes, by using backtracking the state sequence is reconstructed.

An alternative Viterbi algorithm is introduced for the same reasons as in previous section, when the $P(O, q^*|\lambda)$ becomes too small to be numerically represented on computers. The alternative Viterbi algorithm applies logarithmic transform. Pseucode follows:

1. **Preprocessing**

$$\tilde{\pi}_i = \log(\pi_i), \quad 1 \le i \le N \tag{4.37}$$
$$\tilde{a}_{ij} = \log(a_{ij}), \quad 1 \le i, j \le N \tag{4.38}$$
$$\tilde{b}_i(t) = \log(b_i(t)), \quad 1 \le i \le N, 1 \le t \le T \tag{4.39}$$

2. **Initialization**

$$\tilde{\delta}_1(i) = \tilde{pi}_i + \tilde{b}_i(o_1), 1 \le i \le N \tag{4.40}$$
$$t = 2 \tag{4.41}$$

3. **Induction**

$$\psi_t(i) = \underset{1 \le j \le N}{\arg\max} \left(\tilde{\delta}_{t-1}(j) + \tilde{a}_{ji} + \tilde{b}_i(o_t), \right), \quad 1 \le i \le N \tag{4.42}$$
$$\tilde{\delta}_t(i) = \tilde{\delta}_{t-1}(\psi_t(i)) + \tilde{a}_{\psi_t(i)i} + \tilde{b}_i(o_t), \quad 1 \le i \le N \tag{4.43}$$

4. **Update**
   If $t < T$ then $t = t + 1$ and go to step 3. Otherwise go to step 5.

5. **Sequence reconstruction**

   (a) **Initialization**

$$q_t^* = \underset{1 \leq i \leq N}{\arg\max}\, \tilde{\delta}_T(i), \quad 1 \leq i \leq N \tag{4.44}$$

$$\log P(O, q_t^* | \lambda) = \tilde{q}_T(q_t^*), \quad 1 \leq i \leq N \tag{4.45}$$

   (b) **Backtracking**

$$t = t - 1 \tag{4.46}$$

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \tag{4.47}$$

   (c) **Termination**
   If $t \geq 1$ go to step 5b. Otherwise, terminate and return $\log(P(O, q_t^* | \lambda))$ and q^*

The whole observation sequence O is not be available if the algorithm should run in real time. This is when windowed alternative Viterbi algorithm can be introduced. The basic idea is that the sequence O is separated to consecutive subsequences $\Theta_1, \ldots, \Theta_n$ of constant length T, such that $\bigcap_i \Theta_i = O$. The alternative Viterbi runs normally on the first Θ_1. The next $\Theta_{i>1}$ then use the variable $\delta_T(i)$ from previous run as the input to the next run, such that

$$\tilde{\pi}_i = \delta_T(i), \quad 1 \leq i \leq N \tag{4.48}$$

The windowed approach does not compute optimal sequence and there are can be found examples where this approach fails. Consider following simple example:

$$A = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0.98 & 0.01 & 0.01 \\ 0.01 & 0.98 & 0.01 \\ 0.01 & 0.01 & 0.98 \end{bmatrix} \quad \pi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.49}$$

$$\lambda = (A, B, \pi) \qquad O = (1, 1, 3, 3) \tag{4.50}$$

It is a linear model that emits three observation symbols. The optimal sequence of hidden states for observation O computed by non-windowing is $q^* = (1, 2, 3, 3)$.

However, with window size 2, the subsequences are $\Theta_1 = (1, 1), \Theta_2 = (3, 3)$ and the windowed approach yields sequence $\hat{q} = (1, 1, 1, 1)$.

In practice this is not such a big problem for a large enough window, as shown later in Section 5.4

4.4 Improving accuracy of HMM

One way of looking at the hidden Markov model is that it is a segmentation tool to parse the input to segments called *candidates* that are either `exercise` or other `free motion`. In practice, the HMM produces a lot of false positives (free motion). A classifier trained on features extracted from the candidates can be later used to discern whether the candidate is free motion or repetition of exercise.

The work of [24] shows that it is possible to extract simple statistical features of these candidates (such as mean, standard deviation, maximum, minimum, etc.) that can be used for classification with logistic regression.

Other approach that I tried is to use histogram of values in the repetition, with bins spanning linearly or somewhat "logarthmically", with denser bins for smaller values around which the repetitions are mostly distributed and sparse bins for larger values. The histogram was then normalized and served as input to neural network. This wasn't however successful and had very low success rate, so I don't mention it later in evaluation.

It is important that the HMM is able produce candidates with high accuracy. The training data has labels for the start and end of each repetition, an interval I_r. These can be compared to the start and end positions of candidate, an interval I_c using error function

$$e(I_r, I_c) = \frac{I_r \cap I_c}{I_r \cup I_c} \tag{4.51}$$

A candidate is considered a *match* if $e(I_r, I_c) \geq 0.9$, and being *close* if $e(I_r, I_c) \geq 0.7$. Below the threshold of 0.7 it is considered as a false negative.

One important part of [24] is the use of an exercise template to which the candidate is aligned to to calculate the features.

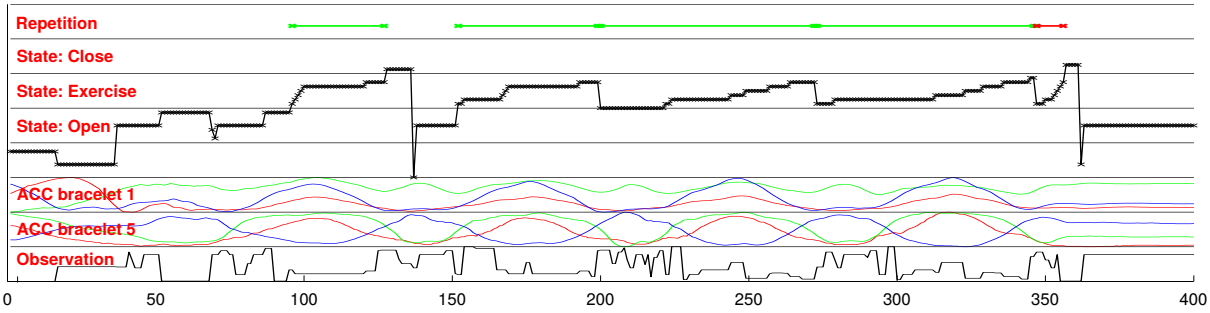


Figure 4.8: Segmentation from HMM results in false positives which should be discarded.

4.4.1 Exercise repetition template

Let's try to find a repetition called *template* that would be the best representant of the entire exercise for all people, from all available data. This would be useful for better placing reference segmentation marks and for comparing exercise candidates with this template to classify the movement.

One could think of trying to construct this template from multiple sources, from available segmentation marks for repetitions. However, this segmentation is not always properly done because of human error during the capture of the data. There can be occurence of some outliers amongst the repetitions, but on average the reference segmentation marks are good.

Using multiple sources would be similar to finding DNA multiple sequence alignment. One way of searching for such a template would be running DTW multiple sequence alignment and taking the final warped sequence. However, it is not feasible: given k sequences of length n the time complexity is $\mathcal{O}(k^2 2^k n^k)$ if we use sum of pairs. Since time complexity of dynamic programming approach is exponential in the number of sequences, heuristic methods are usually used. Another problem is that the DNA alignment is used with a small set of 4 symbols. The number of different values is in the order of 2^{12} (size of the input space, see 4.1) and is not practical at all. However, a perfect template is not needed, just something that would be "average enough" and not some outlier.

So this is the proposed method: let's have a set of N repetitions called R, in which each repetition is a set with $dim(I) = x \cdot 6$ time series (Equation 4.1). We want to find a repetition r_T, a *template*, that is the most similar to all other in the set.

$$r_T = \arg\min_{x \in R} \sum_{r_i \in R} sim(x, r_i). \tag{4.52}$$

How to create the similarity metric? It is not feasible to use multidimensional DTW alignment for each time serie because of memory complexity. This can be reduced to usage of only one axis for the alignment. Let's take the one that has the highest variance on average througout repetitions. It is not necessarily the most descriptive axis for the given exercise, but it is good enough" for good alignment. For the contrast, axis with low variance means there is a small number of local extremes on which DTW can make a good match and lower the overall DTW distance.

4.4.2 Multi layer perceptron

Multi layer perceptron [27] (abberviated as MLP) is very well known classifier based on neural network architecture. They are well described for example in [11] or in many machine learning books, so I won't go into great details about them. They have a high expressivity, in fact they have been shown to be able to represent any kind of (non-linear) functions from the input space to the output space with sufficient amount of

hidden neurons and arbitrary bounded and nonconstant activation function [14]. Training of MLP (or neural networks in general) requires tuning of hyper parameters, such as weight decay, learning rate or selecting appropriate gradient-descent algorithms (such as Stochastic Gradient Descent[28], AdaGrad[7], RMSprop[31] or others). It is important to evaluate accuracy of the network on a validation set while training to avoid overfitting. L2 regularizers are often used to improve generalization.

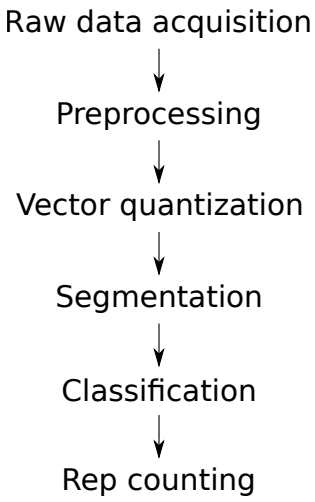
4.4.3 Updating precision of segmentation marks

The segmentation marks cannot be obtained precisely, human error occurs during the capture of the data and it is important for the proper training of the algorithm to have good source of data. The mistakes are however not large, and we can update the segmentation marks only locally. Once we have the template, we can iteratively move the the starting and ending positions within a small range to find placement that minimizes DTW distance to the template, and use these marks as the reference segmentation marks.

Chapter 5

Recognition system design

The process of recognition can be separated into these steps (modules):



The steps of this pipeline will be described in each section in this chapter, using algorithms introduced in Chapter 4.

5.1 Raw data acquisition

The raw data comes from 9-axis IMU sensor Bosch BMX055[1] with acceleration range $\pm 4g$, gyroscope range $\pm 250°/s$ and 12bit resolution.

The raw values are *not* converted into their respective physical units because it would require introducing floating numbers to this stage, which would be inefficient for memory

[1] Datasheet: http://www.mouser.com/ds/2/621/BST-BMX055-DS000-01v2-371988.pdf

and computation. If other sensor is used or with different range of values they must be scaled accordingly (this was neccessary for Arduino implementation, as there is another sensor that different acceleration range). The orientation of the axis is in Figure 5.1. This orientation has been used on all 8 locations on the body (Figure 2.6), from the point of view of the person that was wearing the sensors.

