Università degli Studi di Padova

Department of Information Engineering

Master Thesis in Telecommunication Engineering

Extraction of Gait Patterns from Smartphone-acquired Inertial and Video Signals

Supervisor
Michele Rossi
Università di Padova

Co-supervisor
Riccardo Bonetto
Università di Padova

Master Candidate
Alberto Lanaro

21 February 2017

Academic Year 2016/2017

Abstract

In recent years, smartphones and wearable devices are the fastest growing technologies that, together with advances in Wireless Sensor Networks (WSN), will enable ubiquitous sensing. These devices are suitable for indoor and outdoor scenarios, can be worn on many parts of the human body and integrate a large number of sensors able to gather physiological and behavioural biometric information.

In this work, I am concerned with the design and implementation of a gait analysis system able to extract meaningful gait patterns and parameters from the biometric characteristics of a user's walking style. A smartphone located in ad-hoc made chest support is utilized to gather inertial data and video signals from the built-in sensors and rear-facing camera. After a data and video processing phase, the performance of the Growing When Required (GWR) unsupervised clustering algorithm is investigated using different combinations of the input signals. Final results show how the synthesis of signals extracted from the reference video and accelerometer data is able to detect all the expected input data clusters in all the analyzed scenarios.

Sommario

Negli ultimi anni, la tecnologia legata al mondo degli smartphone e dei dispositivi indossabili é sicuramente tra quelle con tasso di crescita maggiore. Essa, assieme agli sviluppi in ambito delle reti di sensori (WSN), ha inoltre permesso la realizzazione di innovativi scenari di ubiquitous sensing. Questi dispositivi possono essere impiegati sia in situazioni indoor che outdoor, possono essere indossati in diverse parti del corpo e integrano un gran numero di sensori che permettono di ricavare informazioni di varia natura sull'utente analizzato.

Questa tesi é dedicata alla progettazione e all'implementazione di un sistema di analisi della camminata in grado di estrarre parametri e modelli caratterizzanti un dato individuo. Durante la fase di acquisizione, i dati provenienti dai sensori integrati di uno smartphone posizionato al livello del torace sono sono affiancati da un video di riferimento acquisito dalla videocamera posteriore. Dopo le successive fasi di processazione dei dati inerziali e del video, sono testate le performance dell'algoritmo di clustering non supervisionato GWR utilizzando diverse combinazioni dei segnali di input. I risultati finali mostrano come, accostando i dati accelerometrici ai segnali estratti dal video di riferimento, in tutti gli scenari analizzati, il sistema proposto riesca a rilevare, in maniera ottimale, i cluster previsti per i segnali di input.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ARBF Augmented Radial Basis Function Neural Network

BIC Bayesian Information Criterion

CCD Charge-Coupled Device

CoM Center of Mass

CPU Central Processing Unit

CWT Continuous Wavelet Transform

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DoG Difference of Gaussian

ECF Explicit Complementary Filter

EER Equal Error Rate

EM Expectation-Maximization

EMG Electromyography

FC Final Contact

FIR Finite Impulse Respose

FTP File Transfer Protocol

GMM Gaussian Mixture Model

GNG Growing Neural Gas

GWR Growing When Required

HAR Human Activity Recognition

IC Initial Contact

IMU Inertial Measurement Unit

KLT Kanade, Lucas and Tomasi feature tracker

KNN K-Nearest Neighbours

LoG Laplacian of Gaussian

MEMS Micro Electro-Mechanical Systems

ML Machine Learning

OS Operating System

PCA Principal Component Analysis

PPM Perspective Projection Matrix

PSD Power Spectral Density

RCE Rapid Cluster Estimation

SIFT Scale-Invariant Feature Transform

SVD Singular Value Decomposition

SVM Support Vector Machine

1 Introduction

The development of smartphones equipped with a set of different and powerful sensors has enabled new techniques that allow to better understand the relationship between physical activity and health condition. Moreover, smartphones sensors can be also exploited to develop new frameworks that can be utilized to recognize a target from its walking style [3], [4], or to detect anomalies in gait patterns [5]. Generally, we can find three main application fields: Human Activity Recognition (HAR), medical and healthcare, identification and authentication.

1.1 Human Activity Recognition

Monitoring daily activities can help in recognizing the health and wellness of different users and can induce behavioural changes for an healthier and more active lifestyle. HAR has attracted increasing attention in the field disease prediction. For example, when abnormal actions of elders or postoperative patients are detected, nursing staff or family members may receive a warning message so that sick patients can receive medical attention. Besides, old people might have diseases such as Alzheimer, Parkinson or epilepsy if their action often deviate from normal habits and activities.

In HAR, different types of sensing technologies have been exploited to improve the recognition rate and adapt to different application scenarios. They can be di-

vided in three main categories: vision-based approaches, environment interactive sensor based approaches and wearable sensor based approaches. Vision-based techniques mainly exploits cameras or videos to monitor and recognize different activities. Although this method can reach high recognition rate, the use of camera or video is not very practical in many indoor environments, especially for privacy issues. Another issue of vision-based approaches that greatly limits their practical use, is that they often suffer from illumination variation, ambient occlusion and background change.

For environmental interactive sensor-based approaches, they try to recognize human activities by analysing the interaction between objects and the user. This is done assuming that there exists a relation between objects and activities. For example, if a sensor that is embedded in a chair is triggered, we can deduce that the user is sitting on it. With this approach activities like eating, washing, sleeping can be recognized. However, this usually comes at a high cost and is often only limited to indoor scenarios. Other issues include how to efficiently deploy the sensors without causing much inconvenience to the users and how to set up and maintain the system in order to make everything work normally.

For the sensor based approach, the small size and the flexibility of the sensors make it possible for a user to wear or carry mobile devices during his daily routine without any problems. These devices are suitable both for indoor and outdoor scenarios, and can be worn on many parts of the human body like legs, ankles, arms and wrist. Moreover, a user can take more than one mobile device at the same time in order to increase the detection rate of the HAR system [6]. As a consequence, more recent studies focused on exploring the potential of wearable devices for activity recognition in a pervasive and ubiquitous way. Among all the wearable devices, smartphones have the advantage that they do not force the user to wear additional sensing components and therefore the interest on smartphone-based HAR is drastically increasing. Also, modern smartphones have powerful computing and communication capabilities that allow us to process data in real time and efficiently interact with remote servers. Moreover, smartphones contain several sensors such as gyroscope, accelerometer, GPS, camera, microphone, compass, etc. that can be exploited to collect important signals and data that provide an alternative and economic way for activity recognition. Besides using their powerful sensing functionalities, smartphones can be used as platforms for

persuasive applications to motivate healthier behaviour because of their ubiquitous presence. In the next section, I briefly recall the available sensors on the mobile phones that can be used for activity recognition purposes and next I focus on the main activities that can be recognized using them.

1.1.1 Sensors

Although today's mobile phones are very powerful devices and their computing power and functionalities are increasing, from the user's point of view they still as communication devices. One possible big step could be to use them as active assistant devices that support users' daily activities, and this could be realized thanks to the embedded sensors on mobile phones.

Figure 1.1 shows the main sensors available on current smartphones: the standard sensors such as cellular radio, WiFi, Bluetooth, microphone, camera and GPS, and the newer ones like accelerometer, gyroscope, digital compass, the light sensor and the proximity sensors. In the near future, it is expected that the number of sensors integrated in the smartphones will grow to support new applications. Sensors providing health information like heart beat rate or blood pressure are not very useful if embedded on a smartphone since they require contact with the skin of the user. However, there already exist wrist/chest bands and smartwatches that are able to collect this information and that are directly connected via Bluetooth to the smartphone.

Figure 1.1: Principal sensors available on smartphones.

One fundamental sensor on a smartphone, the *radio for cellular communication*, is used in ubiquitous applications for context recognition. By using the connection information between the radio and the cell tower, it is possible to locate the user, for example, it is possible to determine when he/she is at home. Even though this only provides a high level activity information, it gives a clue of what the user may be doing. Moreover, signal fluctuations between the radio and the tower may be used to assess if the user is walking, driving or if he is in a stationary state.

Besides the radio for cellular communication, *Bluetooth* and *WiFi radios* can be used for context and activity recognition. For example, interactions among different smartphones Bluetooth transceivers can be used to infer social interactions between users. Another application the transceivers can be utilized for is localization. For example, fluctuations in WiFi signals can be used to locate the user.

The *microphone* and the *camera* can also be used for HAR purposes by collecting audio or pictures during user's daily activities, such as being in a conversation or being in a noisy environment. *GPS* is one other sensor used mainly to track the location and the speed of the mobile phone.

The *three-axis accelerometer* is probably the most effective sensor in the HAR with smartphones context. Even though it was integrated in the mobile phone with the objective of enhancing the user experience (by changing the orientation of the screen according to the orientation of the phone), it can be used to recognize activities like walking, running, sitting, climbing stairs, and even falling. Activity recognition using smartphones inertial sensors is receiving a lot of interest since with these type of sensors a very high classification rate can be achieved. Similarly, the *gyroscope* and the *digital compass* can be used for HAR by measuring the orientation of the smartphone and they lead to very good results when combined with the accelerometer.

Proximity and *light* sensors were introduced to enhance the user experience. For example, when the proximity sensor detects that the user is holding the phone close to its face, the keyboard is disabled or similarly the light sensor automatically adjusts the brightness of the screen according to the surrounding environment. For activity recognition purposes, they can be used in combination with other sensors in order to infer more detailed information about users activities.

For instance, the light sensor may provide information about the environment of the user and consequently also prior information about the activities that can be performed in that situation.

1.1.2 Activities and Applications

If we analyze the early works on HAR with smartphones, they only considered high-level activities associated with location information like staying at home or being at the office. However, these results do not tell much about the exact activity performed by the user. In fact, being at the office is not equivalent to working, or staying at home does not tell if the user is watching TV or having lunch. With other sensors like accelerometer or gyroscope more specific activities can be recognized, like sitting or standing. From the microphone, for example, we can detect that a conversation is going on, from the bluetooth sensor we can understand that the user's contacts are around and consequently we can conclude that the user is sitting at a meeting in the office.

Class	Activity types
Locomotion	Walking, running, standing, still, lying.
Mode of transportation	Biking, travelling with a vehicle, riding a bus, driving.
Exercise	Outdoor bicycling, playing soccer, biking on a fitness bike.
Health related activities	Falls, rehabilitation activities, following routines.
Daily activities	Shopping, using a computer, sleeping, going to work, going back home, working, lunch, dinner, breakfast, in a conversation, attending a meeting.
Usage of the phone	Text messaging, making a call, browsing the web, writing an email, using an app.

Table 1.1: Types of activities studied in literature

Location and motion-associated activity recognition are the two main fields of activity recognition using mobile phones. Recent works used smartphones to recognize even more complex activities like, in the sports field: cycling, playing soccer, running, sitting, standing, walking, lying, falling, etc. or, for daily activities: shopping, using a computer, sleeping, going to work, working, having lunch, dinner or breakfast. In Table 1.1 I present the main types of activities studied in the literature classified in six different categories according to their objectives [7]. HAR with smartphones can be used in various application scenarios. They are classified in three main categories [8]: applications for end users (health monitoring, fitness tracking, etc.), application for third parties, applications for crowds and groups (activity-based social networks, place/event detection, etc.). The development of *health-care* applications is mainly due to the strong correlation between the level of physical activity and the level of well-being. Common diseases like obesity are related to a high level of physical inactivity. Nowadays patients are asked to keep a diary about all their principal physical daily activities. However, this practice can only succeed if the patients are very meticulous and precise. On the other hand, an automatic activity recognition system based on mobile sensing can surely offer a more reliable solution.

Activity recognition with smartphones can also help in following daily routines and this can be very useful, especially for elderly users. Deviations from routines and habits can be easily identified and this helps doctors to diagnose particular conditions that would not be observed during routine analysis. In the case of elderly patients, in fact, inconsistencies in their daily routines can be a symptom of dementia or Alzheimer. One of the main challenges in this field, especially when working with elderly, is that these type of patients may experience strong difficulties with the smartphone interface due to their limited experience.

Another field of application of activity recognition with mobile phones is the rehabilitation of diseases. For instance, a HAR system can detect if a user is correctly doing the exercises recommended by a physician or a physiotherapist.

Well-being and *fitness monitoring* are also typical applications in HAR studies. The mobile phone can act as a pedometer and can count the steps and calories burnt during our daily activities. Moreover, using persuasive techniques, it can interact with the users to change and improve their behaviour and lifestyle.

Ambient-assisted living is another field that can benefit from activity recogni-

tion systems. Assistance for people with cognitive disorders or people with chronic condition can be provided and their physical activities and routines can be monitored. Another useful application exploited recently by researchers in this field is the smartphone-based fall detection. When a fall is detected, especially outdoor, these systems provide support with an online location identification using the GPS.

1.1.3 Activity Recognition Process

The activity recognition process can be summarized in collecting sensor readings, processing them and assigning each sensor reading to the appropriate activity (i.e. the process of how to interpret raw sensor data to classify a set of different activities). Most of the studies, not necessarily in the field of mobile sensing, focused on statistical and Machine Learning (ML) tools [9] in order to infer useful information about the activities from the sensor data. The learning phase can be of two different types: *supervised* or *unsupervised*. The first one relies on sample observations (labels) associated with a specific activity used to train a classifier. Unsupervised techniques, on the other hand, do not rely on labeled data. Usually supervised or semi-supervised methods are the two favourite approaches [9], [10], [11].

Supervised methods are composed of two main phases: training and classification or testing. In the training phase a given set of labelled data called "training set" is used to train a given classifier in order to discover patterns from the sensor readings. One of the main issues of this phase is labelling the data in the training set. Either the user itself labels each data according to the corresponding activities, or the performed activities should be recorded with a video camera and then automatically labeled by the system.

Activity Classification Steps

After the sensor data are collected, the main steps of activity recognition, shown in Figure 1.2, are *preprocessing*, *segmentation*, *feature extraction*, optionally *dimensionality reduction* and finally *classification* [12], [13].

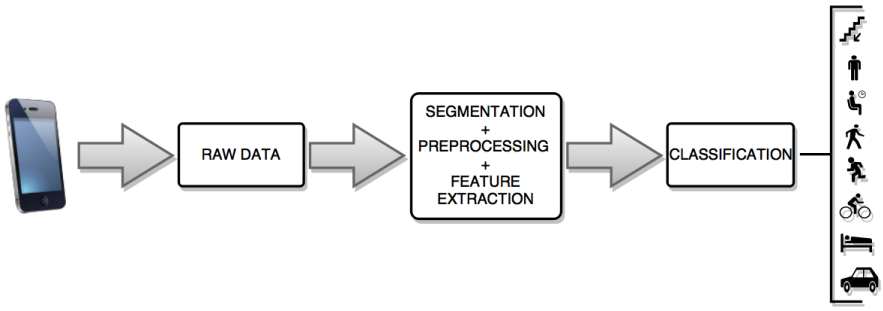


Figure 1.2: Main steps of activity recognition.

The preprocessing phase includes noise removal and representation of raw data. The segmentation step is applied to a continuous stream of sensor data in order to divide the signal into smaller segments. The feature extraction phase includes the generation of abstractions that better characterize the sensor data. In practice, raw data are reduced to a smaller set of features, called *feature vector*, that represents the original data in the best way. Dimensionality reduction is not mandatory and can be applied to remove irrelevant features to decrease the computational effort and also to increase the performance of the activity recognition process [14]. After all these intermediate steps, the processed data can be passed to the final classifier. The classification phase is a mapping of the extracted features to a set of activities. The classification procedures may involve different techniques, from a simple thresholding scheme to a more refined ML algorithm based on pattern recognition or neural networks.

Performance Measures

In the testing phase the output should be compared with the ground truth, i.e. the actual activity the user is performing, in order to evaluate the performance of the implemented classification system. Most of the times, cross validation is employed using a large part of the training set as ground truth and the other as test set [15]. Other times training and test data may be split [16].

The performance measures used in most of the studies on HAR with smartphones are: accuracy, precision, recall or sensitivity, confusion matrices and F-measure, derived from precision and recall [17]. Even though accuracy and precision are the mostly adopted performance measures in literature, since we often

deal with unbalanced datasets, other performance measures such as average precision and recall over all possible activities should be used.

1.1.4 Challenges of Activity Recognition with Smartphones

In this section we investigate the main specific challenges related to activity recognition with smartphones.

Continuous Sensing *Continuous sensing*, i.e. the continuous sampling of the sensor on the smartphone, is fundamental for every activity recognition application, but it highly affects the battery life of the mobile phone. When supporting continuous sensing applications, the user experience should not be affected, so that the user should be able to use the phone for making calls, sending SMS, taking pictures and browsing the web. Therefore, energy-efficient mechanisms that duty-cycle or turn off the sensors are required to save as much energy as possible.

Running Classifiers on Smartphones As mentioned above, most of the algorithms used in activity classification originate from ML techniques. However, in the field of activity recognition, especially with mobile phones, these algorithms may not reach very high performance because of the limited processing power and the battery constraints. Moreover, if we look at the literature on activity recognition using smartphones, we can observe that in most of the studies sensor data are collected from the smartphone and are then processed off-line, using most of the collected data for training [18], [19], [20]. However, off-line processing can be exploited only when online recognition is not necessary. For example, if we are interested in following the daily routine of a person, the sensors can collect the data during the day; the data can then be uploaded to a server and finally processed off-line for classification purposes. However, for applications where the user is given a program with a set of activities and their duration, we might be interested in what the user is actually doing and if he is performing the assigned tasks. Therefore, online activity recognition becomes important, especially for personal fitness and well-being applications running on smartphones.