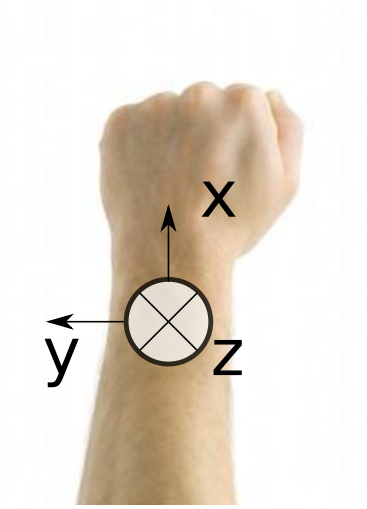


Figure 5.1: Orientation of the axis of sensor.

5.2 Preprocessing

The data is preprocessed using window averaging. The window size is 8 time steps large. This filter has a tiny delay of 0.16 seconds because of the filter starting up. This can be completely ignored in the implementation in the embedded device.

Since the orientation of the sensors is fixed relative to the body, there is no need for transformation of coordinates as in the work of [32]. Normalization is also not required, since it is strictly specified what the range of used values is.

5.3 Vector quantization

Theoretical aspects of vector quantization have been discussed in Section 4.3.1. The available data for vector quantization comes from the original input space, see 4.1.

The question is, what is the best input for the quantization? Or rather, which observations from the sequence $O = (o_1, o_2, ...o_n), o_i \in I$ is the most appropriate to take to produce quantized observations $O' = (o'_1, o'_2, ..., o'_n))$? Now one might be tempted to use more than a single value from this multidimensional time-series, such as a windowed subsequence (such that window of length w takes input $(o_t, o_{t+1}, ..., o_{t+w})$ and produces one o'_t). This approach is however useless [17]. Thanks to my random browsing of scientific papers I stumbled upon this information and I couldn't believe it, so I decided to test this statement and create another empirical evidence. However, it is not a rigorous approach without supporting theoretical foundation, just sort of an intuition, so it should be taken with some reserve.

The purpose of VQ is to produce observation sequences. They are *the only input* to HMM, so it is important that the observations are well produced – in the sense that the observations for exercise are quite similar to each other and distinct from observations for the free motion. Also it is a good idea that the observations are versatile within the sequence o', because of the windowed Viterbi algorithm (discussed in Section 4.3.5).

Since the length of observation sequence O' is varied and is about $|O'| = 100$ steps (depending on exercise), it is useful to quantify them somehow. If we count the occurences of each observation symbol and divide it by the length of the sequence we get a probability of occurence of each symbol within given sequence. This allows to directly compare observation sequences of different length. Let's call this quantity q (I can't think of a better name, so just call it q). It logically holds that $|q| = |C|$ (size of codebook).

Similar observation sequences O'_1, O'_2 will have a very similar q_1, q_2, and therefore the distance $\|q_1 - q_2\|$ will be very small, and vice versa for not so similar sequences. This distance can be used to assess the quality of VQ. The average distances of q within exercise observations should be smaller than distances of q to free motion observations to achieve good separation.

As it can be seen in Figure 5.2, there are several "squares" that correspond to similar exercises as they have been done by the same person in one exercise set. The average sum of all pairs of q_i within the "square" is by order of magnitude lower than to the sum of distances to free motion observations, which supports expected properties for this evaluation metric.

The types of input in Figure 5.2 are {`single`, `window`} and {`no difference`, `difference`}.

- `Single, no difference` means there is only one value (o_t) to produce o'_t,
- `Window, no difference` means two values (o_t, o_{t+5}) are used to produce single o'_t,

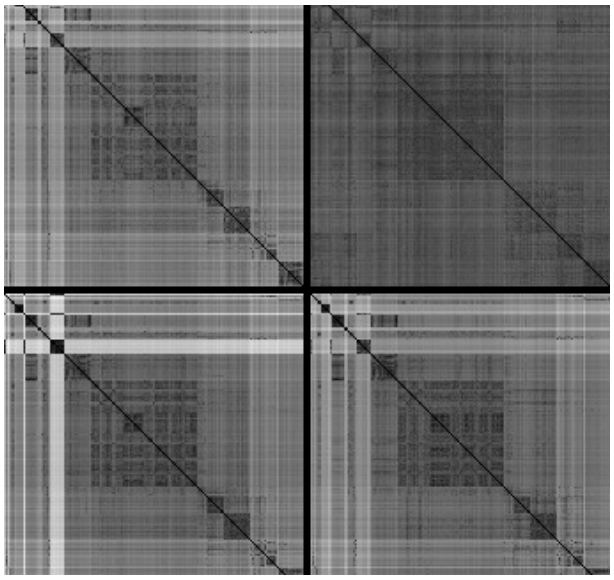


Figure 5.2: Visualization of distance matrices for each type of VQ.

Position	Input type	Within exercise distance	Distance to free motion
Top left	`single, no difference`	34.5709	103.6928
Top right	`single, difference`	23.8501	62.4549
Bottom left	`window, no difference`	34.2475	97.8390
Bottom right	`window, difference`	32.6048	103.7762

- `Single, difference` means value $(o_{t+5} - o_t)$ is used to produce single o'_t,
- `Window, difference` means two values $(o_t, o_{t+5} - o_t)$ are used to produce single o'_t.

The window length is chosen to $w = 5$ because it is sufficiently small window to capture the variation in signal (5 timesteps correspond to 0.1 seconds).

As it can be seen from the calculated values, there is isn't a large difference between the types {`single no difference`, `window no difference`}, and it supports the surprising argument of [17].

As the VQ type with the lowest exercise distance is `single, difference` it is the one that will be used. It has also one other nice property: if the person is not moving, the quantized observation will belong always to the same symbol, invariant to his physical position.

5.4 Segmentation

The hidden Markov models from Section 4.3 are introduced. However the structures described in 4.3.3 are not sufficient by themselves for task segmentation, because they describe a whole sequence, not its parts.

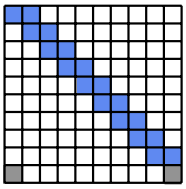
Therefore two new models are introduced:

1. **Simple** model is based on the linear model. The last state does not finish in a self-loop, but goes back to the first state. Example matrix A for 4 hidden states:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ \epsilon & 0 & 0 & 1-\epsilon \end{bmatrix} \tag{5.1}$$

   The model is trained as a linear model on the training observations which are segmented into repetitions. The last transition is then changed to value ϵ, or $1-\epsilon$ respectively.

   I can visualize the transition matrix for better grasping of the structure:



   The blue cells in the grid are transitions from the original partial model, while the gray cells are the transitions probabilities. White cells are transitions that are not allowed (with zero value).

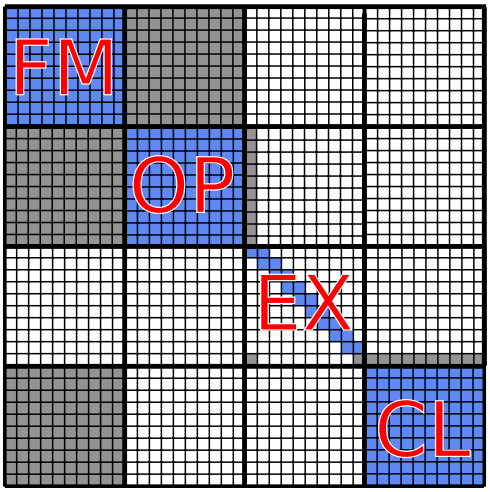
2. **Composed** model is based on the idea that the motion can be represented by 4 distinct parts (Section 2.3): `free motion` (`FM`), `opening` (`OP`), `closing` (`CL`) and `exercise` (`EX`). The first three can be described by fully connected models and the last by a linear model (or rather, the *simple* model, which is "recycled"). These *partial models* are trained separately for each observations corresponding to the parts of the motion. Then they are joined together in the whole composed model. The transitions between two partial models X and Y are following:

   (a) `full - full` any state from X can go to any state of Y

   (b) `full - linear` any state from X can go to the first state of Y

   (c) `linear - full` last state from X can go to any state of Y

   (d) `linear - linear` last state from X can go to the first state of Y



The composed model with corresponding transitions can be depicted graphically as *composed transition matrix* A:

The emission matrices $B_{partial}$ are simply stacked vertically to create a *composed emission matrix* B. The compose initial probability distribution is assumed to be 1 for the first state in partial model `FM` since movement always starts with free motion.

These models require learning newly introduced transitions.

1. The transition value ϵ in the simple model can be estimated from the average number $\bar{n}$ of occurences of the last hidden state on the repetition data. If we assume that after each repetition there is again another one, or in other words the first hidden state occurs again, the probability of transition $\hat{\epsilon} = 1/\bar{n}$.

2. The transitions in the composed model are more tricky. Basically they are transitions between the whole *partial models*, so the models must be scaled down by some value η to leave space for these new transitions so rows sum up to 1. Then this room left by η can be (evenly) distributed amongst the transitions values. It is likely that probability transitions of type `full - full` will not be distributed evenly, and it would be appropriate to change them so that the model maximizes its likelihood $P(O|\lambda)$.

   The value η can be estimated from annotated data of hidden sequences (Section 2.3). It can be thought of as a transition probability between these partial models. If hidden parts A and B produce on average n_A and n_B occurences then the transition $\hat{\eta}_{AB} = \frac{n_A}{n_B}$.

These complex models can be again retrained using Baum-Welch to locally optimize them for higher model likelihood.

The most appropriated structure is selected in next chapter. The hidden states correspond to one of four stages of exercise: {`start`, `middle`, `end`, `other`}. Each stage has hidden

states that are associated with it. Once the sequence of hidden states succesfully passes all first three stages consecutively, the timesteps at which the repetition started and ended are used for segmentation.

The non-optimality of windowed Viterbi is mentioned in Section 4.3. This problem however is not so frequent when the codebook size is much larger and the observation sequences are highly varied (there is not a long repetition of the same observation symbol). One can intuit it for a more complicated example as following: let's have hidden states q_a, q_b, q_c with the same emission probabilities $b_a = b_c$, different b_b, linear model going from q_a to q_b to q_c and initial state q_a. When the observation symbols that are more probable for distribution b_a than b_b occur, it is "more difficult" to determine whether the correct state is a or c for a windowed approach rather than for non-windowed. However, if the distribution b_a is very different to b_c, the algorithm will calculate a transition to state q_c (through q_b).

The exercise model is linear, and it is not important to find the *exact* hidden state, as long as the hidden state corresponds to `middle` stage of evaluation. The states in the `start` and `end` stages have very different emission probabilities than those for `middle` and the mistake of non-optimal decoding generally does not occur.