Despite the limited resources available on the smartphones, examples of activity recognition show that simple classifiers like decision trees, minimum distance classifiers and K-Nearest Neighbours (KNN) can run on mobile phone providing

quite good accuracy rates [21]. When considering more complex and resource-intensive classifiers, we must rely on backend servers uploading the collected data and downloading the final results. However, with this approach, no support for real-time applications is available.

Phone Context Problem One of the main problem associated with mobile sensing is the *phone context problem*. It occurs when the phone is carried in an inappropriate position with respect to the event being sensed. For example, this problem is encountered when the application wants to take a sample from the light sensor but the phone is located in the pocket. Especially with accelerometer-based recognition systems, the phone location strongly influences the classification performance. In most of the studies using inertial sensors, the phone is forced to be carried in a particular location by the user, or only the signal magnitude, that is independent of the phone orientation, is to be considered. In [22], a two-phase activity recognition system is proposed. In the first phase, the position of the smartphone is detected while in the second phase the classification procedure is performed.

Although there exist other attempts in the literature to solve the phone context problem [23], [24], it is still an open issue, especially in real time applications including different poses and activities.

Training Burden Another challenging phase is the training phase. Usually, all the training models are created off-line and then used in the classification phase. This off-line training phase is not an easy task and it is essential for an activity recognition system to create effective models. Generally, the training sets are collected over a long period with a adequate number of test subjects. However, collecting a sufficient amount of data may be hard and tedious and so it would be interesting to have applications that do not require to deal with the burden of the training phase. Moreover, having classifiers able to recognize activities with an high detection rate with only a limited set of training data is very important. This is partially solved by semi-supervised approaches like active and self learning [25], [11]. Self-learning employs a single classier, which is used to classify unlabelled data. When the classifier's confidence in its prediction for a sample is high, it labels that sample with its prediction and adds it to the training set. When only

the most confident results are used as labels, the accuracy of the classifier should increase. Active learning, instead, does not use predictions as labels. Rather, it chooses samples of interest and asks the user to label them manually. The data labeled by the user are then added into the training data to recreate the classifier. Samples are chosen based on the confidence of prediction, but instead of using those with a high confidence like self-learning, those with the lowest confidence are selected.

1.2 Gait Analysis Using Wearable Sensors

A particular way or manner of walking is the definition for gait [26] and human gait is the continuous repetition of cycles, which generally consist of two steps each. Gait analysis is the study of human locomotion and it provides useful information in various areas such as health care and therapy.

Gait is not a new topic in the research field, in fact it has been investigated in several medical studies, such as [27], [28]. In this works Murray stated "Although the excursions of both pelvic and thoracic rotation in repeated trials of the same subject were similar, there were striking differences in these excursions among the individual subjects tested and also if all gait movements are considered, gait is unique ." This study presents a first input to gait uniqueness of each person. Moreover, in [26] it is shown how several human factors like aging, weight, injuries, operations, diseases, etc. may change a person's way of walking in a permanent or temporary way.

Moreover, Figure 1.3 shows how Dr. J. Perry divided the biological process of the musculo-skeletal system of a gait cycle, which consists five stance phase periods and three swing phase periods [29].

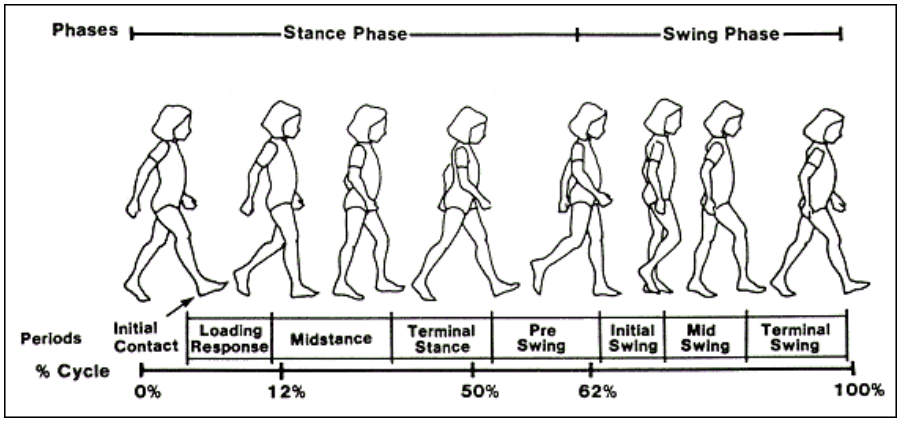


Figure 1.3: Gait phases in a normal gait cycle.

1. **Initial contact**: This is the initial phase of each gait cycle. It comprises the moment when the foot touches the floor.

2. **Loading response**: This is the initial double-stance period. It begins with the initial floor contact and continues until the other foot is lifted for swing.

3. **Midstance**: In this phase the limb advances over the stationary foot while the knee and hip extend.

4. **Terminal stance**: This phase completes the single-limb support. The stance begins with the heel rising and continues until the other foot strikes the ground. Throughout this phase, body weight moves ahead.

5. **Pre-swing**: This final phase of the stance is the second double-stance interval in the gait cycle. It begins with the Initial Contact (IC) of the opposite limb and ends when the foot lifts from the floor.

6. **Initial swing**: This phase is approximately one-third of the swing period, it begins with a lift of the foot from the floor and it ends when the swinging foot is opposite the stance foot.

7. **Mid-swing**: This phase begins as the swing limb is opposite the stance limb and ends when the swinging limb is forward and the tibia is vertical.

8. **Terminal swing**: This final phase of the swing begins with a vertical tibia and ends when the foot strikes the floor.

The sequential combination of these phases enables the limb to accomplish three basic tasks, namely, *weight acceptance*, *single-limb support* and *limb advancement*. Weight acceptance begins the stance period through IC and loading response. Single-limb support continues the stance through the midstance and terminal stance. Limb advancement begins in the pre-swing phase and continues through initial swing, mid-swing and terminal swing.

The main achievements of human gait analysis can be divided into three areas, namely, *kinematics*, *kinetics* and *Electromyography (EMG)*. The kinematics of the human gait describes the movements of the major joints and components of the lower extremity. Gait kinetics focuses on the study of forces and moments that result in the movement of human segments. The EMG of the human gait is mainly used to detect and analyze muscle activity during human walking.

For many years, the quantitative analysis of gait patterns has been studied in gait laboratories equipped with many sophisticated analysis and measurement devices that required specialized personnel and laboratory environment. Moreover, most of the equipment was costly and the data acquisition procedures were often cumbersome. More recently, different types of low cost and small sensors such as accelerometer, gyroscope, force sensors, etc. were used to perform human gait analysis because they are easy to use, they do not require a laboratory environment and their reliability is satisfactory [30]. A current and quite recent trend has seen the deployment of accelerometers in off-the-shelf cellphone handsets such as smartphones and, consequently, studies of gait analysis have been attempted using smartphones embedded sensors.The effectiveness of this approach has been shown in [31]. Here, the authors confirmed that smartphones with gait analysis applications have the capacity to quantify gait parameters with a degree of accuracy that is comparable to that of the tri-axial accelerometer. Hence, parameters like step length, step velocity, cadence and motion intensity can be calculated through the inertial sensors of the smartphone and can be used for anomaly detection or medical studies.

1.3 Gait Identification

Gait identification is a new biometric technology dealing with the identification of individuals through the analysis of the way they walk. Gait recognition systems, like HAR, have typically three main components. A low-level sensing module that gathers raw data using motion sensors. A feature processing and selection module that processes the raw sensor data and extracts a set of useful features. A classification module that uses the extracted features to identify the users. Researchers works differ according to the techniques and algorithms used in these three modules, but the common aspect is that the gait data acquisition is performed through wearable sensors.

In [5], it is used a device called *GaitShoe*, worn on shoes and equipped with an extensive sensor suite. The results suggested that GaitShoe is able to extract useful features and to recognize individual subjects, as well as groups of subjects with a similar gait. Moreover, Neural Networks appeared to be a promising method for discriminating between both individual subjects and between groups of subjects with normal gait, and groups of subjects with Parkinson's disease.

The first work aimed at investigating user identification from accelerometer signals was [32], where the recognition is based on the analysis of the three-dimensional acceleration signal produced by a portable device worn on a belt with fixed orientation. Processing is performed using peak detection, and for identification purposes they analyzed the correlation in the frequency domain. The results are quite promising with a mean Equal Error Rate (EER) of 13%.

In other works, data acquisition is performed with inertial sensors worn in different body positions leading to unrealistic scenarios, such as ankle, hip and waist. Moreover, inertial sensors are used in combination with other sensors (e.g GPS) due to their sensing ability.

Recently, attention has been given to accelerometer-based gait recognition on mobile devices due to their rapid deployment and to the explosion of their usage in people's daily lives. In 2010, Derawi et al. [33] acquired acceleration data from a mobile phone attached to the belt of the subject. Only the acceleration on the x-axis is used to extract gait cycles and they used a single representative average cycle for each person. Dynamic Time Wrapping [34] is used as comparison method. Finally, ML algorithms are investigated in the classification phase for

user identification. Supervised algorithms such as KNN, Support Vector Machine (SVM), Multi Layer Perceptron and, Decision Trees [35] are typically used.

1.4 Motivations and Contributions

This thesis describes the development of a novel system to extract meaningful gait patterns and gait parameters from smartphone-acquired accelerometer data and signals obtained from a reference video. Moreover, it is able to discriminate between different types of gait and to detect anomalies like limps or other walking problems. Therefore, it can be used in rehabilitation scenarios to monitor the evolution of patient's gait during the treatment. Additionally, it can be useful in Parkinson's disease studies to analyze the gait changes along different stages of the disease. Besides, this system can be included in a personal health counseling application in order to warn a person about his possible walking issues.

The proposed system includes three main sequential phases typically implemented in almost every gait recognition system:

- data acquisition;
- data preprocessing;
- patterns extraction.

This work differs from the approaches present in the literature for the inclusion of signals extracted from a reference video acquired during a walk and for the techniques and algorithms used in the patterns and parameters extraction phase.

First of all, for data collection and video acquisition, I used a last generation smartphone with built-in inertial sensors and camera. From the inertial sensor accelerometer and gyroscope data were acquired with the corresponding timestamps. The former are widely used for gait analysis and the latter, in this work, are used mainly in combination with accelerometer ones in order to extract gait cycles. Regarding the data acquisition procedure, volunteers walked in a real world scenario at their preferred walking speed. Every volunteer was asked to record four walking acquisitions: the first two in a way that he considered normal, the third simulating a limp on the right leg and the fourth simulating a limp on

the left leg. All acquisitions were performed with the smartphone located in the chest with a support that blocked it in a fixed vertical position.

From the recorded video, using different computer vision techniques and algorithms like optical flow estimation and frame keypoints extraction [1], the information about the positions of the user in the sagittal, transverse and vertical axis during the walking cycles are retrieved along with the pitch, roll and yaw rotations. Inspired by [36] and [37], IC and Final Contact (FC) instants of every step are detected and gait cycles are extracted. From the extracted gait cycles, classical parameters like stance and swing time, step length and step velocity are calculated.

For the pattern extraction phase, I proposed an unsupervised ML algorithm that is able to find meaningful clusters in the space of gait cycles creating a representative template for each of them. In particular, the proposed algorithm is an adaptation of the Growing Neural Gas (GNG) clustering algorithm [38]. In the classical GNG algorithm a new node is introduced in the network after a fixed number of iteration, while in this version, called Growing When Required (GWR), a new node is added only when no existing node sufficiently matches the input [39].

The rest of this thesis is organized as follows. In Chapter 2, I illustrate the literary work related to the proposed gait analysis system. In Chapter 3, I provide an introduction to computer vision and to the algorithm used to estimate the position and the rotation of the user during the walking acquisitions. In Chapter 4, I present an overview of unsupervised machine learning and unsupervised clustering. Moreover, the GNG and GWR algorithm are analyzed in detail presenting their main differences. In Chapter 5, I describe the design and the implementation of the proposed gait system analysis, while, in Chapter 6, the data logger application is presented together with the main issues faced during its implementation. Then, in Chapter 7, I present the experiments and tests carried out to asses the performance of the proposed system. Finally, in Chapter 8 I draw the conclusions.

2
Related Work

In this thesis, I consider a gait analysis system based on accelerometer and gyroscope collected data together with a reference video. Extracting meaningful and significative parameters and analyzing irregularities and anomalies in the walking gait of a person may be very useful for healthcare and prediction of some diseases like Parkinson or Alzheimer. The instruments and procedures of gait assessment should be simple, inexpensive and easy-to-use.

In the last decade, a lot of works related to the extraction and analysis of gait parameters and anomaly detection have been proposed in the literature. Most of them are based on the processing of raw data collected from Inertial Measurement Unit (IMU) placed in different parts of the body (e.g. ankles, chest, arm or lower-back). Other approaches investigated the detection of unusual walking patterns with computer vision techniques applied to an input reference video.

In order to obtain accurate gait parameters, different algorithms for the processing of the raw accelerometer and gyroscope data have been proposed. In [40], the authors reviewed five different estimation methods for gait event detection and temporal parameters extraction from acceleration signals collected from a single inertial measurement unit. Their results indicate that the obtained sensitivity ranges from 81% up to 100% across the different methods. The positive predictive values range from 94% to 100%. In the estimation of step and stride durations all methods are acceptable. On the contrary, when estimating swing

and stance times some differences were found due to errors of FC detection. In this work, the authors also show how the placement of the IMU in the lower trunk does not significantly affect the final accuracy. In [37], the authors present a gait parameter estimation method similar to the one I use in this thesis. They take advantage of the continuous wavelet transform that is able to denoise the signal preserving the underlying main frequencies. When combined with differentiation analysis, this technique shows good performance in suppressing noise, correcting baseline drift and resolving overlapping peaks problems. Moreover, it can detect IC and FC instants of every gait cycle without errors. They also present a method based on the filtered vertical component of the gyroscope that is able to distinguish whether an IC event comes from the right or left foot. In their experiments, all ICs and FCs are correctly detected and the time error is approximately 2% and 3% in each side, respectively.

In [41], the authors propose a personal health counseling application that is able to find some special features of the user and monitor their weekly or monthly changes. If irregular or abnormal walking parameters (like asymmetry or skew, stumbles or slip) are often detected, the user may have some physical problems that can possibly increase the fall risk. In this study the data are collected from an embedded tri-axial accelerometer and tri-axial gyroscope of a last generation smartphone located in the belt of the user. In the data collection phase, in order to avoid problems related to noise recursion, accelerometer drift, and to remove the effect of the gravity on the raw data, the authors propose low-pass filtering and a "zero velocity updating" technique assessing the drift of the accelerometer with every step. Thus, allowing the removal of the effects of the drift over one step. The walking distance is calculated through a quadratic integral acceleration over time. Mean step length and mean walking speed are estimated using the upward and downward movement of the trunk according to the inverted pendulum model of the body's Center of Mass (CoM) trajectory. Qin et al. [42] present a smartphone based system to collect and calculate classical gait parameters like step length, velocity, cadence, motion intensity and walking regularity. Moreover, the authors developed a prototype of gait parameter collection and visualization system on a laptop and a cell phone with an integrated fall and anomaly detection function. The anomaly detection algorithm, that includes a dynamic threshold set according to a normal distribution model, can find distinct abnormal gait

parameters that come from an occasional accident. If the latest data deviates from the normal activity model too much, the data will be set as an abnormal event. The proposed system shows high accuracy and reliability with respect to counting steps and walking duration with a global error smaller than 5.45%. In [43] another anomaly detection system is proposed. It is an accelerometer-based solution with predefined extracted features. This approach uses an unsupervised ML algorithm able to detect abnormal situations with a reasonable detection rate. In order to detect whether a pattern is classified as abnormal or not, a probability model based on Gaussian Mixture Model (GMM) is created. In order to define the suitable probability function and determine the optimal number of classes of the GMM, the Expectation-Maximization (EM) algorithm is used together with Bayesian Information Criterion (BIC). Using the defined probability density function, if the random variable associated with the current instance has a probability that is below a given threshold, likely it is associated with an abnormal event.

Gupta et al. [44] performed falling detection using a tri-axial accelerometer. They put an accelerometer on subjects' waist combining it with a pressure sensor in the insole. The authors propose a basic threshold-based algorithm and an innovative one based on frequency analysis which uses wavelet transform. Chen et al. [45] present a fall detection method using non invasive wearable sensors in conjunction with a wireless network. They use a low-power and low-cost Micro Electro-Mechanical Systems (MEMS) technology based accelerometer and build a wearable circuit on board for that. The device is then secured on the waist of the user in order to measure the acceleration of the subject. Once the norm of the acceleration exceeds a predefined threshold an impact is found. If there is no impact within the next several seconds, the system starts to look for orientation change. If no orientation change is detected, the subject is assumed to be injured or unconscious and an alarm system sends a warning message. Selvabala et al. [46] implement a human fall detection system using two different network architectures. This system is conceived with two sensors including a three-axis accelerometer and a passive infrared sensor to monitor the activities of elderly people. The real time output is compared with stored templates and the computational analysis is done within a microcontroller in real time. The system informs the caretaker through the wireless architecture whenever an abnormal

data is detected. Chehade et al. introduced a normal distribution walking model based on one feature extracted from acceleration signal during walking activities [47]. They tried different placements for the tri-axial accelerometer on the human body. Finally they found that the sensor on the chest performed best in separating stumble from normal activities. Stumble data, in fact, are considered as an abnormal deviation from the normal activity model, so they trained their model through a long period of history data.