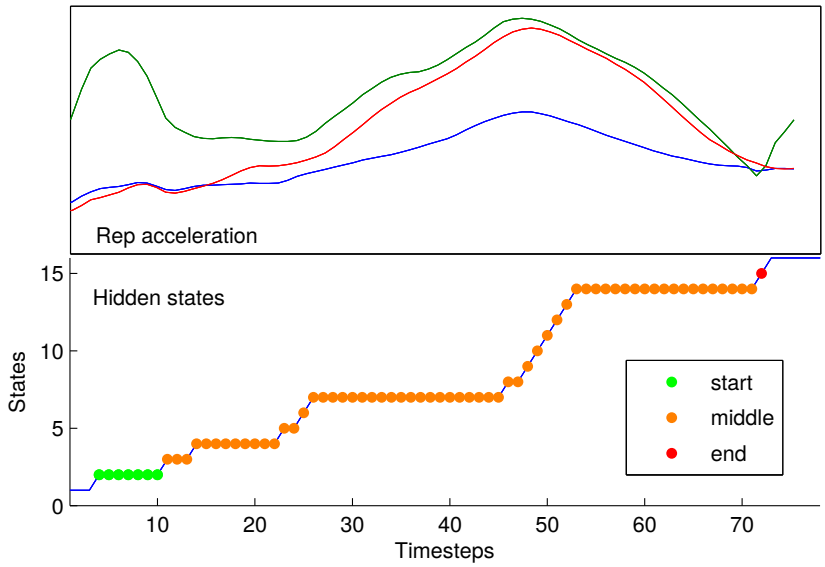


Figure 5.3: Stages of hidden states for simple model of size N=16. Rep starts at time $t = 4$ and ends at $t = 72$ for states 2 and 15.

5.5 Classification

The segmented sample is classified to either `exercise` or `free motion`. At first, samples that are too short or too long are discarded immediately. Then, inspired by [24], dynamic time warping is used to extract following features:

- `mean, standard deviation, max, min` on unwarped (original) signal for each axis
- `dx, dy`, warping distances as explained in Section 4.2.3 (Equation 4.2).

DTW is quite an expensive operation, therefore some reduction must be done in order to deploy it on embedded system. Template with sampling frequency 50Hz contains too much information and can be downsampled. A useful subsampling rate has been found to be 2×, which is also very simple to implement.

Warping for each axis of measurement would multiply the running times linearly, and is not necessary as it can be used only in one axis. The selected axis is one whose template has the highest difference between maximum and minimum in the signal as it is usually one to which the most informative warping can be done.

These features are then fed into fully connected MLP with one hidden layer of 16 neurons. It classifies the sample in one forward pass, which is not a costly operation.

5.6 Rep counting

Once the segemented sample is classified as `exercise`, the repetition counter is incremented.

Chapter 6

Implementation

6.1 Datalogger

I wrote data logging application implemented in .NET to record data and repetition annotations. The code uses multiple threads to receive and parse data for each transmitter. GUI shows its current status, along with the training plan that the sportsman should perform and a large button for annotation. The user interface is shown in Appendix A.1.

6.2 Offline training/evaluation

All of the algorithms described in the Chapter 4 have been implemented in Matlab, along with data preprocessing, evaluation framework that distributes the training and testing data, visualisation procedures, and data analysis. Gluing data from different transmitters is implemented in a simple script in PHP. Efficient 3D DTW for experiments is implemented in MEX (C) code, interfaced with Matlab.

Third-party code such as SVM, MLP, PCA or TSNE comes from standard or publicly available Matlab packages without prohibitive license restrictions.

6.2.1 Directory tree

```
/
├── algorithms ..... List of algorithms
│   ├── codebook ..... Train codebook
│   ├── hmm_hidden ..... Generate hidden state annotations
│   ├── hmm_observations ..... Generate observations for given codebook
│   ├── hmm_collect ..... Collect hidden and observation data
│   ├── hmm_model ..... Train and test model
│   ├── dtw ..... Train and test classifier
│   └── exportmodel ..... Export model for arduino
├── analysis ..... Analysis of data and results
├── data
│   ├── raw ..... Data from sensors
│   ├── united ..... Merged together from receivers
│   └── exercise ..... Properly segmented data with labels
├── eval ..... Evaluation framework
├── helpers ..... Common functions
├── preprocess ..... Preprocessing from raw data
├── results
└── visualisations ..... Show graphs
```

6.2.2 Data format

Processed data is stored in `/data/exercise/`*`exercise_name`*`/labeled/`. The CSV files have 49 columns, with 6×8 groups. Last column is the repetition annotation (A), where "1" is a start of repetition and "2" is the end (both inclusive), while "0" is no label.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
Sensor 1						Sensor 2						Sensor 3						Sensor 4						Sensor 5						Sensor 6						Sensor 7						Sensor 8						A
acc		gyr		acc		gyr		acc		gyr		acc		gyr		acc		gyr		acc		gyr		acc		gyr		acc		gyr		acc		gyr		acc		gyr		acc		gyr			

6.3 Embedded device

Embedded devices implement only model evaluation, not its training. The C++ implementation is an efficient transcription of Matlab code for preprocessing, alternative Viterbi algorithm and candidate classifier. It is optimized to use the least amount of memory as possible. Matlab is used to generate the entire model with all necessary matrices and other settings and dumps them into a binary file. This file is read from SD card on the embedded device.

The program loads automatically one type of exercise and after starting the recognition

by pressing a button beeper buzzes after recognizing one repetition.

Since this was not a required part of this thesis I will not get involved into too many details about the implementation, but I am open for requests for demonstration :-)

Chapter 7

Evaluation

7.1 Dataset

The selected exercises $E, |E| = 7$ are:

1. burpees (`BUR`) - http://michal.sustr.sk/exercise/bur.mp4
2. sit-ups (`SIT`) - http://michal.sustr.sk/exercise/situps.mp4
3. squats (`SQT`) - http://michal.sustr.sk/exercise/sqt.mp4
4. pushups (`PUP`) - http://michal.sustr.sk/exercise/pup.mp4
5. jumping jacks (`JCK`) - http://michal.sustr.sk/exercise/jack.mp4
6. raising arms (`ARM`) - http://michal.sustr.sk/exercise/armup.mp4
7. uppercuts (`CUP`) - http://michal.sustr.sk/exercise/cuts.mp4

Their example inputs are visualized in Appendix B.

I decided to use these because they are symmetrical unconstrained exercises with large range of movement that most people can do easily. The symetricity is useful for finding optimal locations of sensors, since it will not favor one half of the body over the other. It is more difficult to count number of repetitions for unconstrained exercises than for constrained exercises (which are mostly performed on machines in the gym, such as leg press, butterfly, but also bench press or bicep curls) because they have limited scope of motion. The constraint causes the measured data to be much more smooth and easier to evaluate and is not as spread out (see Figure 7.1). The idea is that if algorithms are successful in counting of unconstrained exercises, they will be successful also in the constrained case, since it's a simper task.

In machine learning tasks it is always highly recommended to visualize your data before doing any work with them. This data is quite high dimensional and techniques for dimensionality reduction such as PCA or TSNE can be used for approximate visualization, see Figure 7.1.

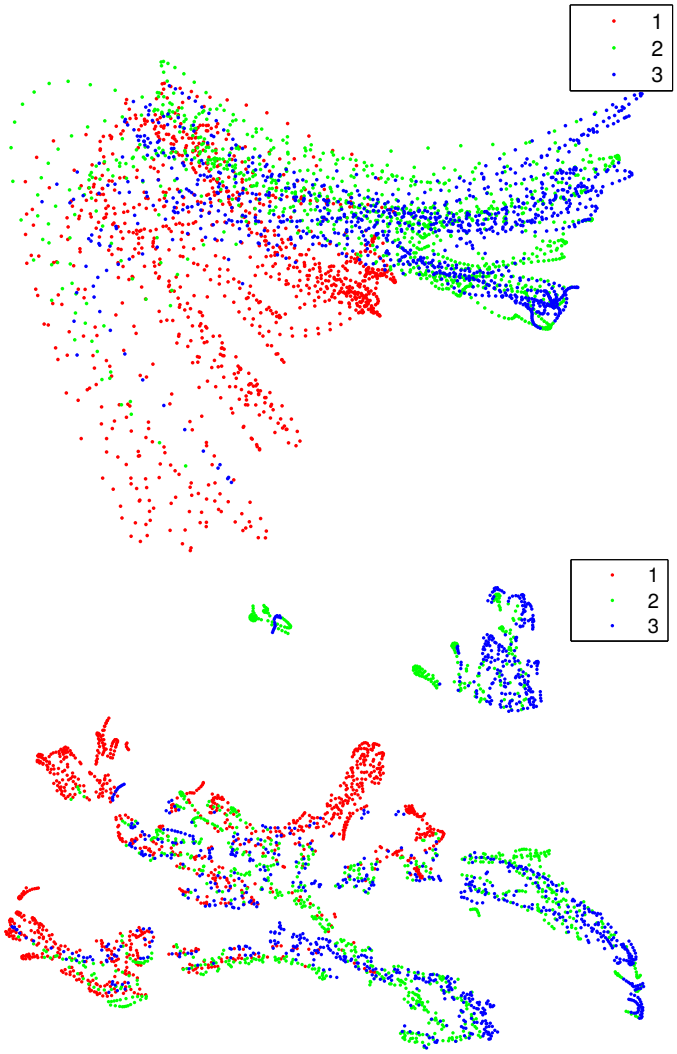


Figure 7.1: Visualisation of acceleration values of exercise `raising hands` (`ARM`) using PCA (top) and TSNE (bottom). The different colors represent the stages of movement in time (first, second and third 1/3). The motion is highly varied throughout the different stages and it cannot be easily clustered in a small number of clusters whose index would correspond to the progression of motion through time. This is a support for using clustering approach over the entire dataset using methods such as k-means.

The whole dataset consists of exercise from 10 different people and was divided into training, validation and testing set (which have been taken as IID samples from the whole dataset). The model parameter selection results are computed on the validation set, and overall accuracy on the testing set.

There are following numbers of captured data. The rep sets consists of multiple repetitions in a row with varying length, or only one repetition.

Exercise	**Rep sets**	**Reps in total**	**Training**	**Testing**	**Validation**
BUR	54	135	90	27	18
SIT	40	152	110	24	18
SQT	36	169	118	33	18
PUP	77	172	120	32	20
JCK	50	252	176	40	36
ARM	77	386	270	76	40
CUP	25	115	80	23	12

A lot of the captured data had to be thrown away, because of the acquisition troubles.