Recent works demonstrated how spectrogram image processing can be exploited as a powerful signal processing tool in many different fields including speech processing and biomedical engineering. In [48], for example, El-Gohary shows the capability of spectrogram analysis using kinematic sensors to track tremors in Parkinson's disease patients in free-living conditions. In [49], the authors implemented a system that uses neural networks and clustering of wavelet-decomposed spectrogram images able to detect gait episodes. Signals collected from a chest-worn IMU are processed using Explicit Complementary Filter (ECF) in order to estimate and track the torso angle. Using the features obtained from the wavelet decomposition of spectrogram images, they use an Augmented Radial Basis Function Neural Network (ARBF) to classify gait episodes. ARBF is a variant of RBF neural network that has been used for the classification of head movement command using head-worn accelerometer [50] and classification of falls using waist-worn accelerometer [10]. Cluster centroids of ARBF are then optimized using Rapid Cluster Estimation (RCE). A pilot study of 11 participants suggests that the proposed approach is able to distinguish between walk and non-walk activities with up to 85.71% and 91.34% specificity.

In parallel with traditional inertial sensors-based gait analysis, researchers have recently begun to explore low-cost computer vision technologies to augment existing sensing networks into smarter visual monitoring systems to automatically assess human behaviours for potential medical diagnosis. For example, a tracking system for home-based rehabilitation is proposed in [51] to help stroke patients recover their mobility, Nait-Charif et al. [52] performed activity summarisation and fall detection in a supportive home environment and Gao et al. [53] analyzed eating activities of patients at a nursing home. In [54], a computer vision based system able to determine different walking styles and to detect whether a walking action deviates from usual walking patters is proposed. The analyzed

method starts with the extraction of human silhouettes from an input video and the computation of frame-to-frame optical flows, then motion metrics based on histogram representations of silhouettes-masked flow and finally gait analysis with eigenspace transformation. This work, based on the analysis of six different walking gaits, suggests good results with an accuracy of about 90%.

Differently from the existing sensor-based or smartphone-based gait analysis studies proposed in the literature, in this thesis I focus on merging the information coming from the embedded sensors of the chest-worn smartphone together with the signals extracted from the rear phone video camera, recorded while the user is walking. The main objectives of my work are to detect and extract meaningful patterns and parameters that characterize user's gait and to observe if different walking styles can be reliably detected in an unsupervised scenario. As in [54], I used different computer vision techniques like the computation of the frame-to-frame optical flows, described in detail in Chapter 3, that allows me to retrieve the rotation and the translation of the user's body during the acquisition. The encouraging results described in [37] concerning the detection of the IC and FC instants in the gait cycles led me to apply the same algorithms to the accelerometer and gyroscope data extracted from the smartphone.

3
Camera

3.1 Computer Vision

Between all the human senses, vision is largely known to be the one that has more potentiality. The biological system capabilities are very powerful: the eye is able to collect all the electromagnetic radiations coming from different objects and the brain can processes this information creating an image of the scene we are looking at. If we would give a definition of *Computer Vision*, we could say that it deals with image analysis and its main objectives are to detect *what* is present in the scene and *where* every element is located.

In [55], Ullman distinguishes between high-level and low-level computer vision. The former is focused on extracting some physical properties of the visible environment like tridimensional shape or the edges of the objects. Low-level computer vision processes are typically parallel, spatially uniform and independent from the problem and the prior knowledge associated with some particular objects.

On the other hand, high-level computer vision deals with the extraction of spatial relations, recognition and objects classification. High-level processes are usually applied to a portion of the image and depend on the objective of the computation and on the a priori knowledge associated with some particular objects.

High-level problems related to the perception and the recognition of the objects

make the reconstruction of the geometry of the scene a very difficult task. This task can be effectively described as the inverse of computer graphic, in fact, given:

- the geometric description of the scene;
- the radiometric description of the scene (light sources and surface properties);
- the complete description of the camera;

the calculator creates a "synthetic" image seen from the camera. The dimensionality reduction related to the geometric projection and the computation of the luminance and the radiometry of the scene make the problem under-constrained and consequently it has no unique solution.

A theory called reconstructionism [56], describes the computer vision problem by considering that an artificial or natural visual system is simply a system that processes information. Once all the physical assumptions are defined, the main problem is related to the tridimensional reconstruction of a pictorial scene. To summarize, the main purpose of computer vision, as defined in the reconstructionist paradigm, is a complete and accurate description of the world based on tridimensional primitives.

This approach has been contested in [57] where the authors claimed that the description of the world must not be too general, but instead it must be driven by a main objective (*purposivism*).

3.2 Camera Calibration

In this section I introduce a geometric camera model together with the problem related to the computation of the model's parameters (*calibration*).

3.2.1 Camera Model

The most common geometric camera model is the so called *pinhole* camera model that consist in a plane $\mathcal{R}$ and a point $\mathcal{C}$ (*center of projection*) at a distance f (*focal length*) from the plane. The line passing through $\mathcal{C}$ and orthogonal to $\mathcal{R}$ is the *optical axis* (z-axis of Figure 3.1), its intersection with $\mathcal{R}$ is called *principal*

point. The plane $\mathcal{F}$ that is parallel to $\mathcal{R}$ and contains the principal point is called *focal plane*. All the points that belong to this plane are projected towards infinity on the image plane.

In order to analytically describe the perspective projection of the camera, first I have to introduce some suitable cartesian reference systems to express the three-dimensional space coordinates of the point and the coordinates of its projection on the image plane.

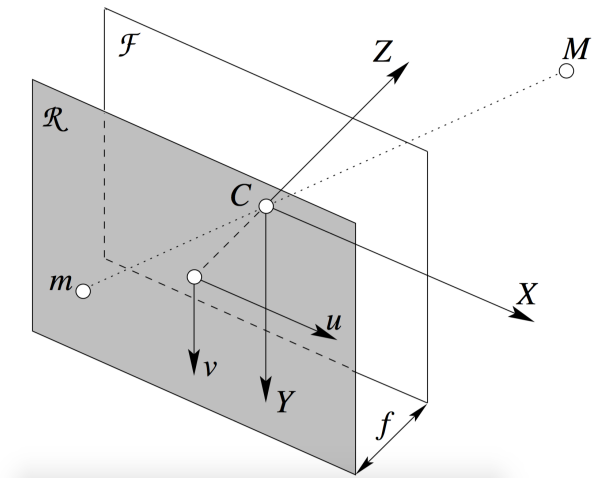


Figure 3.1: Geometric model of the camera, [1].

Simplified Model

Let us introduce a right-handed reference system (X, Y, Z) in the three-dimensional space centered in $\mathcal{C}$ with the Z-axis that corresponds to the optical axis. Moreover, let us define another reference system (u, v) in the plane $\mathcal{R}$ centered in the principal point with u and v respectively oriented like X and Y, as shown in Figure 3.1.

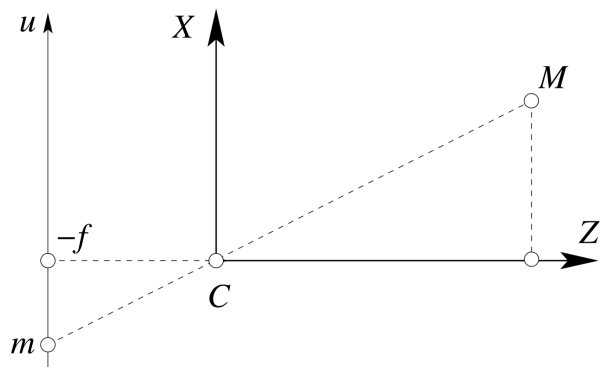


Figure 3.2: Simplified camera model, [1].

Let us consider a point M with three-dimensional space coordinates $\tilde{\mathbf{M}} = [x, y, z]^\top$ and let m with coordinates $\tilde{\mathbf{m}} = [u, v]^\top$ be its projection on $\mathcal{R}$ through $\mathcal{C}$. With simple consideration on triangular similarities (see Figure 3.2) we can write the following relation:

$$\frac{f}{z} = \frac{-u}{x} = \frac{-v}{y}, \tag{3.1}$$

and so:

$$\begin{cases} u = \frac{-f}{z}\ x \\ v = \frac{-f}{z}\ y \end{cases} . \tag{3.2}$$

Equation 3.2 is called *perspective projection*. The transformation from three-dimensional coordinates into two-dimensional coordinates is clearly non linear but, if we consider homogeneous coordinates, it becomes *linear*.

The homogeneous coordinates of a point in $\mathbb{R}^n$ correspond to a vector in $\mathbb{R}^{n+1}$. The additional coordinate specifies whether the point lies at the infinite ("0") or corresponds to a real point in the Euclidean space ("1" or different from "0"). Using homogeneous coordinates we can change the perspective projection, which is a non-linear relation, into a linear one.

If we define

$$\mathbf{m} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{3.3}$$

as the homogenous coordinates of m and M, the perspective equation can be

rewritten as:

$$z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -fx \\ -fy \\ z \end{bmatrix} = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{3.4}$$

In matrix notation we have:

$$z\mathbf{m} = P\mathbf{M}, \tag{3.5}$$

or, also:

$$\mathbf{m} \simeq P\mathbf{M}, \tag{3.6}$$

where $\simeq$ means "equal with respect to a scale factor" (z). Matrix P represents the geometric model of the camera and is called *camera matrix* or Perspective Projection Matrix (PPM).

General Model

A general and realistic camera model must take into account also:

- the rigid transformation between the camera and the scene;
- the pixelization (shape and dimension of the Charge-Coupled Device (CCD) and its location with respect to the optical center).

Intrinsic parameters Pixelization is introduced with an affine transformation that takes into account the optical center's translation and the possible rescaling of u and v axes:

$$\begin{cases} u = k_u \frac{-f}{z} x + u_0 \\ v = k_v \frac{-f}{z} y + v_0 \ , \end{cases} \tag{3.7}$$

where (u_0, v_0) are the coordinates of the principal point and k_u (k_v) is the inverse of the pixel dimension along u (v) direction and it is measured in $pixel \cdot m^{-1}$. In the general camera model the PPM becomes:

$$P = \begin{bmatrix} -fk_u & 0 & u_0 & 0 \\ 0 & -fk_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = K[I|0], \tag{3.8}$$

where

$$K = \begin{bmatrix} -fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.9}$$

Extrinsic parameters In order to take into account the fact that, in general, the reference system of the world may not coincide with the standard reference system of the camera, a rigid transformation that connects the two systems must be introduced. If we introduce a change of coordinates related to the introduction of a rotation R followed by a translation $\mathbf{t}$ and if we denote by $\mathbf{M_c}$ the homogeneous coordinates of a generic point in the standard reference system of the camera and by $\mathbf{M}$ the homogeneous coordinates of the same point in the reference system of the world, we can write:

$$\mathbf{M_c} = G\mathbf{M}, \tag{3.10}$$

where

$$G = \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix}. \tag{3.11}$$

Given equations 3.6 and 3.8, we have:

$$\mathbf{m} \simeq K[I|0]\mathbf{M_c} = K[I|0]G\mathbf{M}, \tag{3.12}$$

and, finally:

$$P = K[I|0]G. \tag{3.13}$$

Equation 3.13 is the most general expression of the PPM: matrix G contains the extrinsic camera parameters, matrix K contains the intrinsic parameters and matrix $[I|0]$ represents the perspective transformation in the standard reference system. The extrinsic parameters are six: three for the rotation and three for the translation. Sometimes the next different factorization may be useful:

$$P = K[R|\mathbf{t}]. \tag{3.14}$$

Normalized Coordinates

If we introduce the change of coordinates defined by

$$p = K^{-1}m, \tag{3.15}$$

the PPM reduces to $[I|0]$. These special coordinates are called *normalized coordinates* or image coordinates. In order to compute the normalized coordinates all intrinsic camera parameters must be known.

Scaling Factor

If we analyze equations 3.5 and 3.6 we can see that they differ only from a scaling factor. In fact, if we substitute P with γP in 3.6 we obtain the same projection, $\forall \gamma \in \mathbb{R} \setminus \{0\}$.

Degrees of Freedom

Matrix P is composed of 12 elements but it only has 11 degrees of freedom since one is lost because of the scaling factor.

3.2.2 Calibration

Camera calibration corresponds to the computation of the intrinsic and extrinsic camera model parameters. The main idea is that if the tridimensional projections of some points are known (calibration points), it its possible to compute the unknown parameters by solving the perspective projection equations. There exist several calibration methods. Among these methods, those called direct formulate the calibration problem identifying the camera parameters as the unknowns. Others, like [58], instead, resolve the PPM estimation problems, from which all the parameters can be retrieved.

3.3 Optical Flow

In camera-centered coordinates, each point on a tridimensional surface moves along a tridimensional path. When projected onto the image plane, each point produces a bidimensional path. The instantaneous direction of this path is the

velocity. The bidimensional velocities for all visible surface points is often referred to the bidimensional motion field. The goal of optical flow estimation is to compute an approximation of the motion field from the time-varying image intensity.

A common starting point for optical flow estimation is to assume that pixel intensities are translated from one frame to the next:

$$I(\mathbf{x}, t) = I(\mathbf{x} + \mathbf{v}\Delta t,\ t + \Delta t) \tag{3.16}$$

where $I(\mathbf{x}, t)$ is the image intensity as a function of space and time. Of course, *brightness constancy* rarely holds exactly. The underlying assumption is that surface radiance remains fixed from one frame to the next. Although unrealistic, it is remarkable that the brightness constancy assumption of Eq. 3.16 works very well in practice.

Assuming that the displaced image is well approximated by a first-order Taylor series we obtain [1]:

$$I(\mathbf{x} + \mathbf{v}\Delta t, t + \Delta t) = I(\mathbf{x}, t) + gradI(\mathbf{x}, t)^\top \begin{bmatrix} \mathbf{v}\Delta t \\ \Delta t \end{bmatrix}. \tag{3.17}$$

Splitting the temporal and spatial component we obtain:

$$I(\mathbf{x} + \mathbf{v}\Delta t, t + \Delta t) = I(\mathbf{x}, t) + \nabla I(\mathbf{x}, t)^\top \mathbf{v}\Delta t + I_t(\mathbf{x}, t)\Delta t, \tag{3.18}$$

where ∇I indicates the spatial gradient $(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})^\top$ and I_t is the temporal derivative of I.

If we substitute Eq. 3.16 into Eq. 3.18 and divide by Δt, we obtain:

$$\nabla I(\mathbf{x}, t)^\top \mathbf{v} + I_t(\mathbf{x}, t) = 0. \tag{3.19}$$

This final expression is called *image brightness constancy equation.* We can observe that there are two unknowns (the components of $(v) = (v_x, v_y)$) and only one equation. Hence, the problem is underconstrained. Eq. 3.19 defines a constraint on $\nabla I(\mathbf{x}, t)^\top \mathbf{v}$ that corresponds to the projection of the motion field on the gradient's direction.

3.3.1 Optical Flow Estimation

In order to estimate the optical flow, Eq. 3.19 is used but other constrains must be added since the problem is underconstrained.

Lucas and Kanade

Lukas and Kanade optical flow estimation method is based on the assumption that the optical flow $\mathbf{v}$ is constant in a $n \times n$ neighbourhood W of every point. This basic assumption allows to obtain more than a single equation for the same unknown $\mathbf{v}$. Every point $\mathbf{x}_i \in W$ provides a linear equation:

$$\nabla I(\mathbf{x}_i, t)^\top \mathbf{v} = -I_t(\mathbf{x}_i, t). \tag{3.20}$$

Given all $n \times n$ equations we obtain the following linear system:

$$A\mathbf{v} = \mathbf{b} \tag{3.21}$$

where

$$A = \begin{bmatrix} \nabla I(\mathbf{x_1}, t)^\top \\ \vdots \\ \nabla I(\mathbf{x_{n \times n}}, t)^\top \end{bmatrix} \quad b = \begin{bmatrix} -I_t(\mathbf{x_1}, t) \\ \vdots \\ -I_t(\mathbf{x_{n \times n}}, t) \end{bmatrix} \tag{3.22}$$

The least square solution of this overdetermined system is given by:

$$\mathbf{v} = A^{-1}\mathbf{b} = (A^\top A)^{-1} A^\top \mathbf{b}. \tag{3.23}$$

Usually a gaussian weight function $w(\mathbf{x}_i)$, defined on W, is introduced to enhance the central points and weight less the external ones [1].

Kanade, Lucas and Tomasi Feature Tracker

Starting from the Lucas and Kanade algorithm, Kanade, Lucas and Tomasi feature tracker (KLT) is based on two main steps:

- frame features extraction;
- feature tracking.

In the feature extraction phase we must have $\lambda_{max}/\lambda_{min} \simeq 1$, where λ_{max} and λ_{min} are respectively the maximum and minimum eigenvalue of matrix A. In practice, we must also have large eigenvalues in order to discard weak textures that may be corrupted by noise. The selection of the keypoints by the KLT feature tracker is based on the following condition:

$$min(\lambda_1, \lambda_2) > \lambda_{th}, \tag{3.24}$$

where λ_{th} is a predefined threshold. If λ_{min} is sufficiently large, the condition on the similarity of the eigenvalues is automatically satisfied since the value of λ_{max} has an upper limit.

The feature tracking phase is performed between every couple of consecutive frames. More in details, the algorithm proceeds as described in Algorithm 3.1.

Algorithm 3.1 KLT Tracker

Input: Sequential images
Output: Keypoints tracking

1. spatial filtering with Gaussian 2D kernel;
2. temporal filtering with 1D Gaussian kernel;
3. first frame keypoints extraction;
4. for every frame couple $I(t)$, $I(t+1)$:
   - tracking: for every keypoint in $I(t)$ compute $\mathbf{v}$ on a $n \times n$ window centered in the keypoint;
   - apply motion to every point and obtain the position in the successive frame $I(t+1)$.

3.3.2 Keypoints Extraction

In this section I will introduce a rotation and scale-invariant method to extract *keypoint* or *feature points* from image frames.

SIFT

Scale-Invariant Feature Transform (SIFT) has been introduced by D. Lowe in 2004 [59] and it includes five main steps: scale-space extrema detection, keypoint localization, orientation assignment, keypoint description and keypoint matching.

Scale-space extrema detection In this first phase, in order to detect local extrema over scale and space, a scale-space filtering technique is used. SIFT algorithm uses Difference of Gaussian (DoG) which is an approximation of Laplacian of Gaussian (LoG) and is obtained as the difference of gaussian blurring of an image with two different σ, where σ is a parameter that identifies different scales in the image. DoG is done for different octaves of the image in gaussian pyramid and its representation can be seen in Figure 3.3.

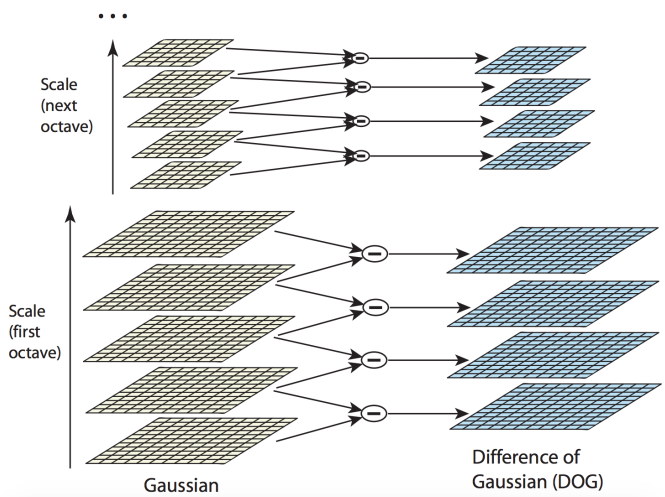


Figure 3.3: Difference of Gaussian.

Once DoG are found, local extrema over scale and space are identified. For example, one pixel in an image is compared with its eight neighbours as well as nine pixels in the next scale and nine pixels in the previous scale as shown in Figure 3.4. If it is a local extrema, it is a potential keypoint. It basically means that the keypoint is best represented in that scale.

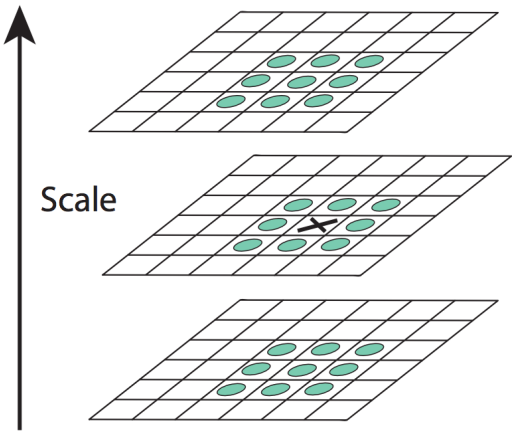


Figure 3.4: Maxima and minima of DoG.

Keypoint localization Once all the potential keypoints are found, they have to be refined to get more accurate results. A Taylor series expansion of scale space is used to get more accurate location of extrema, and if the intensity at these extrema is less than a predefined threshold value, it is rejected. DoG has higher response for edges, thus these also need to be removed. For this, a 2×2 Hessian matrix is used to compute the principal curvature. The two eigenvalues are computed and if their ratio is grater than a certain threshold (i.e., an edge is detected), the keypoint is discarded.

Orientation assignment In this phase an orientation is assigned to each point in order to achieve invariance with respect to image rotation. A neighbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction are calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. It is weighted by the gradient magnitude and the Gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. Then, the highest peak in the histogram is taken and any peak above 80% of it is also included in the calculation of the orientation. This procedure creates keypoints with same location and scale, but different directions.

Keypoint description Once the keypoint descriptor is created, a 16×16-pixel neighbourhood around the keypoint is analyzed. It is divided into 16 sub-blocks of 4×4 size. For each sub-block, eight-bin orientation histogram is created. So, a total of 128 bins values are available. It is then represented as a vector to form a keypoint descriptor. In addition, several measures are taken in order to achieve robustness against illumination changes, rotation, etc.

Keypoint matching Keypoints between two images are matched by identifying their nearest neighbours. In some cases, however, the second closest-match may be very near to the first. This may happen due to noise or some other reason. In this case, the ratio between the closest-distance and the second-closest distance is taken. If it is greater than 0.8, they are rejected. This step eliminates around 90% of false matches while discarding only around 5% of the correct ones.

3.4 Motion and Structure

In this section I discuss the *structure from motion* problem. Given different views of the scene taken from a moving camera with known intrinsic parameters, and given a set of corresponding points, we have to reconstruct the camera motion and the structure of the scene.

3.4.1 Essential Matrix

Let us assume that we have a camera with known intrinsic parameters that is moving in a static scenario following an unknown trajectory. Let us consider two images taken from the camera in two different temporal instants and assume that there exists a given set of common points between the two images. Let P and P' be the PPM of the cameras corresponding to the two temporal instants and let $\mathbf{p} = K^{-1}\mathbf{m}$, $\mathbf{p}' = K'^{-1}\mathbf{m}'$ be the normalized coordinates of the matching points.

If we take the reference system of the first camera as the world reference system, we can write the following PPM:

$$P = [I|0] \quad and \quad P' = [I|0]G = [R|\mathbf{t}] \tag{3.25}$$

From 3.25 we can derive the bilinear form that relates the matching points:

$$\mathbf{p}'^{\top}[\mathbf{t}]_{\times}R\mathbf{p} = 0^{1}. \tag{3.26}$$

The matrix

$$E \triangleq [\mathbf{t}]_{\times}R, \tag{3.27}$$

is called *essential matrix*. It depends on three parameters for the rotation and two for the translation. We can observe that the essential matrix is defined with respect to a scale factor and it is singular since $det[t]_{\times} = 0$. In order to get to five degrees of freedom, two more constraints must be added. The following Theorem 3.2 states that these two constraints are given by the equality of the non-zero singular values of E.

3.4.2 Essential Matrix Factorization

Lemma 3.1. *Given a rotation matrix R and two orthogonal matrices U and V, then $det(UV^{\top})URV^{\top}$ is a rotation matrix, i.e., it has positive determinant.*

Theorem 3.2. *A 3×3 real matrix E can be factorized into the product of a non-zero antisymmetric matrix and a rotation matrix if and only if E has two equal non zero singular values and one singular value that is equal to 0.*

Proof. Let $E = SR$ where R is a rotation matrix and S is antisymmetric. Let $S = [\mathbf{t}]_{\times}$ where $\|t\| = 1$. Let U be a rotation matrix such that $U\mathbf{t} = [0, 0, 1]^{\top} \triangleq \mathbf{a}$, so $S = [\mathbf{t}]_{\times} = [U^{\top}\mathbf{a}]_{\times}$. Now we can write:

$$S = [U^{\top}\mathbf{a}]_{\times} = U^{\top}[\mathbf{a}]_{\times}U.$$

[1] Given a vector $\mathbf{a} \in \mathbb{R}$, the matrix

$$[a]_{\times} \triangleq \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix},$$

behaves as an external product, i.e. $[\mathbf{a}]_{\times}b = a \times b$.

Let us consider matrix $EE^\top$:

$$EE^\top = SRR^\top S^\top = SS^\top = U^\top [\mathbf{a}]_\times U U^\top [\mathbf{a}]_\times^\top U = U^\top \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} U.$$

The diagonal elements are the eigenvalues of $EE^\top$, i.e. the squared singular values of E. This fact proves one implication.

Let us consider now the other way around. Let $E = UDV^\top$ be the Singular Value Decomposition (SVD) of E, with $D = diag(1, 1, 0)$, U and V orthogonal. The key observation is that:

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \triangleq S'R',$$

where S' is antisymmetric and R' is a rotation matrix. Therefore

$$\begin{aligned} E = UDV^\top = US'R'V^\top = (US'U^\top)(UR'V^\top) = \\ = det(UV^\top)(US'U^\top)det(UV^\top)(UR'V^\top). \end{aligned}$$

If we take $S = det(UV^\top)US'U^\top$ and $R = det(UV^\top)UR'V^\top$, the factorization is: $E = SR$. In fact, $US'U^\top$ is antisymmetric and matrix $det(UV^\top)UR'V^\top$ is orthogonal with positive determinant. □

This is not the only possible factorization. It is possible to change the sign of D by changing the sign of S' or by transposing R' and, therefore, we have four possible factorizations given by:

$$S = U(\pm S')U^\top \tag{3.28}$$

$$R = det(UV^\top)UR'V^\top \quad or \quad R = det(UV^\top)UR'^\top V^\top \tag{3.29}$$

where

$$S' \triangleq \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad R' \triangleq \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.30}$$

3.4.3 Essential Matrix Computation

Given a sufficiently large set of matching points $\{(\mathbf{p}_i, \mathbf{p}_i^\top)|i = 1, \cdots, n\}$ defined in normalized coordinates, we want to determine the essential matrix E connecting the points in the bilinear relation described by:

$$\mathbf{p}_i^\top E \mathbf{p}_i = 0. \tag{3.31}$$

The unknown matrix can be retrieved with the vec[2] operator and the Kronecker product[3]. In fact, we can derive:

$$\mathbf{p}_i'^\top E \mathbf{p}_i = 0 \Longleftrightarrow vec(\mathbf{p}_i'^\top E \mathbf{p}_i) = 0 \Longleftrightarrow (\mathbf{p}_i^\top \otimes \mathbf{p}_i'^\top) vec(E) = 0. \tag{3.32}$$

Therefore, every couple of matching points generate an homogeneous linear equation with nine unknowns that correspond to the essential matrix E. Given n couples of matching points we have a linear system of n equations:

$$\underbrace{\begin{bmatrix} \mathbf{p}_1^\top \otimes \mathbf{p}_1^{\top\prime} \\ \mathbf{p}_2^\top \otimes \mathbf{p}_2^{\top\prime} \\ \vdots \\ \mathbf{p}_n^\top \otimes \mathbf{p}_n^{\top\prime} \end{bmatrix}}_{U_n} vec(E) = 0. \tag{3.33}$$

The solution of this homogeneous linear system is the kernel of U_n. With $n = 8$ the kernel matrix has dimension equal to one and so the solution is determined with respect to a scaling factor. Therefore, this method is called *eight-points algorithm* [60].

In practice, more than eight points are available and the elements of the essen-

[2] *Vectorization*: vectorization of a given $m \times n$ matrix A, denoted by $vec(A)$, is the $mn \times 1$ vector obtained by stacking the columns of A.

[3] *Kronecker product*: Let A be a $m \times n$ matrix and B a $p \times q$ matrix. The Kronecker product between A and B is the $mp \times nq$ matrix defined by

$$A \otimes B = \begin{bmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{bmatrix}.$$

tial matrix can be retrieved by solving a least square linear problem. The solution is the unitary eigenvector that corresponds to the minimum eigenvalue of $U_n^\top U_n$ that can be easily computed with the SVD of U_n.

4

An Overview of Unsupervised Learning

4.1 Unsupervised Learning

As described in Section 1.1.3, *supervised learning* relies on sample observations (labels) associated to a specific output used to train a given classifier. Let us denote by $Y = (Y_1, \ldots, Y_m)$ the response variables for a given set of input predictor variables $X = (X_1, \ldots, X_p)$. Let $x_i = (x_{i1}, \ldots, x_{ip})$ be the input for the ith training case, and let y_i be a response measurement. The predictions are based on the training samples $(x_1, y_1), \ldots, (x_n, y_n)$ of previously solved cases, where the joint values of all the variables are known.

Assuming that (X, Y) are random variables represented by some joint probability density $P(X, Y)$. Then, the supervised learning can be formally characterized as a density estimation problem where one is concerned with determining properties of the conditional probability $P(Y|X)$. Usually the properties of interest are the "location" parameters μ that minimize the expected error at each x,

$$\mu(x) = \underset{\theta}{\operatorname{argmin}}\, E_{Y|X} L(Y, \theta), \tag{4.1}$$

where $L(Y, \theta)$ is a given loss function.

If we apply the definition of conditional probability we get

$$P(X, Y) = P(Y|X) \cdot P(X),$$

where $P(X)$ is the joint marginal density of the X values alone. In supervised learning $P(X)$ is typically of no direct concern. One is mainly interested in the properties of the conditional density $P(Y|X)$. Since the dimension of Y is usually small (often one), and only its location $\mu(x)$ is of interest, the problem is greatly simplified.

In *unsupervised learning*, or "learning without a teacher", one has a set of N observations $(x_1, x_2, \ldots, x_n)$ of a random p-vector X having joint density $P(X)$. The goal is to directly infer the properties of this probability density without the help of a supervisor providing correct answers or degree-of-error for each observation. The dimension of X is sometimes much higher than in supervised learning and the properties of interest are often more complicated than simple location estimates. These factors are somewhat mitigated by the fact that X represents all of the variables under consideration; one is not required to infer how the properties of $P(X)$ change, conditioned on the changing values of another set of variables.

In low-dimensional problems, there are a variety of effective nonparametric methods for directly estimating the density $P(X)$ itself at all X-values, and representing it graphically. Owing to the curse of dimensionality, these methods fail when dealing with high-dimensional data. One must settle for estimating rather crude global models, such as Gaussian mixtures or various simple descriptive statistics that characterize $P(x)$.

Generally, these descriptive statistics attempt to charaterize X-values, or collections of such values, where $P(X)$ is relatively large. Principal components [61], multidimensional scaling, self-organizing maps [62] and principal curves [63], for example, attempt to identify low-dimensional manifolds within the X-space that represent high dimensional data densities. This provides information about the associations among the variables and whether or not they can be considered as functions of a smaller set of "latent" variables. Cluster analysis attempts to find multiple convex regions of the X-space that contain modes of $P(X)$. This can tell whether or not $P(X)$ can be represented by a mixture of simpler densities

representing distinct types or classes of observations. Mixture modeling has a similar goal. Association rules attempt to construct a simple description of the high density regions in the special case of very high dimensional binary-valued data.

With supervised learning there is a clear measure of success or lack of it, that can be utilized to test the performance in particular situations and to compare the effectiveness of different methods in various scenarios. The lack of success is directly measured by the expected loss over the joint distribution $P(X, Y)$. This can be estimated in a variety of ways including cross-validation. In the context of unsupervised learning, there is no such direct measure of success. It is difficult to establish the validity of inferences drawn from the output of most unsupervised learning algorithm. One must rely on heuristic methods not only for motivating the algorithms, as is often the case in supervised learning, but also for judging the quality of the results. This uncomfortable situation has led to the heavy proliferation of methods, since the effectiveness cannot be verified directly.

4.2 Unsupervised Clustering

Clustering (or *cluster analysis*) aims to organize a collection of data items into clusters, such that items within a cluster are more "similar" to each other than they are to items in the other clusters. This notion of similarity can be expressed in very different ways, according to the purpose of the study, to domain-specific assumptions and to prior knowledge of the problem.

Clustering is usually performed when no information is available concerning the membership of data items to predefined classes. For this reason, clustering is traditionally seen as part of unsupervised learning. In this section I describe *unsupervised clustering* to distinguish it from a more recent an less common approach that makes use of a small amount of supervision to "guide" or "adjust" clustering called *semi-supervised clustering*.

To support the extensive use of clustering in computer vision, pattern recognition, information retrieval, data mining, etc., very different methods were developed in several communities. Detailed surveys of this can be found in [64], [65] or [66]. In the following, I review some important and core concepts of cluster analysis and describe the main categories of clustering methods.

4.2.1 A Taxonomy of Clustering Methods

In this section, I describe some criteria that provide significant distinction between different clustering methods:

- *Objective of clustering.* Many methods aim at finding a single *partition* of the collection of items. However, obtaining a *hierarchy* of clusters can provide more flexibility and other methods focus on this. A partition of the data can be obtained from a hierarchy by cutting the tree of clusters at some level.

- *Nature of the data items.* Most clustering methods were developed for numerical data, but some can deal with categorical data or with both numerical and categorical data

- *Nature of the available information.* Many information methods rely on rich representations of the data (e.g. vectorial) that leave one define prototypes, data distributions, multidimensional intervals, etc., beside computing (dis)similarities. Other methods only require the evaluation of pairwise (dis)similarities between data items. While imposing less restrictions on the data, these methods usually have a higher computational complexity.