To be able to evaluate the accuracy of the classification module, I collected 2 hours of random movement that does not contain any of the exercise motion. This data is separated into multiple (170) data samples for practical reasons.

7.2 Experimental setup

A sportsman is asked to move under supervision of an instructor. The sportsman performs sets of repetitions of exercises, as well as single repetitions. Instructor takes segmentation marks that serve as reference marks to separate repetitions. As a sportsman moves, the person in charge of collecting data presses a button in a measuring application after he does every single repetition (how the app looks like is in Appendix A.1). The measuring app puts all the data in separate "raw" files that are later processed on PC. This postprocessing uses timestamps to glue the data from all sensors appropriatly together.

Because of human error in measurement the segmentation marks are updated to be more precise. More about this update is written in 4.4.3.

Acquisition troubles

The device transmitting data (Section 3.1) was transmitting at frequencies 2420 - 2440 MHz. The transmitting power for these devices is quite low, and if the person turned in such a way that his body shadowed the signal from trasmitter to receiver there was a severe packet loss, up to 40%. Human body contains mostly water and it has a high attenuation of these frequencies [18]. This loss happened for the most important data. A "hack" that I used for this trouble was in introducing two receivers placed on opposite sides of the sportsman. I took both of their data and merged it together. The packet loss was reduced to 2%, which is acceptable. Figure 7.2 illustrates the setup.

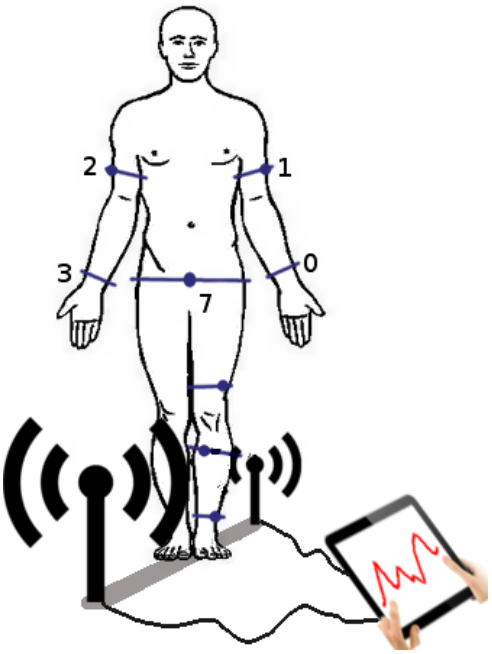


Figure 7.2: Illustration of experimental setup that uses two receivers to combine received data from transmitters.

7.3 Parameters of the model

The recognition model requires handful of parameters, which are combinatorially explored to find the most suitable values for a preselected position of sensors. Evaluating the models across all the combinations of positions and parameters would take too long, so only one position combination is used to find the most appropriate parameters. Models with the same parameters are then trained for all combinations of positions. The selected positions are $X = (1, 5)$ (left wrist and right ankle), because they are the largest extensions of the body and will provide good data across all the exercises (contrary to selecting wrist and bicep on the same arm, for example). This parameter training was done on a validation set which was taken apart from the training set as 15% of the data. Cross validation would be too computationally expensive in this case so it wasn't used.

Codebook

The training of recognition model is done globally, not for each individual. There was not enough data to do this for each person individually. In work of gesture recognition [32] two types of codebooks have been investigated: global for all gestures and individual one. The individual codebooks had higher accuracy. A codebook of 128 prototype vectors and 12 input axis takes 6kB while available SD card space is 8GB. Since storing a different codebook for each exercise is not problematic a global codebook is not investigated.

The chosen sizes are "nice" numbers - one of $M = \{64, 128, 256\}$. The larger the codebook is, the longer it takes for finding the prototype vectors and encoding the observations, so a smaller codebook is preferred.

Input data

Another parameter for the codebook creation is what input data should be used. Is it enough to use acceleration, or do we need gyroscope data as well? The possibilites are $I = \{\texttt{acc}, \texttt{gyr}, \texttt{accgyr}\}$.

HMM structure

Section 4.3.3 shows two structures $S = \{\texttt{simple}, \texttt{composed}\}$ that can be used for segmentation. These two types of models will be evaluated with different sizes of hidden states $N = \{8, 16, 20\}$. Higher number of states is not appropriate, because linear HMM requires that the number of time steps of one repetition is at least equal to the number of the hidden states, which would violate the minimum time of exercise constraint (Section 2.3). Also, higher number of states creates a more complex model, which generally

tends to approximate the training data better while worsening generalization for the test data. (Think of regresssion problem where we fit data that does not comply with IID[1] condition of PAC learning with higher order polynomial instead of using a polynomial of lower degree, which will be less biased than the more complex model).

Viterbi vs alternative Viterbi

Alternative Viterbi (Section 4.3.5) brings imprecision of finding hidden states. The performance of the models is evaluated using both of these algorithms as $D = \{\texttt{viterbi}, \texttt{alternative}\}$.

Classification parameters

The segmented candidates acquired from the most appropriate HMM model with their annotations (based on evaluation of error from Equation 4.51) are used to train the candidate classifier. Calculated features are separated using MLP with one hidden layer.

7.4 Results

The recognition system can be basically divided into two components for which results should be examined. One is the segmentation module based on HMM, and the other is the classification module. The most interesting result of HMM is the false negative rate (miss rate)

$$\texttt{FNR} = \frac{\texttt{false negative}}{\texttt{\# of positive samples}}. \tag{7.1}$$

This is because once HMM misclassifies repetition as free motion, the next-stage classifier cannot reverse this "damage". In the next section best model parameters are selected as minimization of FNR.

The best placement on the body is examined using the results of the entire recognition model using false omission rate, because we want to minimize missing a repetition.

$$\texttt{FOR} = \frac{\texttt{false negative}}{\texttt{\# of negative test outcomes}}. \tag{7.2}$$

The results are in the form of tensor of size $|E| \times |M| \times |N| \times |D| \times |I| \times |H| = 7 \times 3 \times 3 \times 2 \times 3 \times 2$ and 756 values, which is difficult to outline in a table, so following extractions are presented. The whole results can be viewed on attached CD.

[1] IID - Independent and identically distributed random variables

7.4.1 Best model parameters

At first, the parameters that minimize the FNR for each exercise are selected:

Exercise	FNR	M	N	D	I	H
BUR	0.06	256	8	viterbi	accgyr	composed
SIT	0.02	128	20	viterbi	accgyr	single
SQT	0.01	128	8	viterbi	accgyr	single
PUP	0.00	128	16	viterbi	accgyr	single
JCK	0.00	64	8	viterbi	accgyr	single
ARM	0.00	64	16	alternative	accgyr	single
CUP	0.06	128	16	viterbi	gyr	composed

It can be seen that except for uppercuts (`CUP`) all other exercises use codebook derived from `accgyr`, which is a good sanity check. Single model performs better for simpler exercises, and the composed model for more complicated ones (burppes, uppercuts). For implementation in embedded device only `alternative` Viterbi can be used. Fixating these fields yields following results:

Exercise	FNR	M	N	D	I	H
BUR	0.15	256	8	alternative	accgyr	single
SIT	0.05	256	16	alternative	accgyr	single
SQT	0.01	128	16	alternative	accgyr	single
PUP	0.02	64	16	alternative	accgyr	single
JCK	0.01	64	20	alternative	accgyr	single
ARM	0.00	64	16	alternative	accgyr	single
CUP	0.13	128	20	alternative	accgyr	single

A setting that minimizes average FNR error across the parameters M and N can be found across all the combinations:

		N		
		8	**16**	**20**
M	**64**	0.0710	0.0353	0.0383
	128	0.0567	0.0264	0.0275
	256	0.0619	0.0370	0.0346

Table 7.1: Average FNR values for all exercises given N and M.

That setting is M=128, N=16, with these FNR results across all the exercises:

Exercise	FNR	M	N	D	I	H
BUR	0.23	128	16	alternative	accgyr	single
SIT	0.10	128	16	alternative	accgyr	single
SQT	0.01	128	16	alternative	accgyr	single
PUP	0.03	128	16	alternative	accgyr	single
JCK	0.03	128	16	alternative	accgyr	single
ARM	0.03	128	16	alternative	accgyr	single
CUP	0.18	128	16	alternative	accgyr	single

It should be noted that it is possible to use different models for each exercise - the implementation in embedded device could switch between the parameters. The purpose of this parameter selection is to have a single setting that can be used to find the best placement of sensors.

There is also an underlying problem of overfitting - the movements BUR and CUP are much more complex than the others, and more complex model with a higher number of parameters will tend to fit it better. The testing data is however biased – it comes from the same people that it has been trained on. Better models could be accomplished only by having a larger amount of training data to match the model complexity.

7.4.2 Placement on the body

Using settings discovered in previous section:
$M = 128, N = 16, D = alternative, I = accgyr, H = single$
I ran evaluations for each combination of 1, 2, 3 or 8 bracelets (together 93 different combinations).

In the table below I present the results based on each exercise - along with sanity check of using 8 bracelets. To see where each sensor is located you can look again at Figure 2.6.

Exercise	1 bracelet	2 bracelets	3 bracelets	3 bracelets and 8 bracelets
BUR	0.155 — 0	0.113 — 0x1	0.093 — 0x2x4	0.093 — 0x2x4
SIT	0.094 — 1	0.039 — 3x4	0.023 — 0x2x7	0.023 — 0x2x7
SQT	0.035 — 6	0.007 — 0x6	0.007 — 5x6x7	0.007 — 0x5x6
PUP	0.085 — 1	0.008 — 2x7	0.000 — 1x2x3	0.000 — 0x1x2x3x4x5x6x7
JCK	0.018 — 4	0.005 — 0x6	0.000 — 0x1x7	0.000 — 0x1x7
ARM	0.012 — 3	0.003 — 1x3	0.000 — 0x2x3	0.000 — 0x2x3
CUP	0.148 — 6	0.111 — 0x7	0.120 — 0x1x6	0.111 — 0x1x6

The results for 1,2,3 bracelets seem in general reasonable - they are in placements where

the largest range of motion is exerted:

- `BUR`: Except for combination 1x2, they are placed in arms and legs
- `SIT`: Maybe a bit surprising is 4x5, because the legs don't move that much in sit-ups
- `SQT`: The tighs move the most, which works well
- `PUP`: Recognition at biceps is better than at wrists, and the best result comes from biceps and wrist.
- `JCK`: Again uses mostly the largest extensions.
- `ARM`: Arm raises uses only arms, not legs - sanity check.
- `CUP`: Again uses mostly the largest extensions.

Maybe a bit disappointingly using all of the sensors doesn't help very much - the underlying reason is probably that it is difficult to do vector quantization effectively in such a large dimensionality - 8 sensors provide 48-D data. This might be also the reason why `CUP 0x1x6` has larger FN than `CUP 0x7`, a strange rise of error while having more data to learn from.