- *Nature of the clusters.* The degree of membership of a data item to a cluster is either in $[0, 1]$ if the clusters are *fuzzy* or in $\{0, 1\}$ if the clusters are *crisp*. For fuzzy clusters, data items can belong to some degree to several clusters that don't have hierarchical relations with each other. This distinction between fuzzy and crisp can concern both the clustering mechanisms and their results. Crisp clusters can always be obtained from fuzzy clusters.

- *Clustering criterion.* Clusters can be seen either as distant *compact* sets or as *dense* sets separated by low density regions. Unlike density, compactness usually has strong implications on the shape of the clusters, so methods that focus on compactness should be distinguished from methods that focus on the density.

Several taxonomies of clustering methods were suggested in [67], [65] or [66]. By focusing on some of the discriminating criteria mentioned above, I propose a simplified taxonomy inspired by the one suggested in [65].

Partitional clustering Partitional clustering aims at directly obtaining a single *partition* of the collection of items into clusters. Many of these methods are based on the iterative optimization of a criterion function reflecting the "agreement" between the data and the partition. Here are some important categories of partitional clustering methods:

- *Methods using the squared error* rely on the possibility to represent each cluster by a prototype and attempt to minimize a cost function that is the sum over all the data items of the squared distance between each item and the prototype of the cluster it is assigned to. In general, the prototypes are the cluster centroids, as in the popular k-mean algorithm [68]. Several solutions have been developed for cases where a centroid cannot be defined, such as the k-medoid method [66]. In this method the prototype of a cluster is an item that is "central" to the cluster. Another method is the k-modes method [69] that is an extension of the previous one to categorical data. By employing the squared error criterion with a Minkowski metric [70] or a Mahalanobis metric [71], one makes the implicit assumption that clusters have elliptic shape. The use of multiple prototypes for each cluster or of more sophisticated distance measures can remove this restriction.

  Fuzzy versions of methods based on the squared error were defined as, for example, the Fuzzy C-Means [72]. When compared to their crisp counterparts, fuzzy methods are more successful in avoiding local minima of the cost function and can model situations where clusters actually overlap. To make the results of clustering less sensitive to outliers, several fuzzy solutions have been proposed.

  Many early methods assumed that the number of clusters was known a priori; since this is rarely the case, techniques for finding an "appropriate" number of clusters had to be devised. This is an important issue for partitional clustering in general. For methods based on the squared error, the problem is partly solved by adding a *regularization* term to the cost function.

- Density-based methods consider that clusters are dense sets of data items separated by less dense regions; clusters may have arbitrary shape and data

items can be arbitrarily distributed. Many method, such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [73], rely on the study of the density of items in the neighbourhood of each item.

Grid-based solutions, such as DenClue [74] or CLIQUE [75], can be included among density based methods, even though they were mostly developed for spatial data mining. These methods quantize the space of the data items into a finite number of cells having high density of items; isolated data items are ignored. Quantization steps and density thresholds are common parameters for these methods. Many of the *graph-theoretic* clustering methods are also related to density-based clustering. The data item are represented as nodes in a graph and the dissimilarity between two items is the "length" of the edge between the corresponding nodes. In several methods, a cluster is a subgraph that remains connected after the removal of the longest edges of the graph. However, some other graph-theoretical methods rely on the extraction of *cliques*. This makes these methods more related to the squared error ones.

- Mixture-resolving methods assume that the data items in a cluster are drawn from one of several distributions (usually Gaussian) and attempt to estimate the parameters of all these distributions. The introduction of the EM algorithm [76] was an important step in solving the parameter estimation problem. Mixture-resolving methods make rather strong assumptions regarding the distribution of the data. The choice of the number of clusters for these methods was studied in different works like [77] or [78]. In some cases a model for the noise is explicitly considered. Most mixture-resolving methods view each cluster as a single simple distribution and thus strongly constrain the shape of the clusters.

Hierarchical clustering. Hierarchical clustering aims at obtaining a hierarchy of clusters, called *dendrogram*, that shows how the clusters are related to each other. These methods proceed either by iteratively merging small clusters into larger ones (agglomerative algorithms) or by splitting large clusters (divisive algorithm). A partition of the data items can be obtained by cutting the dendrogram at a desired level [79], [80].

Agglomerative algorithms need a criterion for merging small clusters into larger ones. Most of the criteria deal with the merging of pairs of clusters and are variants of the classical single-link, complete-link or minimum-variance criteria. The use of the single-link criterion can be related to density-based methods but often produces upsetting effects: clusters that are "linked" by a "line" of items cannot be separated or most items are individually merged to one or a few clusters. The use of the complete-link or of the minimum-variance criteria relates more to squared error methods.

4.2.2 Cluster Validity Analysis

As discussed above, the effectiveness of an unsupervised learning procedure is usually more difficult to assess than that of a supervised one. Several questions can be asked regarding the application of clustering methods [81]:

1. Are there clusters in the data?

2. Are the identified clusters in agreement with the prior knowledge of the problem?

3. Do the identified clusters fit well the data?

4. Are the results obtained by a method better than those obtained by another?

The first question concerns the *cluster tendency* of the data and should in principle be answered before attempting to perform clustering, using specific statistical tests. Unfortunately, such tests are not always very helpful and require the formulation of specific test hypotheses.

The other questions concern the analysis of the cluster validity and can only be answered after the application of some clustering methods to the data. According to [65], three main types of validation procedures can be distinguished:

- *External* validation consists in finding an answer to the second question and can only be performed when some general characteristics of the clusters or relations between specific items are explicit.

- *Internal* validation concerns the third question and is based on the evaluation of the "agreement" between the data and the partition. In the literature, different validity indices have been proposed for the categories of clustering methods discussed above. Some of the possible validity indices are: the modified Humbert's statistic, related to the alignment between the dissimilarity matrix and the crisp partition matrix. The Davies-Bouldin, roughly defined as the ratio between within-cluster scatter and between-cluster separation index [82]. Dunn's index for the diameter of a set and for the distance between sets [83]. The average partition density, instead, is obtained as the mean ratio between the "sum of central members" of each cluster and the volume of the cluster. Finally, the Xie-Beni index, is the ratio between the average intra-cluster variation and the minimum distance between cluster centers [84].

- *Relative* comparisons attempt to provide an answer to the fourth question and are usually the main application of the indices defined in internal validation. Such comparisons are often employed in order to select good values for important parameters, such as the number of clusters.

In this work, I employed a modified version of the unsupervised Growing Neural Gas (GNG) clustering algorithm, called Growing When Required (GWR). In order to fully clarify how GNG and GWR can be implemented, I dedicate the next two sections to the presentation of their architectures.

4.3 Growing Neural Gas

In this section I describe in detail and explain the GNG unsupervised incremental clustering algorithm proposed by Bernd Fritzke in [38]. Given some input distribution in $\mathbb{R}^n$, the GNG incrementally creates a graph, or network of nodes, where each node in the graph has a position in $\mathbb{R}^n$. GNG can be used for vector quantization by finding code-vectors in clusters. It can also be used for finding topological structures that closely reflect the structure of the input distribution. Moreover, GNG only has parameters that are constant in time and since it is incremental, there is no need to determine the number of nodes (clusters) a priori. Starting with two nodes, the algorithm builds a graph in which pair of nodes are

considered neighbours if they are connected by an edge. The neighbour information is maintained throughout execution by a variant of competitive Hebbian learning [16], that is,

> *For each input signal $\boldsymbol{\xi}$ an edge is inserted between the two closest nodes. The distance between nodes is measured by the Euclidean distance.*

4.3.1 GNG Pseudo-code

The GNG algorithm assumes that each node k consists of the following:

- $\mathbf{w}_k$: a reference vector in $\mathbb{R}^n$.
- $error_k$: a local accumulated error variable.
- A set of edges defining the topological neighbours of node k.

The reference vector may be interpreted as the position of a node in the input space. The local accumulated error is a statistical measure that is used for determining the appropriate insertion points for new nodes. Further, each edge has an age variable used to decide when to remove old edges in order to keep the topology updated.

INIT: Create two randomly positioned nodes, connect them with a zero age edge and set their errors to 0.

1. Generate an input vector $\boldsymbol{\xi}$ conforming to some distribution.
2. Locate the two nodes s and t nearest to $\boldsymbol{\xi}$, that is, the two nodes with reference vectors $\mathbf{w}_s$ and $\mathbf{w}_t$ such that $\|\mathbf{w}_s - \boldsymbol{\xi}\|$ is the smallest value and $\|\mathbf{w}_t - \boldsymbol{\xi}\|$ is the second smallest, for all nodes k.
3. The winner-node s must update its local error variable so the squared distance between $\mathbf{w}_s$ and $\boldsymbol{\xi}$ has to be added to $error_s$

$$error_s \leftarrow error_s + \|\mathbf{w}_s - \boldsymbol{\xi}\| \,, \tag{4.2}$$

4. Move s and its topological neighbours (i.e all nodes connected to s by an edge) towards $\boldsymbol{\xi}$ by fractions e_w and e_n of the distance. ($e_w,\ e_n \in [0, 1]$)

$$\mathbf{w_s} \leftarrow \mathbf{w_s} + e_w(\boldsymbol{\xi} - \mathbf{w}_s), \tag{4.3}$$

$$\mathbf{w_n} \leftarrow \mathbf{w_n} + e_n(\boldsymbol{\xi} - \mathbf{w}_n),\ \forall n \in Neighbours(s). \tag{4.4}$$

5. Increment the age of all edges from node s to its topological neighbours.

6. If s and t are connected by an edge, then set the age of that edge to 0. If they are not connected then create and edge between them.

7. If there are any edges with an age larger than a_{max} then remove them. If, after this, there are nodes with no edges then remove these nodes.

8. If the current iteration is an integer multiple of λ and the maximum node-count has not been reached, then insert a new node. Insertion of a new node r is done as follows:

   - Find the node u with largest error.
   - Among the neighbours of u, find the node v with the largest error.
   - Insert the new node r between u and v as follows:

$$\mathbf{w_r} \leftarrow \frac{(\mathbf{w}_u + \mathbf{w}_v)}{2}. \tag{4.5}$$

   - Create edges between u and r and v and r, and then remove the edge between u and v.
   - Decrease the error-variable of u and v and set the error of node r:

$$error_u \leftarrow \alpha \times error_u, \tag{4.6}$$

$$error_v \leftarrow \alpha \times error_v, \tag{4.7}$$

$$error_r \leftarrow error_u. \tag{4.8}$$

9. Decrease all error-variable of all nodes j by a factor β:

$$error_j \leftarrow \beta \times error_j. \tag{4.9}$$

10. If the stopping criterion is not met then repeat.

    The stopping criterion might be, for example, whether the performance on a test set is good enough or not, or when a maximum number of nodes has been reached, etc.

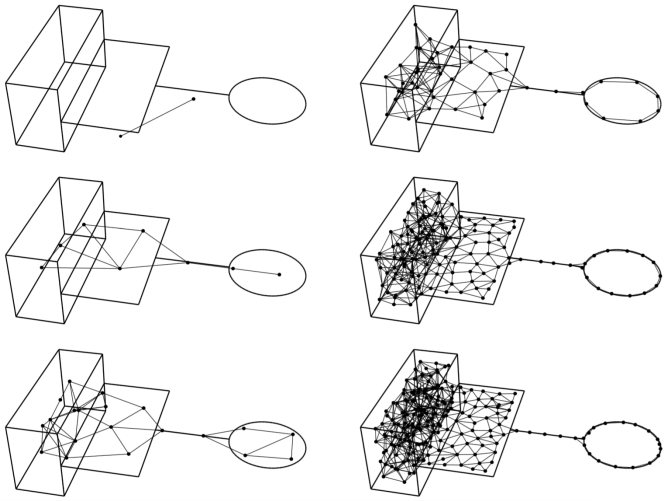


Figure 4.1: The GNG network adapts to a signal distribution that has different dimensionalities in different areas of the input space. Here are shown the initial network consisting of two randomly placed units and the network after 600, 1800, 5000, 15000 and 20000 input signals have been applied, [38].

4.3.2 GNG Explained

The Local Accumulated Error

Equation 4.2 describes how the local error is updated. Updating the error with the squared distance is a way of detecting nodes that cover a larger portion of the inputs distribution. Consequently these nodes will have, statistically, a faster growing error than other nodes. Large coverage is then equivalent to larger updates of the local error, since the input at greater distances will be mapped into the node. Since we want to minimize the error, knowing where the error is large is useful when inserting new nodes. In Equation 4.9 a global reduction of all local errors is performed, in order to give recent errors greater influence and to keep the local error from growing too much.

Node Movements

Equations 4.3 and 4.4 deal with nodes movement. The principle is the same for the winner node and its neighbours. Figure 4.2 shows the idea behind the movement of the winner node.

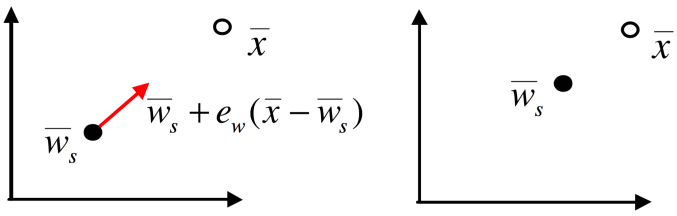


Figure 4.2: Winner node movement, [2].

The winner-node s is shifted along the scaled difference vector $(\boldsymbol{\xi} - \mathbf{w}_s)$. The scaling amount is denoted by e_w and its value is between 0 and 1. The movement is linear and the further the node is from the input the greater the shift distance. This is also true for the neighbour movement. However, the neighbour translation vector is scaled with a constant e_n much smaller than e_w.

Node Insertion

In GNG, nodes are inserted at fixed rate. Every λ-th iteration a new node is inserted between the node with largest error and its neighbour with the largest error. Having a fixed insertion rate policy might not always be desirable, since it may lead to unnecessary or untimely insertions. As will be discussed in the next session, a different policy based directly on the local or global mean error might perhaps be preferreble.

In the case of fixed insertion rate, the λ parameter has significant impact on the performance of the algorithm. Setting it too small will result in poor initial distribution of nodes, since the statistical local error will be badly approximated and since the nodes have not had a chance to distribute themselves over the input space.

4.4 The Growing When Required Network

In this section I describe an evolution of the GNG clustering algorithm called Growing When Required (GWR) [39].

4.4.1 GNG vs. GWR

Like GNG, GWR is a growing network-based incremental clustering algorithm that introduces a different criterion for the insertion and the initialization of a new node. Differently from GNG, where nodes are only added at a fixed rate, in GWR a node is added whenever the current input is not matched by any of the existing nodes to some (arbitrary) accuracy. Moreover, a new node is not added to support the node with highest error but it is added between the two best matching nodes. This has the benefit that once the input space is matched within some defined error bound by the network nodes, the network will stop growing, but it will still be able to grow again if the input distribution changes. The GWR algorithm decides when to add new nodes by evaluating the activity of the node that best matches the current input. In this setup, good matches are identified by high activity. GWR also uses information about how often that node has fired previously. While the network is learning it may be the case that the node that is the best match is still not actually a good match (so its activity is low). If this node has not fired often it may be that the node is untrained, and just needs further training. However, if the node has often won, then its activity should be high, as the training will ensure that the node matches the input well. If the activity of the node is low, then a new node is needed to match the current input and so one is added. The training of each node can be modeled in different ways, the simplest one is the use of a simple counter that is incremented whenever that node is the best match.

An alternative to using the simple counter is to have a variable that decreases exponentially from 1 to 0, so that new nodes have a value of 1 and nodes that have fired frequently are close to 0. This is equivalent to a counter with an upper limit, but has few benefits. First of all, the fact that neighbours of the winning node are also trained can be acknowledged, as their variables can also decrease, although in a smoother manner. Also, the number of times that a node has fired can be

taken into account in the learning rate (see Eq. 4.18), so that nodes that have fired frequently are trained less. This approach removes a problem that affects networks that learn continuously. In this type of networks, in fact, the weights of well-trained nodes continue to move slightly, so that they do not converge to a stable configuration.

4.4.2 GWR Pseudo-code

I now detail the steps of the algorithm. Let A be the set of map nodes, and $C \subseteq A \times A$ be the set of connections between nodes in the map field. Let the input distribution be $p(\boldsymbol{\xi})$, for inputs $\boldsymbol{\xi}$. Define $\mathbf{w}_n$ as the weight vector of node n.

INIT: Create two nodes for the set A

$$A = \{n_1, n_2\},$$

with n_1, n_2 randomly initialized from $p(\boldsymbol{\xi})$. Define C to be the empty set

$$C = \emptyset.$$

Then, each iteration of the algorithm is:

1. Generate a data sample $\boldsymbol{\xi}$ for the input to the network.

2. For each node i in the network, calculate the distance from the input $\|\boldsymbol{\xi} - \mathbf{w}_i\|$.

3. Select the best and the second best matching node, that is the nodes $s, t \in A$ such that

$$s = \underset{n \in A}{\operatorname{argmin}} \|\boldsymbol{\xi} - \mathbf{w}_n\| \tag{4.10}$$

$$t = \underset{n \in A \setminus \{s\}}{\operatorname{argmin}} \|\boldsymbol{\xi} - \mathbf{w}_n\|, \tag{4.11}$$

where $\mathbf{w}_n$ is the weight vector of node n.

4. If there is not a connection between s and t, create it

$$C = C \cup \{(s,t)\}, \tag{4.12}$$

otherwise set the age of this connection to 0.

5. Calculate the activity of the best matching unit

$$a = exp(-\|\boldsymbol{\xi} - \mathbf{w}_s\|). \tag{4.13}$$

6. If the activity $a <$ activity threshold a_t and firing counter $<$ firing counter threshold h_t then a new node should be added between the two best matching nodes:

   - Add the new node r

$$A = A \cup \{r\}. \tag{4.14}$$

   - Create the new weight vector, setting the weights to be:

$$\mathbf{w}_r = (\mathbf{w}_s + \boldsymbol{\xi})/2. \tag{4.15}$$

   - Insert edges between r and s and between r and t:

$$C = C \cup \{(r,s),(r,t)\}. \tag{4.16}$$

   - Remove the link between s and t:

$$C = C \setminus \{(s,t)\}. \tag{4.17}$$

7. If a new node is not added, adapt the position of the winning node and its neighbours i:

$$\boldsymbol{\Delta}\mathbf{w}_s = e_b \times h_s \times (\boldsymbol{\xi} - \mathbf{w}_s) \tag{4.18}$$

$$\boldsymbol{\Delta}\mathbf{w}_i = e_n \times h_i \times (\boldsymbol{\xi} - \mathbf{w}_s) \tag{4.19}$$

where $0 < e_n < e_b < 1$ and h_s is the values of the firing counter for node s.

8. Update age of the edges with an end at s

$$age_{(s,i)} = age_{(s,i)} + 1. \tag{4.20}$$

9. Reduce the counter of how frequently the winning node s has fired according to

$$h_s(t) = h_0 - \frac{S(t)}{\alpha_b}(1 - e^{-\alpha_b t/\tau_b}) \tag{4.21}$$

and the counter of its neighbours i

$$h_i(t) = h_0 - \frac{S(t)}{\alpha_n}(1 - e^{-\alpha_n t/\tau_n}) \tag{4.22}$$

where $h_i(t)$ is the size of the firing variable for node i, h_0 the initial strength and $S(t)$ is the stimulus length, usually set to 1. α_n, α_b, τ_n and τ_b are constants controlling the behaviour of the curve. The firing counter of the winner node reduces faster than those of its neighbours. Equation 4.21 is the solution to the differential equation

$$\tau_b \frac{dh_s(t)}{dt} = \alpha_b[h_0 - h_s(t)] - S(t), \tag{4.23}$$

which was proposed by Stanley in [85] and models how the efficacy of an habituating synapse reduces over time.

10. Check if there are any nodes or edges to delete, i.e. if there are any nodes that no longer have any neighbours or edges that are older than the greatest allowed age a_{max}.

11. If further inputs are available, return to step 1 unless some stopping criterion has been satisfied.

5
The Gait Analysis System

In this chapter I present how the tools described in the previous sections are combined together to develop a gait analysis system based on smartphone-acquired inertial data and video able to extract meaningful users' parameters and gait patterns.

The system is composed of four main blocks:

- Data acquisition.
- Data and video processing.
- GWR clustering and patterns extraction.
- Parameters extraction.

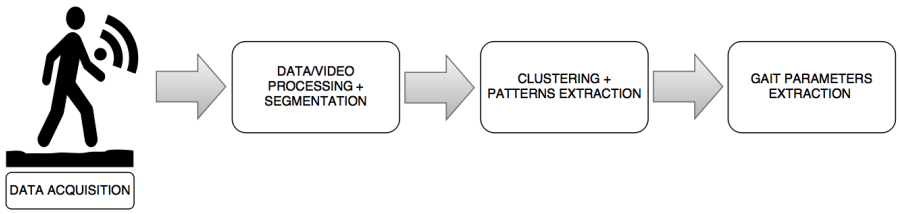


Figure 5.1: General scheme of the proposed gait analysis system.

These main steps are represented in Figure 5.1 and in the next sections I describe in detail each of them, highlighting the fundamental data processing techniques involved.

5.1 Data Acquisition

Input data for the designed system are collected using a Asus Zenfone 2 smartphone. If features a 2.3 GHz quad-core Intel Atom Central Processing Unit (CPU) and it comes with 4GB of RAM. The Operating System (OS) is Android 5 Lollipop. It is equipped with several built in sensors. The sensors of interest for this work are: GPS, proximity sensors, light sensor and IMU. In particular, IMU comprises a tri-axial PSH accelerometer sensor, a gyroscope and a magnetometer, whose specifications are reported in Table 5.1.

Sensor	PSH Accelerometer	PSH Gyroscope	PSH Magnetometer
Vendor	Intel Inc.	Intel Inc.	Intel Inc.
Range	39.227	34.907	800
Resolution	0.01 (0.024%)	0.002 (0.005%)	0.5 (0.062%)
Power	0.006mA	6.1mA	0.1mA

Table 5.1: IMU specifics of Asus Zenfone 2 used for data acquisition.

Data collection is performed with an ad-hoc developed Android application, called *Activity Logger Video* (described in detail in Chapter 6). This application is used to set up the parameters acquisition, give information about the user, collect and save data in the non-volatile memory and, optionally, send them to a File Transfer Protocol (FTP) server. During the acquisition phase, the smartphone is carried with the rear-facing camera looking forward and is placed in an ah-hoc made chest support as shown in Figure 5.2. This support allows the smartphone to be carried in a steady vertical position during the whole acquisition process and thus no orientation transformations are needed. Moreover, in such conditions, the reference video can be acquired without any orientation and occlusion issues.

Figure 5.2: Chest support for smartphone.

Raw data were collected from different volunteers, either males or females, aged between 21 and 54 years. Volunteers were asked to walk in a real world scenario at preferred walking speed along a straight line. Each of them performed four different walking acquisitions: for the first two, the individuals walked as they normally do. For the third and fourth acquisitions, a limp was forced on the right and left leg, respectively. The duration of each acquisition was approximately 35 seconds. Inertial data and the reference video were then saved in the non-volatile memory of the smartphone so that they could be sent to an FTP server through a WiFi network or through cellular data. The server runs on a Raspberry Pi module that is used for data storage. All collected data are then downloaded and processed in a PC.

At the end of each acquisition, only accelerometer and gyroscope data vectors on each axis together with the reference video are considered and processed in the subsequent phases. One of the common problems of collecting inertial data from the built in sensors of a smartphone is that the sampling rate can vary during an acquisition. The sampling rate, in fact, is related to the computational load of

the operative system stack and, as can be seen in Figure 5.3, some smartphones can have a sampling rate that has a distribution with standard deviation larger than zero. Hence, an interpolation phase of all gathered signals is necessary.

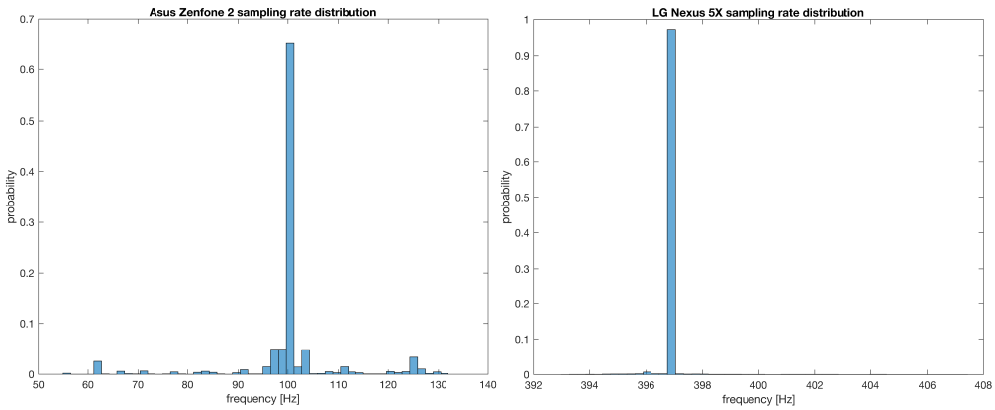


Figure 5.3: Comparison of the sampling frequency distribution of the smartphone employed in the data acquisition (Asus Zenfone 2) and another smartphone (LG Nexus 5X).

As for the video, it was acquired with the rear-facing camera of the smartphone, and the frame rate was set to 30 fps with a H.264 compression, an encoding bitrate of 3 Mbit/s and a resolution of 720×576 pixel. With these parameters every acquisition video has a size of around 13MB.

All the signals extracted from the reference video with the tools described in Section 5.2.2 must also be interpolated to ensure they have the same sampling rate of the inertial accelerometer and gyroscope signals.

5.2 Data and Video Processing

5.2.1 Data Processing

Raw data collected from the accelerometer and gyroscope sensors, together with the signals extracted from the reference video, need to be processed in order to remove possible high-frequency noise and to extract gait cycles. The extracted gait cycles are then utilized to form the dataset for the clustering and pattern extraction phase.

The data processing step includes four main phases: interpolation, filtering, cycles extraction and normalization.

Interpolation

As discussed above, since the sampling rate is not constant during the acquisition due to the non-real-time nature of the Android Operative System, an interpolation step is needed. Moreover, smartphones output data values whenever there is a change in the sensor and, therefore, the time intervals between two samples are not evenly spaced and differ for each sensor. This is solved through a spline interpolation to ensure a sampling rate of $f_s = 200$ Hz and a fixed time interval between any two consecutive samples.

Filtering

Figure 5.4 shows the Power Spectral Density (PSD) of the accelerometer signals through the Welch's method [86], considering a full walking trace and setting the Hanning window length to 1 s, with half window overlap. Most of the signal power, as expected, is located at low frequencies, mostly below 40 Hz. Hence, in order to smooth the signals and remove high-frequency motion artifacts, a lowpass Finite Impulse Respose (FIR) filter of order 40 and cutoff frequency $f_c = 40$ Hz is applied to all the available signals.

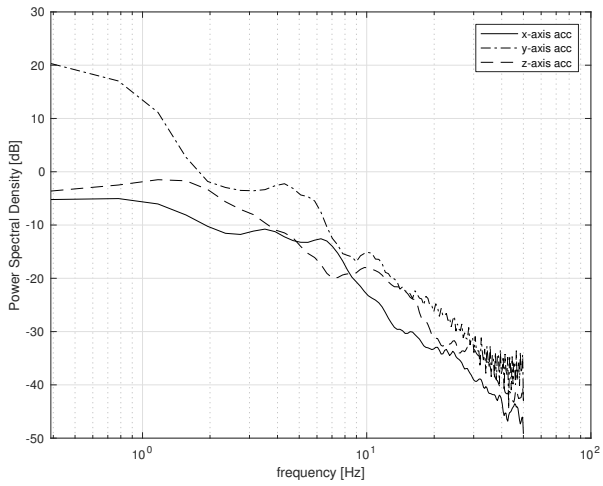


Figure 5.4: Power spectral density of the three-axial accelerometer data.

An example of the output of the described filtering procedure applied to the y-axis acceleration can be seen in Figure 5.5. As expected, the filter denoises the input signal reducing the effect of high-frequency noise.

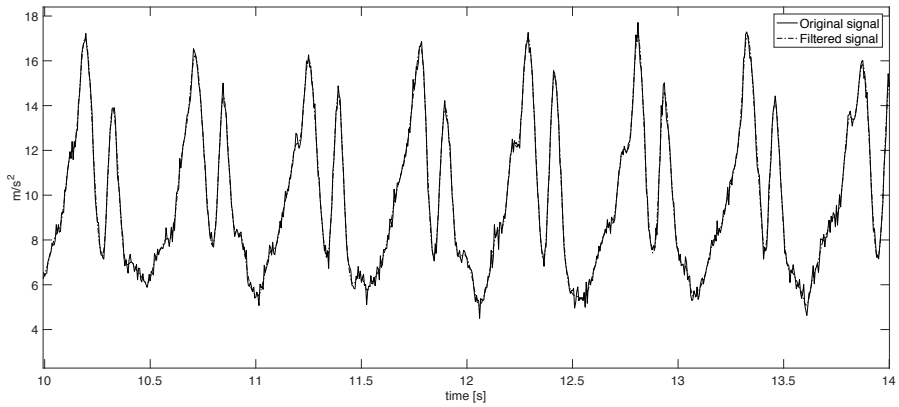


Figure 5.5: Example of y-axis accelerometer data before and after the filtering procedure.

Cycles Extraction

Filtered signals are then passed through a cycle extraction phase. As discussed in Section 1.2, human gait follows a cyclic behaviour where there is a periodic repetition of a pattern, known as cycle, that corresponds approximately to two human steps.

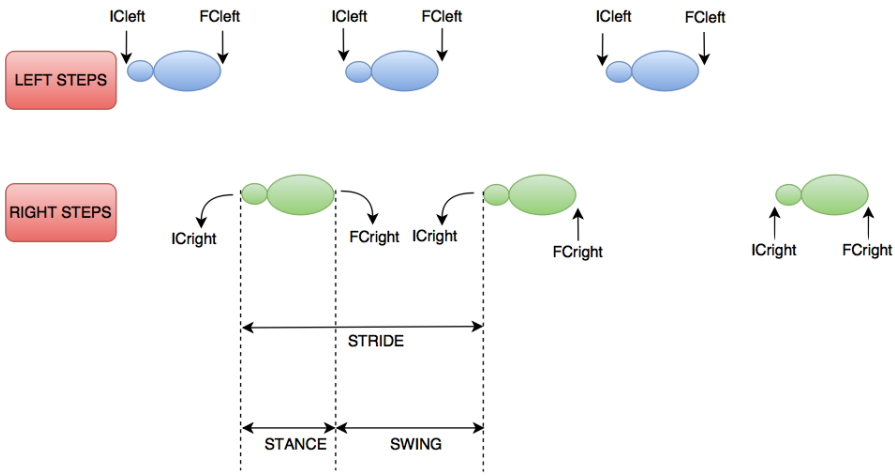


Figure 5.6: Stride, stance and swing times.

In order to extract and identify gait cycles different techniques have been proposed. The one I implemented in this work is based on that used in [36]. The IC and the FC events (see Figure 5.6) within each cycle are estimated by analyzing the vertical component $\mathbf{a}_y$ of the accelerometer data.

IC and FC events are determined using a Continuous Wavelet Transform (CWT) transform of $\mathbf{a}_y$ which is first integrated and then differentiated using a Gaussian CWT. The IC events, as can be seen in Figure 5.7, are detected as the local minima of the CWT. A further differentiation resulted in the local maxima corresponding to the FC events. In order to exclude possible spurious results, a further optimization step is proposed: only IC and FC peaks within a predetermined timed interval (0.25 - 2.25 s) are included. As demonstrated in [36], this technique leads to better estimation results.

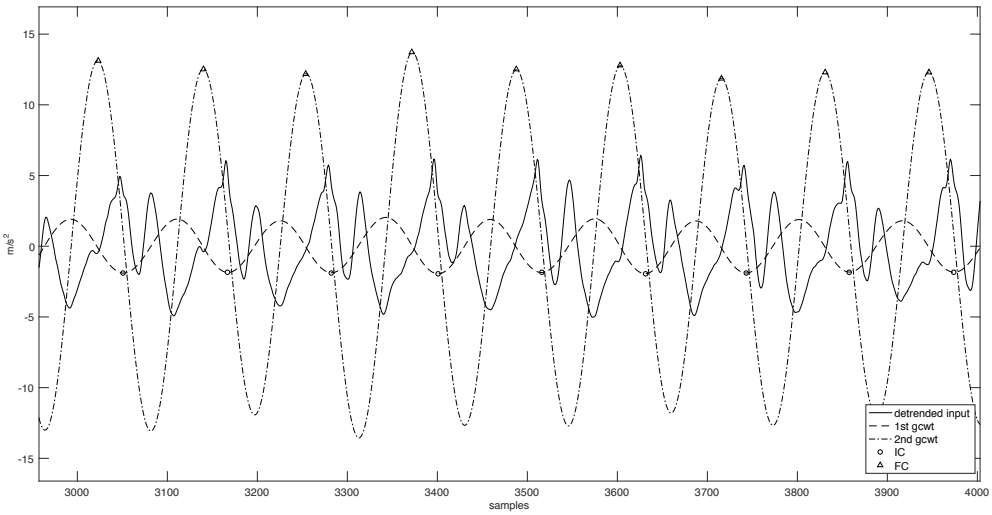


Figure 5.7: Example of ICs (circles) and FCs (triangles) detection. The solid line represents $\mathbf{a}_y$, the dashed line represents the differentiated with Gaussian CWT of $\mathbf{a}_y$ and the dash dotted line represents the second Gaussian CWT differentiation of $\mathbf{a}_y$.

Left and right steps are then identified by the sign, for every IC, of the vertical axis angular velocity $\mathbf{g}_y$ filtered with a low-pass 4th-order Butterworth filter with cutoff frequency $f_c = 2$ Hz (see Figure 5.8).

After this processing phase, gait cycles instants can be easily identified. In fact, a generic walking cycle i starts at $IC(i)$ and ends at $IC(i+2)$. Moreover, in order to avoid possible errors at the beginning and at the end of each acquisition,

the first and last 2 cycles are excluded from the computation. It is thus possible to locate the walking cycles vectors in all the available signals.