Taking median value over all exercises given number of bracelets yields following results for placing the bracelets:

```
best combination for 1 bracelet:      2, FN=0.09
best combination for 2 bracelets:   1x5, FN=0.03
best combination for 3 bracelets: 1x2x7, FN=0.02
```

7.4.3 Accuracy of counting repetitions

The final stage of the algorithm, after the segmentation, is classification. The evaluation is done on the set of data containing repetitions of given exercise and on free motion data. Using features from Section 5.5, I've come to the following results for each best combination of exercises. Since classification is done using MLP, it is important to use validation set to avoid overfitting. The high accuracy results are due to the fact that the data is almost linearly separable because of the use of similarity with exercise template. The accuracy bottleneck lies in proper segmentation.

Exercise	**1 bracelet**		**2 bracelets**		**3 bracelets**	
	Validation	Testing	Validation	Testing	Validation	Testing
BUR	0.78	0.72	0.83	0.75	0.90	0.82
SIT	0.85	0.82	0.95	0.85	0.97	0.91
SQT	0.96	0.95	0.99	0.92	1.00	0.95
PUP	0.92	0.90	0.99	0.96	1.00	0.92
JCK	0.98	0.95	0.99	0.95	1.00	0.94
ARM	0.98	0.95	0.99	0.93	1.00	0.99
CUP	0.80	0.75	0.85	0.81	0.88	0.85

Table 7.2: Accuracy results for given number of bracelets.

Chapter 8

Conclusions

In this thesis I have succesfully applied machine learning algorithms to the task of counting exercise repetitions from body worn sensors. The input data comes from 8 sensors placed on different locations (Figure 2.6) that measure acceleration and angular velocity with 50Hz frequency. The main algorithm modules are based on discrete hidden markov models (for segmentation) and dynamic time warping and multi layer perceptrons (for classification of repetition). I found the best placement for the sensors based on their numbers to be located at:

- 1 sensor: left bicep,
- 2 sensors: left wrist and ankle,
- 3 sensors: left wrist, left bicep and right thigh.

I have managed to create models that can be run efficiently in real time on embedded devices and I made a simple implementation that can run on Arduino Due prototype.

The accuracy of the recognition is exercise dependent and bracelet-dependent and is summarized in Table 7.2. The more bracelets are used, the higher is the accuracy, and it moves in interval of 78% to 98%. The more complicated the motion is, the lower is the accuracy. More complicated motion exhibits more ways of how it can be performed, which is a property that left-to-right hidden markov models have difficulty to capture.

I have not compared my method to other works, because of lack of experimental dataset. Because of this I make the dataset I have used to be publicly available at `http://michal.sustr.sk/exercise/` along with this thesis.

Another approach for recognition can use end-to-end neural networks architecture based on LSTM [13] which is the basis for many state of the art methods for speech recognition [12] or more complex tasks such as image captioning [33]. However this approach requires that the training set is much larger, and the training .

An obvious extension of this work could be in recognizing the movement itself, not just counting repetitions. This might however be quite difficult to implement in embedded device with low power consumption, such I had at my disposition. I had to deal several times with optimizing the memory usage, especially with buffers, and the recognition is quite on the edge of the capabilities of my hardware.

Bibliography

[1] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat, *Np-hardness of euclidean sum-of-squares clustering*, Machine learning **75** (2009), no. 2, 245–248.

[2] David Arthur and Sergei Vassilvitskii, *k-means++: The advantages of careful seeding*, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[3] Media Anugerah Ayu, Siti Aisyah Ismail, Ahmad Faridi Abdul Matin, and Teddy Mantoro, *A comparison study of classifier algorithms for mobile-phone's accelerometer based activity recognition*, Procedia Engineering **41** (2012), 224–229.

[4] Ling Bao and Stephen S Intille, *Activity recognition from user-annotated acceleration data*, Pervasive computing, Springer, 2004, pp. 1–17.

[5] Tomas Brezmes, Juan-Luis Gorricho, and Josep Cotrina, *Activity recognition from accelerometer data on a mobile phone*, Distributed computing, artificial intelligence, bioinformatics, soft computing, and ambient assisted living, Springer, 2009, pp. 796–799.

[6] X. Chu, W. Ouyang, H. Li, and X. Wang, *Structured Feature Learning for Pose Estimation*, ArXiv e-prints (2016).

[7] John Duchi, Elad Hazan, and Yoram Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, The Journal of Machine Learning Research **12** (2011), 2121–2159.

[8] Mahmoud Elmezain, Ayoub Al-Hamadi, Jörg Appenrodt, and Bernd Michaelis, *A hidden markov model-based continuous gesture recognition system for hand motion trajectory*, Pattern Recognition, 2008. ICPR 2008. 19th International Conference on, IEEE, 2008, pp. 1–4.

[9] Gernot A Fink, *Markov models for pattern recognition: from theory to applications*, Springer Science & Business Media, 2014.

[10] Norbert Győrbíró, Ákos Fábián, and Gergely Hományi, *An activity recognition system for mobile phones*, Mobile Networks and Applications **14** (2009), no. 1, 82–91.

[11] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin, *Neural networks and learning machines*, vol. 3, Pearson Education Upper Saddle River, 2009.

[12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al., *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*, Signal Processing Magazine, IEEE **29** (2012), no. 6, 82–97.

[13] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[14] Kurt Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural networks **4** (1991), no. 2, 251–257.

[15] Xuedong D Huang, Yasuo Ariki, and Mervyn A Jack, *Hidden markov models for speech recognition*, vol. 2004, Edinburgh university press Edinburgh, 1990.

[16] Daniel Huson, *Algorithms in bioinformatics i*, 2008.

[17] Eamonn Keogh and Jessica Lin, *Clustering of time-series subsequences is meaningless: implications for previous and future research*, Knowledge and information systems **8** (2005), no. 2, 154–177.

[18] Jaime Lloret, Sandra Sendra, Miguel Ardid, and Joel JPC Rodrigues, *Underwater wireless sensor communications in the 2.4 ghz ism frequency band*, Sensors **12** (2012), no. 4, 4237–4264.

[19] Stuart P Lloyd, *Least squares quantization in pcm*, Information Theory, IEEE Transactions on **28** (1982), no. 2, 129–137.

[20] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan, *The planar k-means problem is np-hard*, WALCOM: Algorithms and Computation, Springer, 2009, pp. 274–285.

[21] Abdelkrim Nemra, Luis M. Bergasa, Elena López, Rafael Barea, Alejandro Gómez, and Álvaro Saltos, *Robot 2015: Second iberian robotics conference: Advances in robotics, volume 1*, ch. Robust Visual Simultaneous Localization and Mapping for MAV Using Smooth Variable Structure Filter, pp. 557–569, Springer International Publishing, Cham, 2016.

[22] Mikael Nilsson and Marcus Ejnarsson, *Speech recognition using hidden markov model*, Department of Telecommunications and Speech Processing, Blekinge Institute of Technology (2002).

[23] Hee-Seon Park and Seong-Whan Lee, *Off-line recognition of large-set handwritten characters with multiple hidden markov models*, Pattern Recognition **29** (1996), no. 2, 231–244.

[24] Igor Pernek, KarinAnna Hummel, and Peter Kokol, *Exercise repetition detection for resistance training based on smartphones*, Personal and Ubiquitous Computing **17** (2013), no. 4, 771–782 (English).

[25] Zoltán Prekopcsák, *Accelerometer based real-time gesture recognition*, (2008).

[26] Lawrence R Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Proceedings of the IEEE **77** (1989), no. 2, 257–286.

[27] Frank Rosenblatt, *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*, Tech. report, DTIC Document, 1961.

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning representations by back-propagating errors*, Cognitive modeling **5** (1988), no. 3, 1.

[29] Lin Sun, Daqing Zhang, Bin Li, Bin Guo, and Shijian Li, *Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations*, Ubiquitous intelligence and computing, Springer, 2010, pp. 548–562.

[30] Emmanuel Munguia Tapia, Stephen S Intille, William Haskell, Kent Larson, Julie Wright, Abby King, and Robert Friedman, *Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor*, Wearable Computers, 2007 11th IEEE International Symposium on, IEEE, 2007, pp. 37–40.

[31] Tijmen Tieleman and Geoffrey Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning **4** (2012), 2.

[32] Tomas Tunys, *Gestures detection and nfc for android os.*

[33] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio, *Show, attend and tell: Neural image caption generation with visual attention*, arXiv preprint arXiv:1502.03044 (2015).

[34] Liyang Zhu, Pei Zhou, Anle Pan, Jian Guo, Wei Sun, Xiaohe Chen, Zhen Liu, and Lirong Wang, *A survey of fall detection algorithm for elderly health monitoring*, Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on, IEEE, 2015, pp. 270–274.