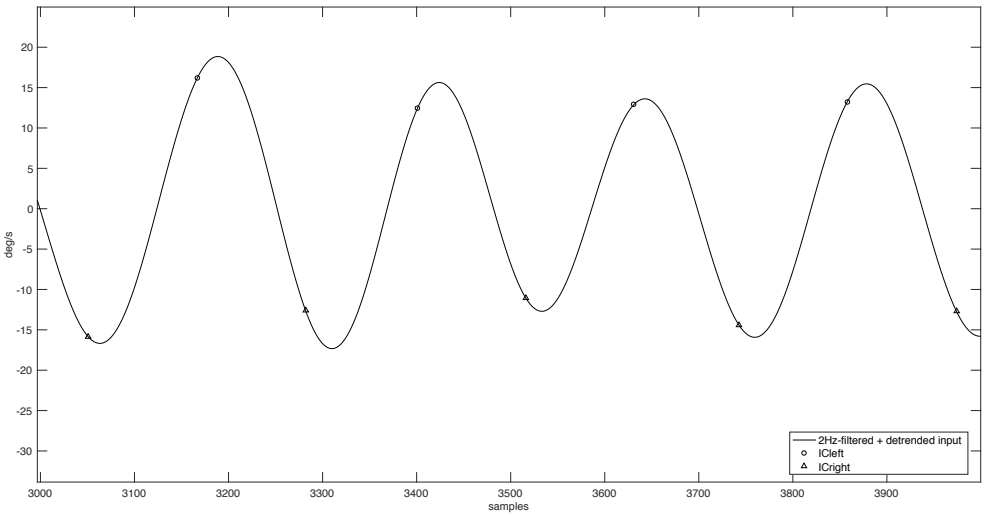


Figure 5.8: An example of left and right IC events detection. The solid line represents the filtered angular velocity. Positive or negative sign of the filtered signal indicates left and right ICs, respectively.

Other useful parameters like stance, swing time, step length and step velocity can be extracted from the IC and FC events and from the accelerometer data related to every walking cycles. The computation of these parameters is described in Section 5.4.

Normalization

Each extracted gait cycle has a different duration, which depends on the walking speed and the stride length. Because of that, the accelerometer data and the signals extracted from the video (these are the signals that will be used as input to the clustering algorithm), have variable-size and, hence, a further adjustment is necessary. This is done using a spline interpolation to represent all the walking cycles through vectors of $N = 200$ samples each. This value of N is selected in order to avoid aliasing. In fact, assuming a maximum duration of $\tau = 2$ seconds, and a signal bandwidth of $B = 40$ Hz, a number of samples $N > 2B\tau = 160$ samples/cycle is required.

5.2.2 Video Processing

In this section I formalize the problem of extracting the signals representing the position of the user in the three-dimensional space and I describe all the steps that allow to retrieve this information. I implemented this algorithm in Python using the OpenCV-Python library [87].

OpenCV-Python

OpenCV is an open source C_{++} library for image processing and computer vision, originally developed by Intel and now supported by Willow Garage. It is free for both commercial and non-commercial use. It is a library composed by many inbuilt functions mainly aimed at image processing. Now it has several hundreds of image processing and computer vision algorithms which make developing advanced computer vision applications easy and efficient.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C_{++} API and the Python language. Extending Python with C/C_{++} modules provides two advantages: first, the code is as fast as the original C/C_{++} code and second, it less time-consuming to code in Python than C/C_{++}. OpenCV-Python makes use of Numpy [88], which is a highly optimized library for numerical operation with MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy as SciPy and Matplotlib.

Problem Formulation

Input We have a stream of gray scale images coming from the camera. Let the frames, captured at time t and $t+1$ be referred to as I_t and I_{t+1}. We have prior knowledge of all intrinsic parameters of the camera, obtained via the calibration technique described in Section 3.2.

Output For every image, we need to find the rotation matrix R and the translation vector t, from which information about the motion and the rotation of the user between two successive frames can be extracted.

Algorithm Outline

1. Capture images: I_t, I_{t+1}.
2. Undistort the above images using the camera parameters.
3. Check if the acquired image is corrupted (number of white pixels $> 90\%$ total number of pixels).
4. Use SIFT algorithm to detect keypoints in I_t, and track them to I_{t+1} with KLT tracking. A new detection is triggered if the number of keypoints drops below a certain threshold.

Left step. Right step.

Figure 5.9: KLT tracker example for two frames representing a left step and a right step. The extracted keypoints are represented by the bold points, while the colored lines represent the estimated direction of the optical flow.

5. Compute essential matrix $\mathbf{E}$. In order to estimate the essential matrix, an evolution of the *eight-points algorithm* called *Nister's 5-point algorithm* is employed [89]. Moreover, an iterative method called *RANSAC* is used to detect the correspondences between two successive frames needed for the computation of $\mathbf{E}$ [90].
6. Estimate $\mathbf{R}$ and $\mathbf{t}$ from $\mathbf{E}$.
7. Recover yaw, roll and pitch rotations from $\mathbf{R}$.

RANSAC If all of the computed correspondences are perfect, then, with the *Nister's 5-point algorithm*, we would need only five feature correspondences between two successive frames to accurately estimate the motion. However, the feature tracking algorithms are not perfect, and therefore we have several erroneous correspondences. A standard technique of handling outliers when doing model estimation is RANSAC. At every iteration, it randomly samples five points from a set of correspondences, estimate the essential matrix, and then check if the other points are inliers when using this essential matrix. The algorithm terminates after a fixed number of iterations, and the essential matrix with which the highest number of points agree is used.

Determining yaw, pitch and roll from a rotation matrix Given a rotation matrix

$$R = \begin{pmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{pmatrix},$$

if we denote by α, β and γ respectively the roll, yaw and pitch angles (see Figure 5.10), we have that:

$$\alpha = tan^{-1}(r_{10}/r_{00}), \tag{5.1}$$

$$\beta = tan^{-1}(-r_{20}/\sqrt{r_{21}^2 + r_{22}^2}), \tag{5.2}$$

$$\gamma = tan^{-1}(r_{21}/r_{22}). \tag{5.3}$$

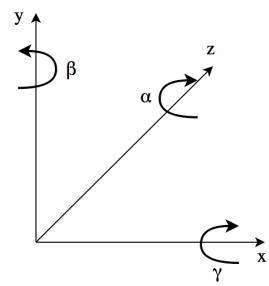


Figure 5.10: Roll, pitch and yaw angles.

5.3 GWR Clustering and Pattern Extraction

Once all the previous steps are completed, the GWR algorithm is utilized to extract meaningful clusters from the data and to obtain a reference pattern from each of them. Not all the available signals are used as input to the algorithm, but only the accelerometer and the signals extracted from the video are considered. The gyroscope data, as will be discussed in Chapter 7, are left out. In particular, the signals extracted from the video are obtained considering the difference between the yaw, pitch and roll angles of two consecutive frames. Therefore, the input to the algorithm is a 6×200 matrix (see Figure 5.11) composed by the three components of the acceleration vector and the three signals representing the rotation in three axes.

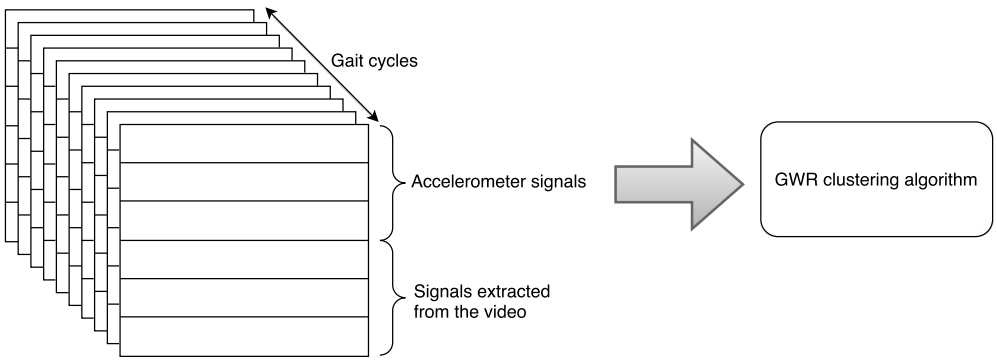


Figure 5.11: Input matrices of the GWR clustering algorithm. At every round a new matrix representing a different gait cycle is used as input to the algorithm.

I recall that every component of the gait cycles is represented by a 200-sample signal after the normalization step. The clustering algorithm, then, takes as input the matrices representing the extracted gait cycles and tries to put together all the gait samples that have similar characteristics in order to distinguish different walking styles.

Since the GWR algorithm, as described in Section 4.4, takes as input a one-dimensional signal, I have modified it in order to deal with multidimensional inputs. In this extended version of the GWR algorithm every node is provided with a 6×200 matrix that, at every round, is compared to the one of the input sample in order to discover the two closest nodes. To compare two different

matrices, however, a distance measure between them must be introduced. Given two $a \times b$ matrices $\mathbf{I}$ and $\mathbf{U}$, their distance $d(\mathbf{I}, \mathbf{U})$ can be defined as:

$$d(\mathbf{I}, \mathbf{U}) = \frac{\sum_{i=1}^{a} \|\mathbf{I}_i - \mathbf{U}_i\|}{a}, \tag{5.4}$$

where $\mathbf{I}_i$ and $\mathbf{U}_i$ represent the i-th row of matrix $\mathbf{I}$ and $\mathbf{U}$, respectively. In other words, $d(\mathbf{I}, \mathbf{U})$ is the average distance between each row of the matrices, where the distances are computed through the euclidean norm.

Since every single row of the input matrix contains a signal that must be compared only with its correspondent signal in the matrix of every node, in this extended version of the GWR algorithm, all the rows of every matrix are independent from the others. Finally, the mean is taken in order to have a single scalar value representing all the distances between the considered signals.

5.3.1 GWR Parameters

In order to achieve the best possible results, several parameters can be adjusted and modified according to the input signals. These parameters are:

- activity threshold a_t and firing counter threshold h_t that are related to the insertion of a new node in the network.
- e_b and e_n that controls the learning rates of the best node and of its neighbours.
- h_0, $S(t)$, α_b α_n, τ_b and τ_n that are tuning parameters of the firing counter $h(t)$ of each node.

In this work, the parameters that regulate the firing counter are set like in [39], i.e: $h_0 = 1$, $S(t) = 1$, $\alpha_b = 1.05$, $\alpha_n = 1.05$, $\tau_b = 3.33$, $\tau_n = 14.3$. The best results are then achieved with $a_t = 0.2$, $h_t = 0.245$, $e_b = 0.3$ and $e_n = 0.006$. The activity and the firing counter curves are represented in Figure 5.12. It can be seen that the firing counter of the winner becomes smaller faster than those of its neighbours. In particular, the threshold h_t is set so that if a node has fired five times then it is considered to be trained. Moreover, we can see that if the value of the threshold a_t is very close to 1, then more nodes are produced. Instead,

for lower values of a_t fewer nodes are added since a higher difference between the reference input and any other node is required in order to extend the network.

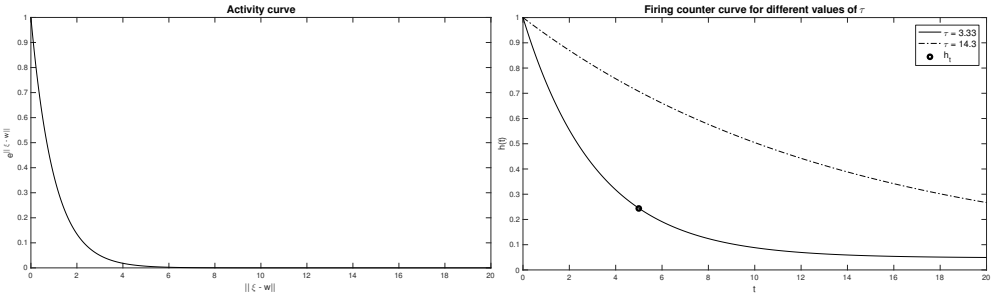


Figure 5.12: Activity curve and firing counter curve with different values of τ.

5.3.2 Pattern Extraction

Once all the input matrices associated to the extracted gait cycles are processed by the clustering algorithm, a network of connected nodes is created. Every node represents a cluster of gait cycles and, in particular, the matrices associated with every existing node in the network contain useful information about the associated cluster. Moreover, each node is considered a valid cluster if it has been trained at least five times during the training phase.

Every row of the matrix associated with a given node in the network is a meaningful pattern representing a subset of the input gait cycles created taking into account the learning rates and the firing counters of all the input samples associated with that cluster (see Equation 4.18). In particular, for every node, we have three patterns representing the acceleration signal in the cartesian coordinate system and three patterns representing the rotation in the three axes. An example of y-axis accelerometer pattern represented together with all the gait cycles assigned to the reference cluster can be seen in Figure 5.13.

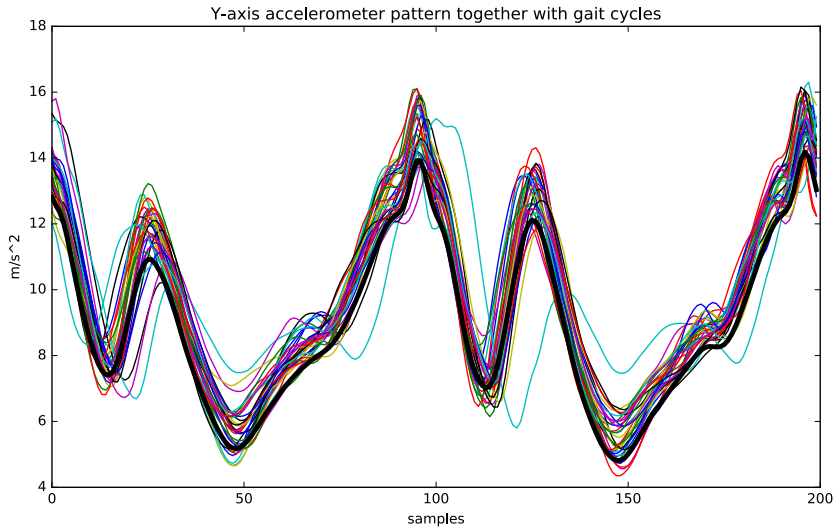


Figure 5.13: Example of y-axis acceleration pattern (bold black line) together with all the reference y-axis accelerometer gait cycles.

5.4 Gait Parameters Extraction

Once all the gait cycles are processed by the GWR algorithm, a certain number of gait parameters charaterizing the particular cluster to which they belong can be extracted. From the identification of the IC and FC instants, in fact, stride and subsequently stance and swing time (see Figure 5.6) of each gait cycle can be computed as:

$$Stance\ time = FC(i+1) - IC(i) \tag{5.5}$$

$$Stride\ time = IC(i+2) - IC(i) \tag{5.6}$$

$$Swing\ time = Stride\ time - Stance\ time \tag{5.7}$$

Therefore an average value of these parameters can be computed for each cluster.

Other two important gait parameters that can be estimated are the step length and the step velocity. They are computed using the upward and downward movement of the trunk. Assuming a compass gait type, Center of Mass (CoM) move-

ments in the sagittal plane follow a circular trajectory during each single support phase. In this inverted pendulum model, changes in height of the CoM depend on the step length [91]. Thus, when changes in height are known, the step length can be predicted from the geometrical characteristics as follows:

$$Step\ length = 2\sqrt{2lh - h^2}. \tag{5.8}$$

In this equation h is equal to the change in height of the CoM and l is equal to the pendulum length. Changes in the vertical position are calculated by a double integration of the vertical component $\mathbf{a}_y$ of the accelerometer data. To avoid integration drift, data are high-pass filtered with a fourth-order zero-lag Butterworth filter at 0.1 Hz. The amplitude in vertical position h is determined as the difference between the highest and lowest position during a step cycle. Leg length is taken as pendulum length l.

Once the step length is estimated, the average step velocity can be retrieved by the simple ratio between distance and time:

$$Step\ velocity = Step\ length/Stride\ time. \tag{5.9}$$

6
Activity Logger Video

Data collection and video recording are performed with an ad-hoc developed Android application, called *Activity Logger Video*, which is used to set up the acquisition parameters, give basic information about the user, collect data from the built-in sensors of the smartphone, record a reference video from the rear-facing camera, save all acquisitions data in non-volatile memory and, possibly, send them to an FTP server.

This application is an extension of the *Activity Logger* Android application developed by Matteo Gadaleta and Michele Rossi in a project aimed at recognizing a target user from his/her way of walking [3]. In their work, only accelerometer and gyroscope data coming from the embedded sensor of a commercial smartphone are used and thus some changes must be introduced in order to adapt this application to this work. First of all, the possibility of recording a video from the rear-facing camera has to be added. Moreover, in the original version of the application, the collected inertial data are stored in a temporary three-dimensional buffer and, when the acquisition ends, the buffer is read, the data of each sensor are saved in a text file and then stored in a directory of the smartphone file system. However, since this process can take several seconds even if the acquisition time is short, I introduce a different mechanism for saving the data: every time an acquisition sample is available from any sensor, it is immediately written in the correspondent file without any additional temporary buffer. This process drasti-

cally reduces the saving time making it almost independent from the acquisition time span.

6.1 Android Programming

Android applications are usually developed in the Java language using the Android Software Development Kit. Once developed, Android applications can be easily packaged and sold out through different stores like Google Play, Opera Mobile store or Amazon Appstore.

6.1.1 Android Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown in Figure 6.1 [92].

- *Linux Kernel.* At the bottom of the layers is Linux 3.6, with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display, etc.

- *Libraries.* On top of the Linux kernel there is a set of libraries including open source Web browser engine WebKit, SQLite database, libraries to play and record audio and video and SSL libraries responsible for Internet security, etc.