Appendix A

Software

A.1 Collecting data program

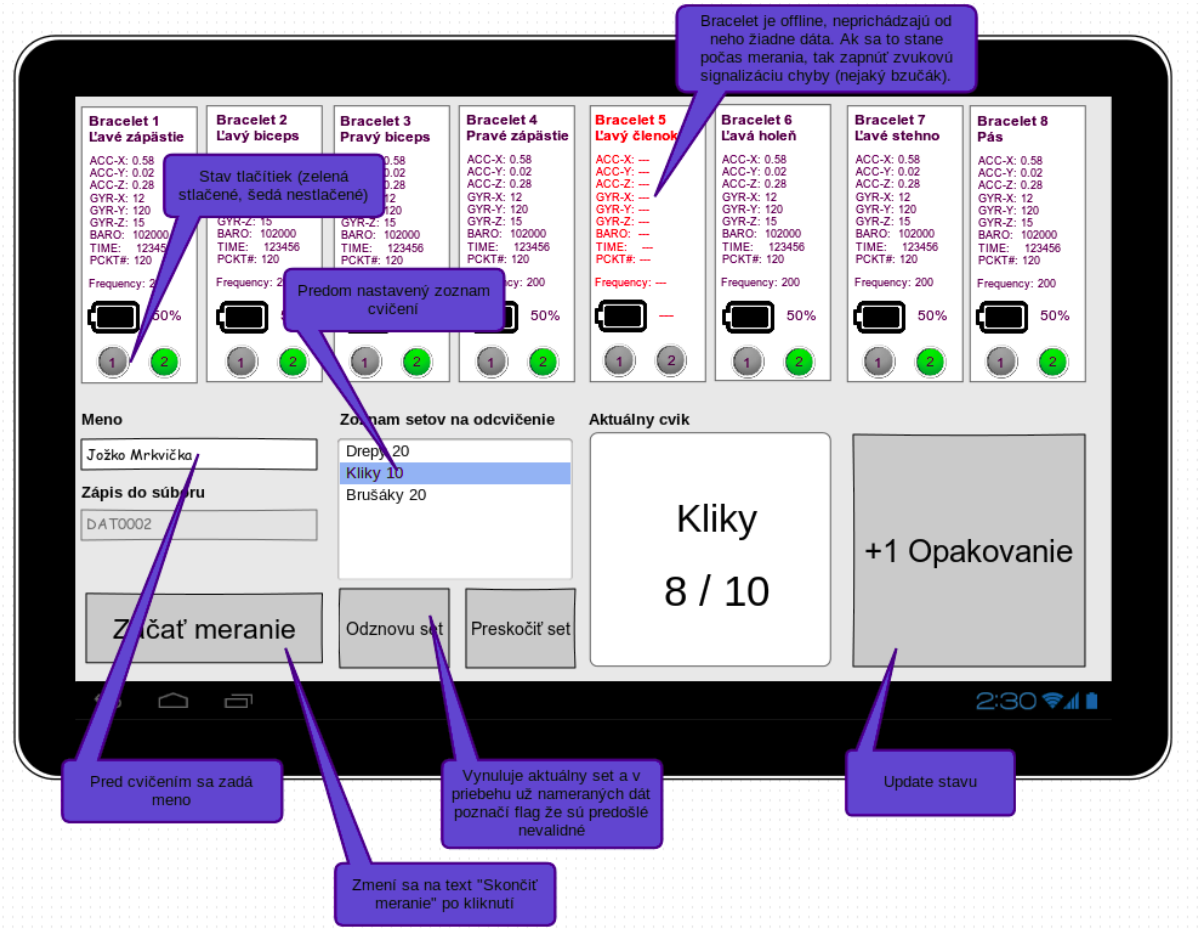


Figure A.1: Datalogger application with labels written in Slovak.

Appendix B

Visualisation of exercises (acceleration values)

Only acceleration values are shown. It would take too much space with gyroscopic data. RGB plots correspond to x,y,z coordinates. All values are normalized into $\langle 0; 1 \rangle$ range to demonstrate the shape of the acceleration data.

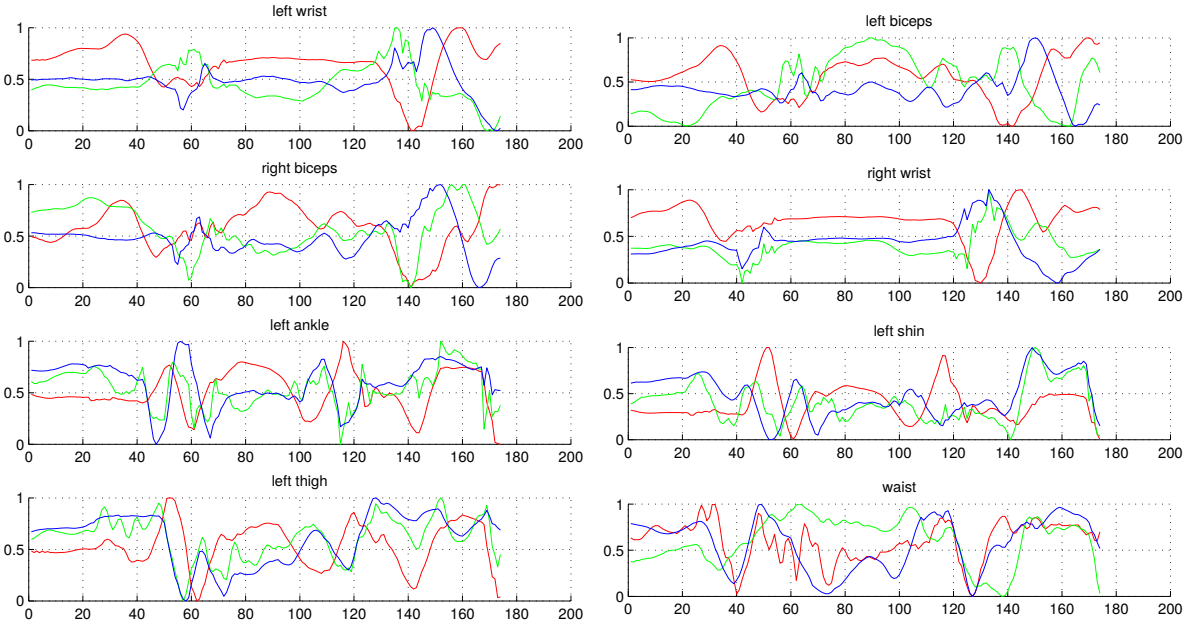


Figure B.1: Burpees

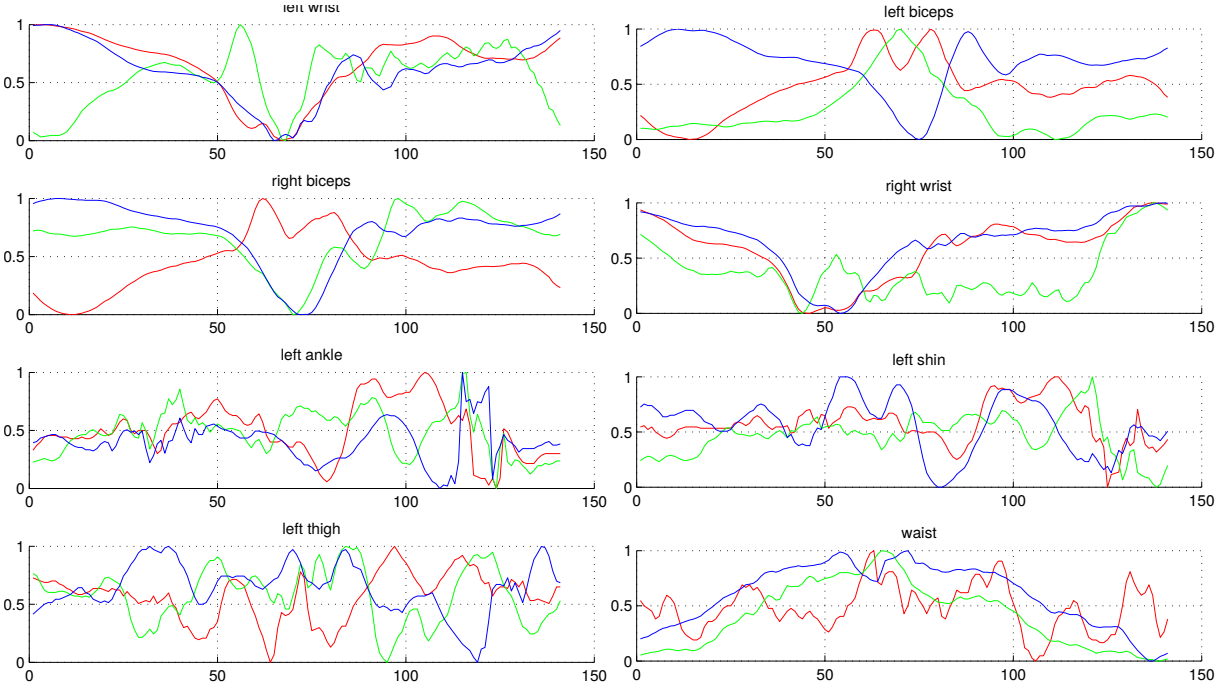


Figure B.2: Sit-ups

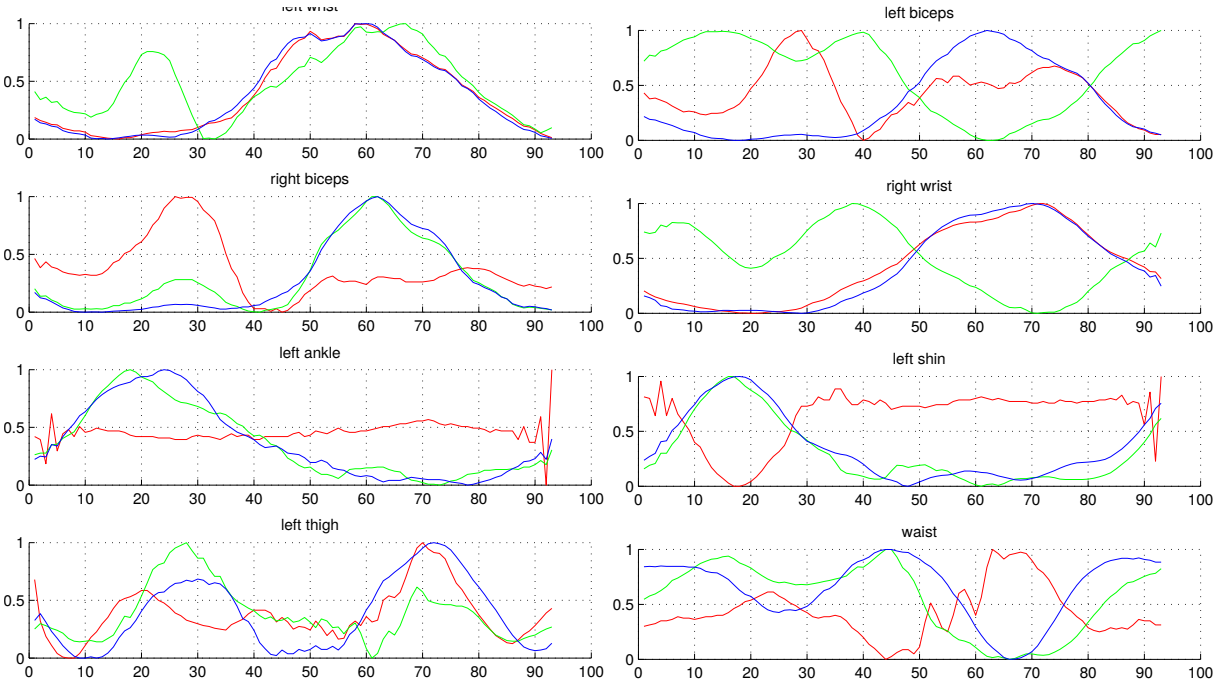


Figure B.3: Squats

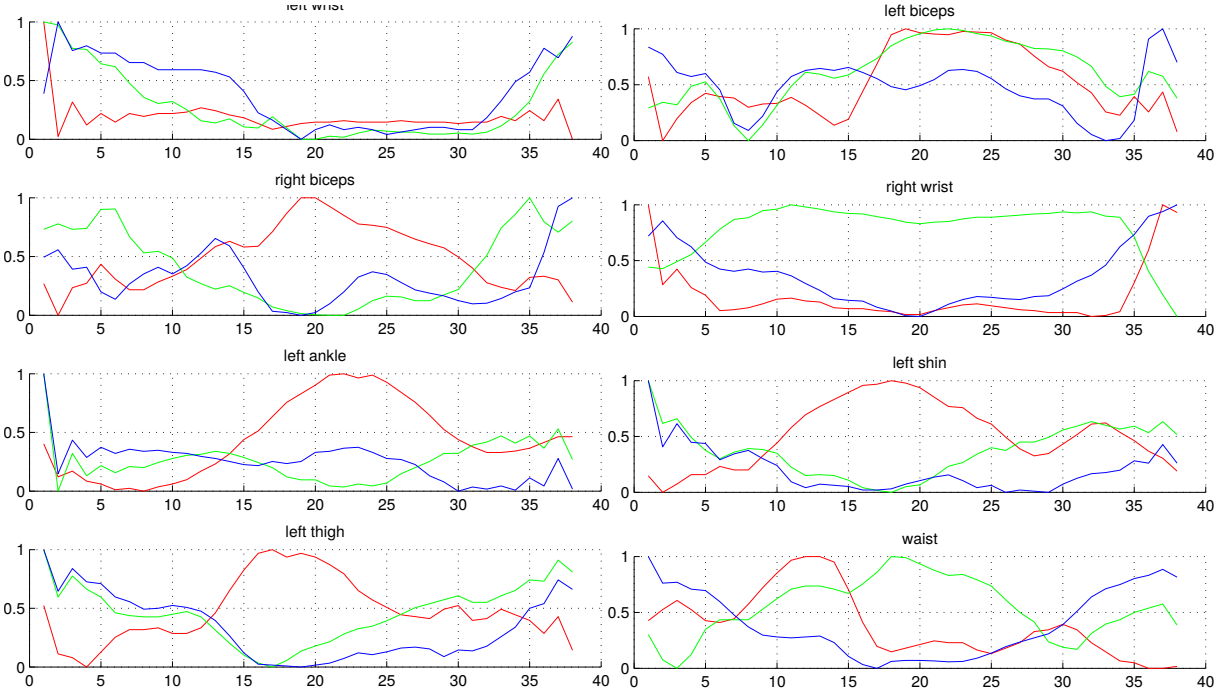


Figure B.4: Pushups

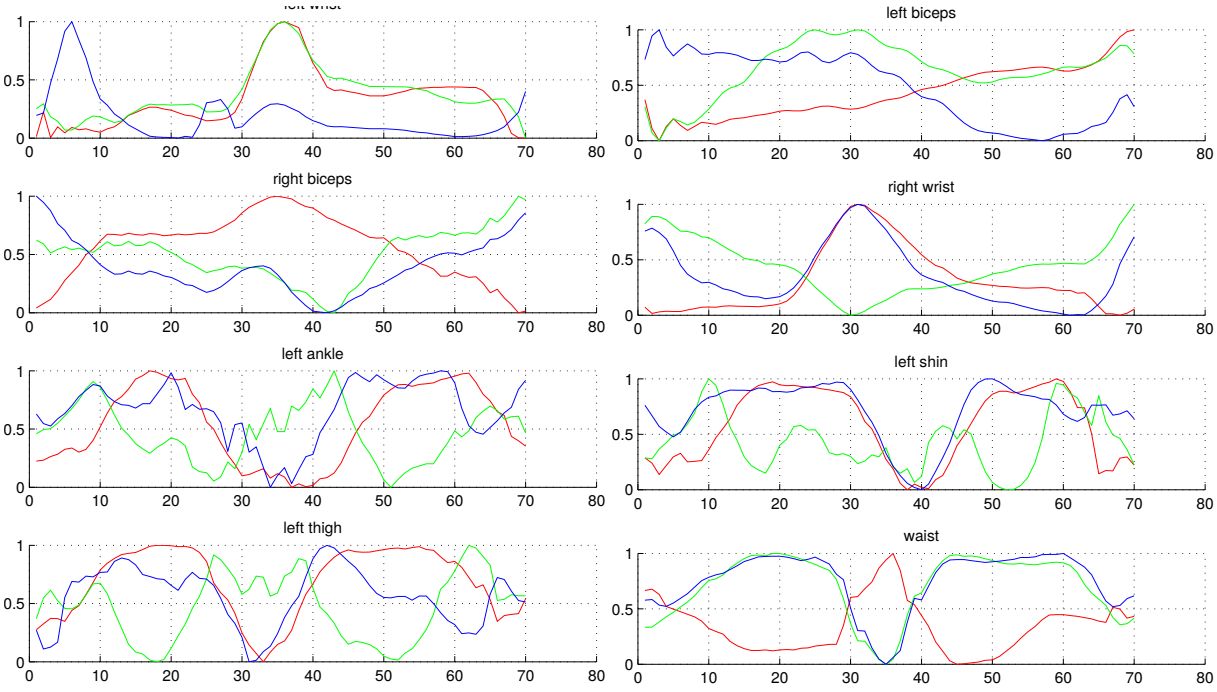


Figure B.5: Jumping jacks

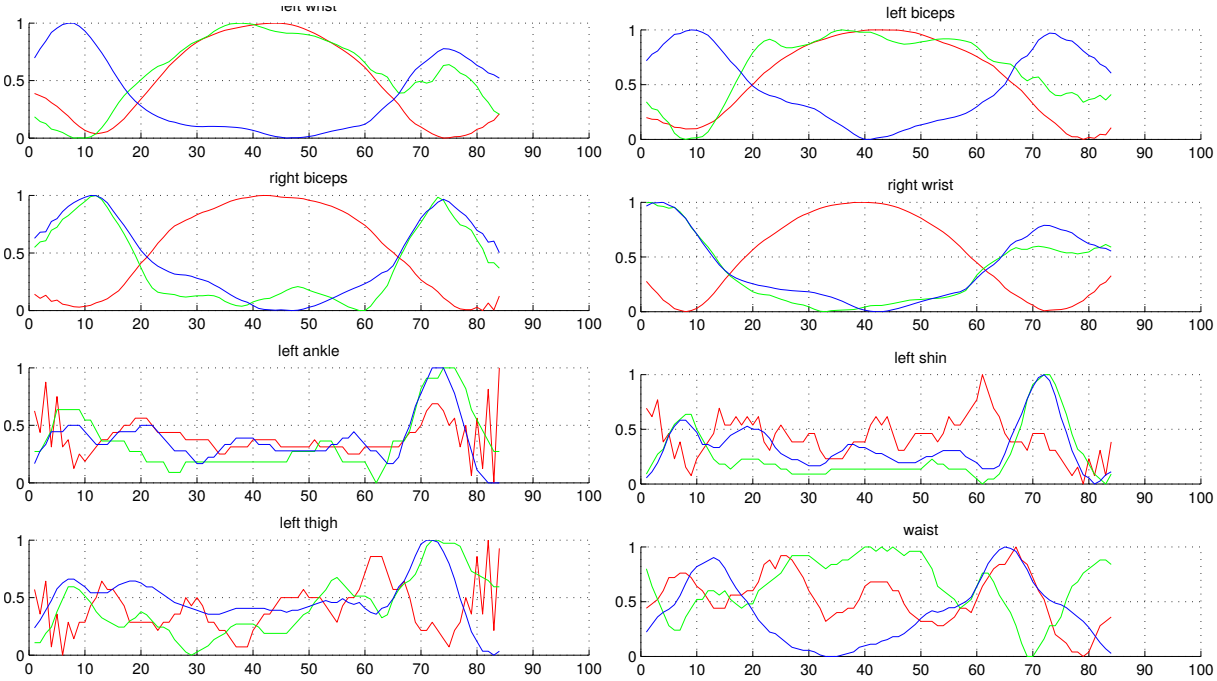


Figure B.6: Arm raises

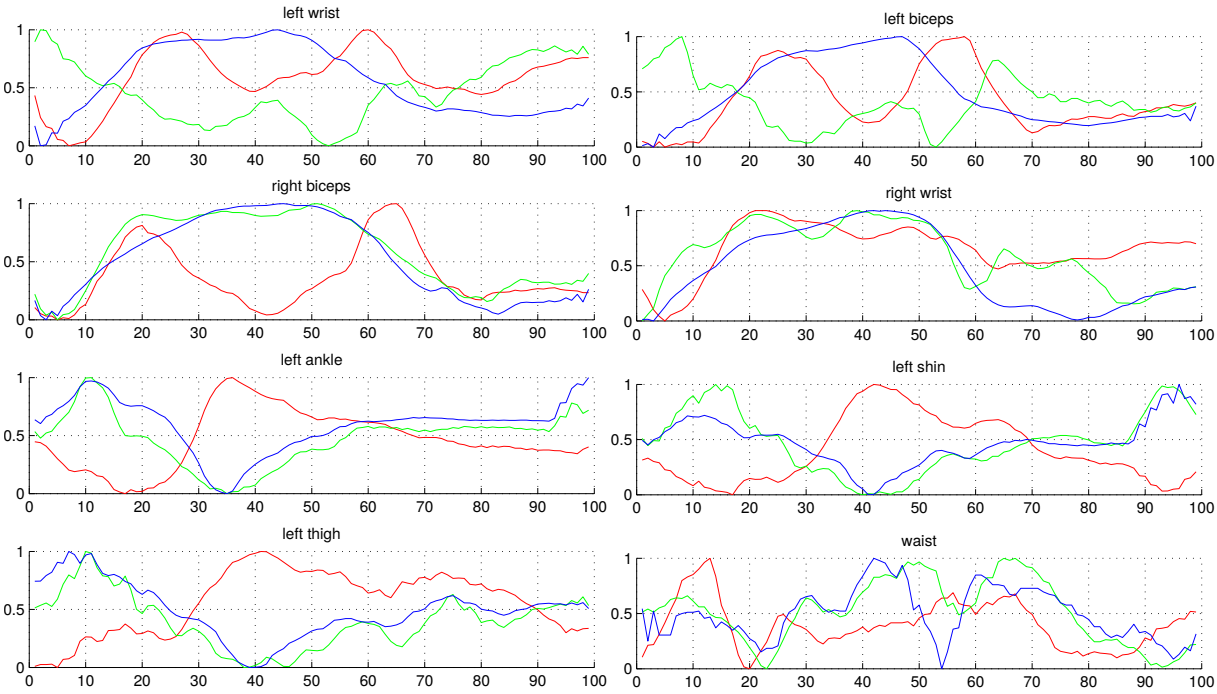


Figure B.7: Uppercuts

Appendix C

Learning curves

HMM EM iteration learning curves for placement 1x5 (wrist and ankle)

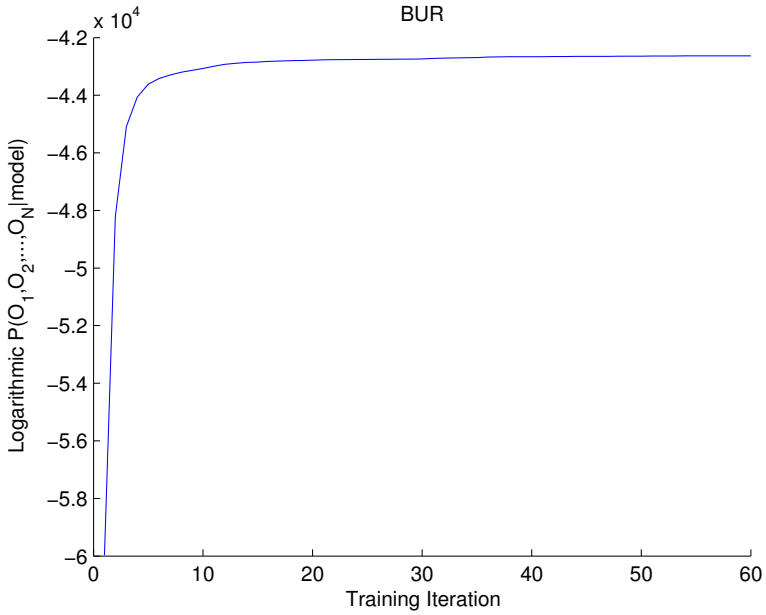


Figure C.1: Burpees

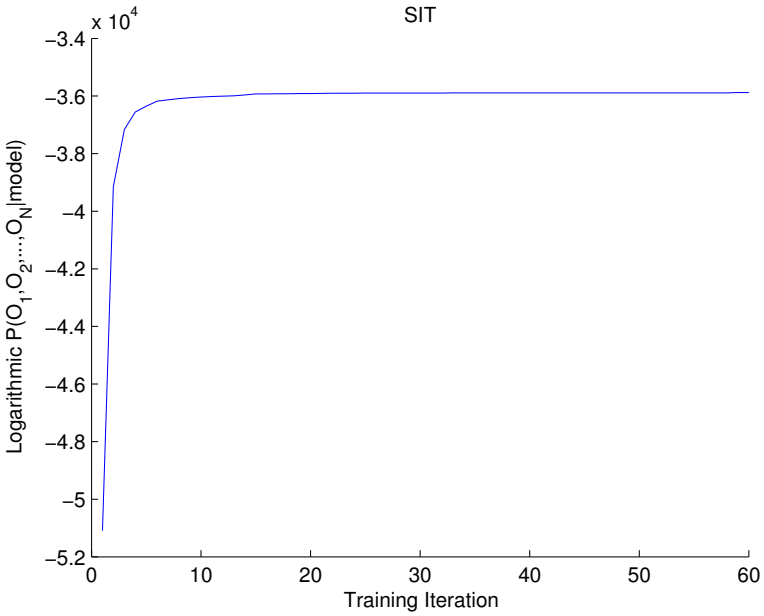


Figure C.2: Sit-ups

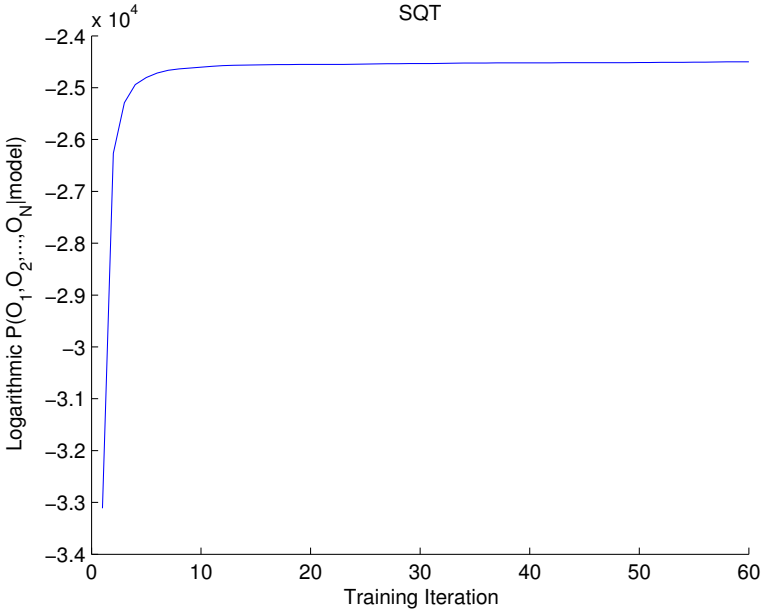


Figure C.3: Squats

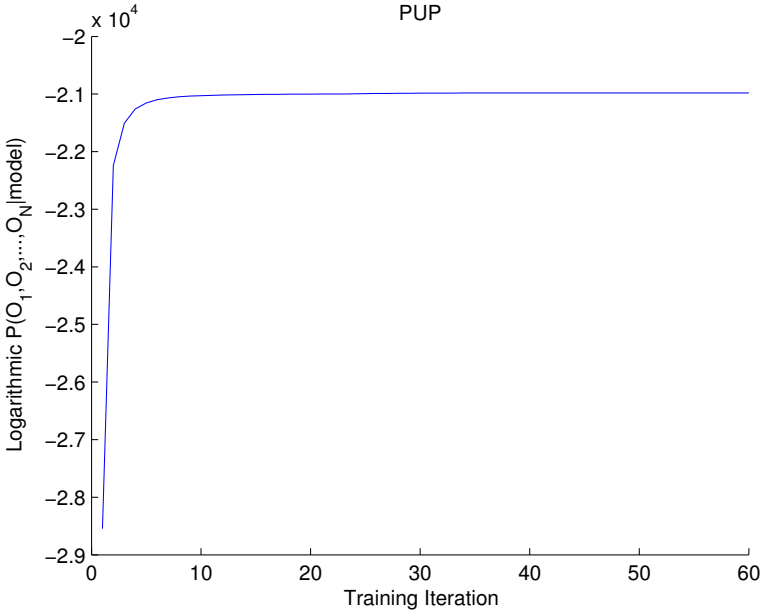


Figure C.4: Pushups

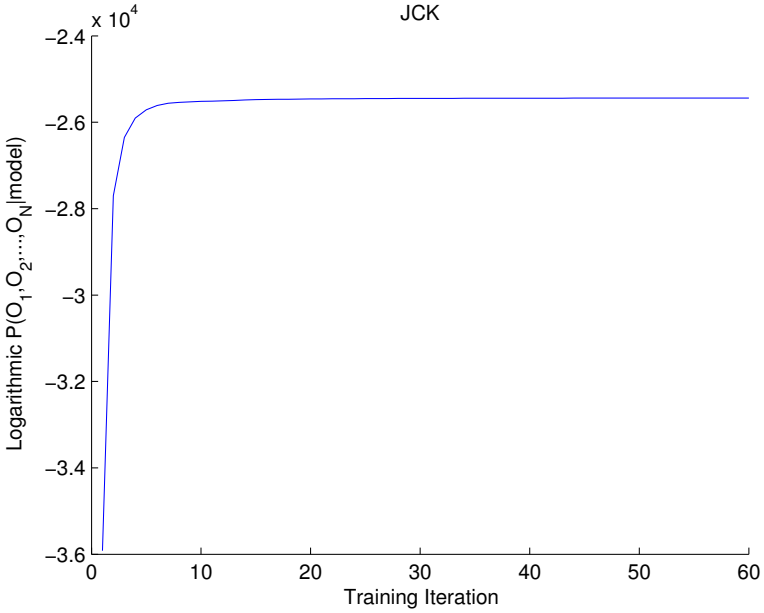


Figure C.5: Jumping jacks

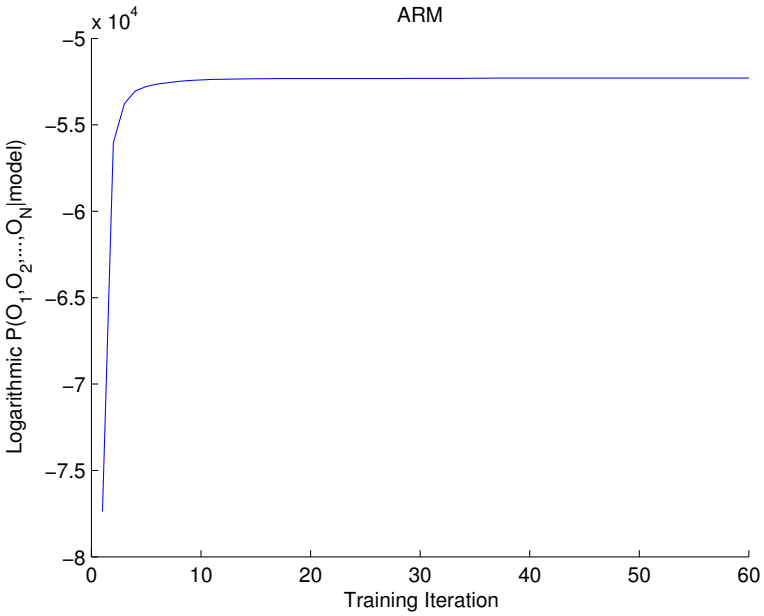


Figure C.6: Arm raises

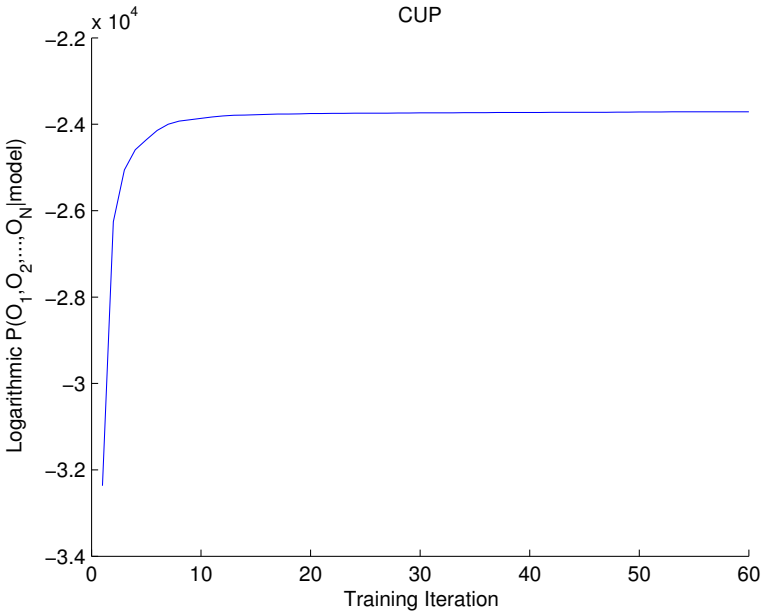


Figure C.7: Uppercuts