- *Android Runtime.* This third section of the architecture provides a key component called *Dalvik Virtual Machine* which is a kind of Java Virtual Machine specially designed and optimized for Android. The Dalvik Virtual Machine makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. It also enables every Android application to run in its own process.

- *Application Framework.* The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

- *Applications.* Android application layers is at the top. Developers write applications to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games, etc.

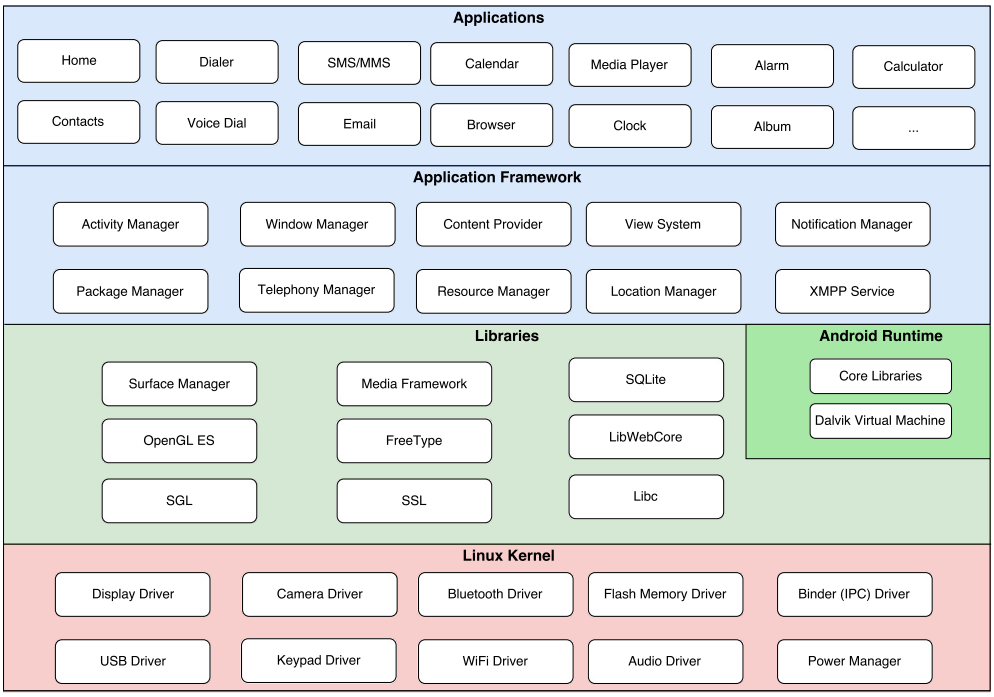


Figure 6.1: Android architecture diagram.

6.1.2 Application Components

Application components are the essential building blocks of an Android application. These components are defined in the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact. There are four main components that can be used within an Android application: Activities, Services, Broadcast Receivers and Content Providers.

- *Activities.* An activity represents a single screen with a user interface. In short, activities performs actions on the screen. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

- *Services.* A service is a component that runs in the background to perform long-running operations. For example, in the Activity Logger Video (the application developed for this thesis), a service starting the video recording is launched when the start button is tapped.

- *Broadcast Receivers.* Broadcast receivers simply respond to broadcast messages from other applications or from the system. Applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device.

- *Content Providers.* A content provider component supplies data from one application to the others on request. Such requests are handled by the methods of the *ContentResolver* class. Data may be stored in the file system, in the database or somewhere else entirely.

6.2 Activity Logger Video

Activity Logger Video requires a minimum API Level of 8 and requests permission for audio and video recording, accessing the camera, the internet and using external storage. All the required permissions together with the main components of the applications are detailed in the *AndroidManifest.xml* file (Listing 6.1).

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.signet.activityloggervideo" >
   <uses-sdk android:minSdkVersion="8" />
   <uses-permission android:name="android.permission.RECORD_VIDEO" />
   <uses-permission android:name="android.permission.RECORD_AUDIO" />
   <uses-permission android:name="android.permission.CAMERA" />
   <uses-feature android:name="android.hardware.camera" />
   <uses-permission
      android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.WAKE_LOCK" />
   <uses-permission android:name="android.permission.GET_ACCOUNTS" />
   <uses-permission android:name="android.permission.INTERNET" />
   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name=".MainActivity"
         android:label="@string/app_name"
         android:screenOrientation="portrait">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER"
               />
         </intent-filter>
      </activity>
      <service
         android:name="com.signet.activityloggervideo.RecorderService"
         />
   </application>
</manifest>
```

Listing 6.1: AndroidManifest.xml file of the application Activity Logger Video.

The application is composed of a main activity that defines all the functions related to the sensor acquisitions, data savings and data uploading. Moreover, when the camera switch is enabled, a service controlling the video recording is started. As can be seen from Figure 6.2, in the home screen the user can personalize the acquisition by entering his/her name, the activity he's going to perform and can add some additional information in the "Note" section. The acquisition can be delayed by a given interval in order to let the user fix the smartphone in the support. Moreover, it can be stopped manually by pressing the stop button or automatically by setting a predefined acquisition time. At the bottom of the home screen, in the video preview section, the video can be analyzed while it's being recorded. In the menu placed on the top-right corner of the home screen the user can delete all the acquisition files stored in the application folder or he can send all the data to the FTP server. In order to control the connection to the server, a class called *send_file_FTP* has been developed. It extends the *AsyncTask* class that enables the proper use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and handlers. At first, in the *send_file_FTP* class, four different strings are defined as is Listing 6.2.

```
String server = "*********";
String user = "*********";
String pass = "*********";
String serverRoad = "*********";
```

Listing 6.2: Strings identifying the connection parameters in the send_file_FTP class.

These strings define, respectively, the FTP server, the user name, the password and the path in which all the acquired data are going going to be saved in the server. Moreover, a protected method *doInBackground* defined in this class is called by the system to perform actions in the worker thread. Also, the system delivers to this method the same parameters given to *AsyncTask.execute()*. This method initially creates a *FTPClient* object (see Listing 6.3), and then tries to connect the the target server using the credentials defined above.

```java
protected Boolean doInBackground(String... file_path){
        String filename =
            file_path[0].substring(file_path[0].lastIndexOf("/") +
            1);
        try{
            FTPClient ftpClient = new FTPClient();
            ftpClient.connect(server);
            ftpClient.login(user, pass);
            ftpClient.enterLocalPassiveMode();
            ftpClient.setFileType(FTP.BINARY_FILE_TYPE);
            ftpClient.changeWorkingDirectory(serverRoad);
            FileInputStream file = new FileInputStream(new
                File(file_path[0]));
            if (!ftpClient.storeFile(filename, file)){
                file.close();
                ftpClient.logout();
                ftpClient.disconnect();
                return false;
            }
            file.close();
            ftpClient.logout();
            ftpClient.disconnect();
            return true;
        }
        catch (Exception e){
            e.printStackTrace();
            return false;
        }
    }
```

Listing 6.3: Protected method doInBackground.

The results of the connections and of the uploading procedures are handled by another protected method, called *onPostExecute*. This method takes as input a boolean value representing the outcome of the previous operations and printed out a message to the user reporting whether they have been successful or not. Once all the saved data have been sent to the server, thy are permanently deleted from the smartphone. The code implementing this method can be seen in Listing 6.4.

```
protected void onPostExecute(Boolean result){
        progress_dialog.dismiss();
        if (result){
           show_only_ok("Data sended.");
           delete_all_data(dir);
        }
        else{
           show_only_ok("ERROR! Data NOT sended.");
        }
     }
```

Listing 6.4: Protected method onPostExecute.

In order not to have any aliasing problem due to a too low sampling rate, data are acquired from the built-in sensors at highest possible sampling rate: this can be achieved with the constant *SensorManager.SENSOR_DELAY_FASTEST*. Android, in fact, let you tune the sensors' sampling rate. The possible choices are:

- *SensorManager.SENSOR_DELAY_NORMAL*: rate suitable for screen orientation changes (set by default).
- *SensorManager.SENSOR_DELAY_UI*: rate suitable for the user interface.
- *SensorManager.SENSOR_DELAY_GAME*: rate suitable for games.
- *SensorManager.SENSOR_DELAY_FASTEST*: get sensor data as fast as possible.

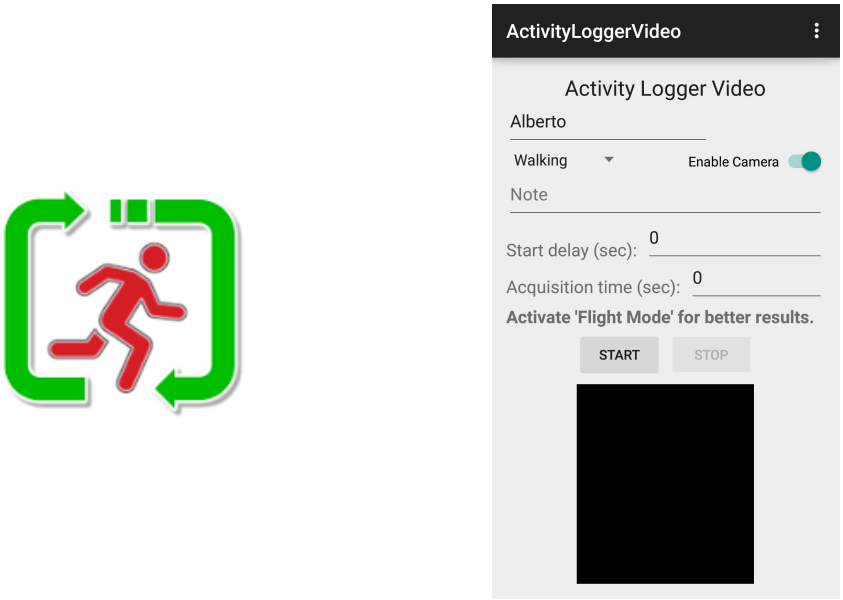


Figure 6.2: Activity Logger Video icon and home screen.

Since the acquisition time may be quite long, a Wakelock is needed in order to let the CPU running and prevent the screen from dimming during the acquisition phase. Wakelocks, in fact, are power manager system features that are able to control the power state of the host device. If not well-managed, however, they have a dramatic influence on the battery drain associated with a given application. Wakelocks are defined, triggered and released as follows:

```
powerManager = (PowerManager) getSystemService(POWER_SERVICE);
// Wakelock definition
wakeLock =
  powerManager.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
  "Acquisition_Wakelock");
wakeLock.acquire(); // Wakelock start
wakeLock.release(); // Wakelock release
```

Listing 6.5: Wakelock definition, triggering and release.

6.2.1 Video Frames / Sensor Data Synchronization

One of the main problems of working with a combination of video frames and data coming from the built-in sensors of the smartphone is how to synchronize them. Sensors and camera, in fact, have different acquisition rate: sensors data are acquired at around 100/200 Hz (depending on the smartphone model) while modern smartphones' cameras have a frame rate of 20/30 fps, depending on the video resolution. Moreover, as discussed above, sensors' sampling rate is not always constant during the acquisition. Video frames and sensor data synchronization, however, is very important in this kind of applications since events that are detected in the inertial signals coming from the sensors must also be analyzed in the signals extracted from the reference video. This can be achieved only when the two kind signals are well-synchronized.

Sensor samples alone can be easily synchronized by looking at the correspondent *SensorEvent.timestamp*. In fact, every time a new sample is available from any sensor, a timestamps measured in nanoseconds is returned. The method that allows to collect data coming from the sensor is *onSensorChanged*. It is defined in the *SensorEventListener* Android interface and it is automatically called by the system every time a new sensor event is detected. An example of how the data coming from the sensors can be managed with this method is shown in Listing 6.6.

```
@Override
public final void onSensorChanged(SensorEvent event){
  long sensorTimens = event.timestamp;
  Float[] temp = {event.values[0], event.values[1], event.values[2]};
  switch (event.sensor.getType()) { // check which sensor produced
      the data sample
  case Sensor.TYPE_ACCELEROMETER:
    write_file(temp, event.timestamp, fos_acc); // write data and
        timestamp in the correspondent file
    break;
  case Sensor.TYPE_LINEAR_ACCELERATION:
    write_file(temp, event.timestamp, fos_acclin);
    break;
  case Sensor.TYPE_GYROSCOPE:
```

```
    write_file(temp, event.timestamp, fos_gyro);
    break;
  case Sensor.TYPE_MAGNETIC_FIELD:
    write_file(temp, event.timestamp, fos_magn);
    break;
  case Sensor.TYPE_GRAVITY:
    write_file(temp, event.timestamp, fos_grav);
    break;
  case Sensor.TYPE_ROTATION_VECTOR:
    write_file(temp, event.timestamp, fos_rotvec);
    break;
          }
 }
```

Listing 6.6: Example of OnSensorChanged method implementation.

With this approach, every time a new sample is available, the reference timestamp for the three axes is saved in the *sensorTimens* variable and then written in a file associated to the correspondent sensor. The sensor that produced the sample can be easily identified with analyzing the *event.sensor.getType* output.

As for the camera, however, Android does not provide any method that allows for a frame by frame analysis and, consequently, it is not possible to have an exact reference timestamp for every frame. In order to cope with this, I used the *System.nanotime* method and the *MediaMetadataRetriever* object to estimate the exact instant in which the video recording is started and the video length. With these two additional information and given the frame rate, the number of video frame can be easily computed as

$$n_{frames} = video\_length \cdot frame\_rate \cdot 10^3, \tag{6.1}$$

where frame rate is expressed in frame per second (fps) and $video\_length$ is expressed in milliseconds. Moreover, starting from the initial recording instant, a timestamp can be assigned to every frame. Let $timestamp_n$ denote the timestamp of the n-th frame and let $start\_rec$ be initial recording instant, we have that:

$$timestamp_n = start\_rec + n \cdot \frac{1}{frame\_rate} \cdot 10^{-9}, \tag{6.2}$$

where *start_rec* is measured in nanoseconds. In the computation of the frame timestamps I assume that the camera frame rate is almost constant. The reference code for the video information and frame timestamp retrieval can bee seen in Listing 6.7.

```
 // Retrieve video information
MediaMetadataRetriever retriever = new MediaMetadataRetriever();
retriever.setDataSource(RecorderService.this, Uri.fromFile(new
    File(Environment.getExternalStorageDirectory().getPath() +
    "/ActivityLoggerVideo/" + video_path + "/" + video_path +
    ".mp4")));
video_length = Long.parseLong(retriever.extractMetadata(
       MediaMetadataRetriever.METADATA_KEY_DURATION));

n_frames = Math.round(video_length / ((1 / fps) * 1e3));

// compute frame timestaps asssuming uniform sampling frequency
frame_timestamps = new long[(int)n_frames];
for(int i = 0; i <= frame_timestamps.length-1; i++)
  frame_timestamps[i] = (long) ((start_rec) + i * (1/fps) * 1e9);

writeFrameTimestamps(); // write timestamp in a file
writeVideoInfo(); // duration and number of video frames are written
    in a file
```

Listing 6.7: Procedure for retrieving video information and frame timestamps.

At this point, even if every frame has an assigned timestamp, sensors data and frames timestamps must be aligned. In fact, they are both computed in nanoseconds but with respect to different initial instants. According to the Android API documentation, *SensorEvent.timestamp* is the time at which the event happened since the operating system started. On the other hand, the value returned from *System.nanotime* represents the nanoseconds since some fixed but arbitrary origin time. Then, in order to align these values, at every new acquisition, the offset between the two measurements is computed and then added to all the SensorEvent.timestamp values. With this procedure a final software synchronization between video frames and sensor data can be achieved.

In Listing 6.8 is presented the code, included in the *OnSensorChanged* method, that implements the alignment between data and frame timestamp. At the beginning of every new acquisition, two different arrays called *Systs* and *Evts* are defined. They are going to contain a certain number of *System.nanoTime* and *event.timestamp* entries. After the filling procedure, the last ten samples of each array are used to compute the average offset value between the two time references. In the computation of the offset, I decided to utilize only the last samples since, at the beginning of each acquisition, the timestamps returned from the two methods are not stable and precise because of the computational load of the application.

```
if(!done){ // do this exactly once at every new acquisition
  Systs[counter] = System.nanoTime();
  Evts[counter] = event.timestamp;
  counter ++;
  if(counter == Systs.length) {
    int samples = 10; // compute average value using last 10 samples
    for(int i = Systs.length - samples; i <= Systs.length - 1; i++)
      startoffset = startoffset + (Systs[i] - Evts[i]);
    startoffset = startoffset / samples;
    done = true;
    counter = 0;
  }
}
```

Listing 6.8: Procedure utilized to align data and frame timestamps.

7 Results

In this chapter, I present the analysis and the experiments carried out to set up the the proposed gait analysis system. In particular, I discuss and detail an example of the GWR algorithm applied to the gathered data describing all the main iterations and steps involved.

Besides, I show that the combination of accelerometer and signals extracted from the video can lead to better clustering results if compared to the "classical" approach that uses accelerometer data together with gyroscope ones. The outcomes of the clustering procedure are also analyzed in a user identification scenario, where the proposed algorithm tries to distinguish between different users from their walking style. Finally, the parameters extracted from the different walking acquisitions are presented and discussed in detail.

7.1 GWR Clustering

As discussed in Section 4.4, the GWR is an unsupervised growing network-based incremental algorithm that can be used to discover topological structures and clusters in the input data. I recall that the parameters regulating the firing counter of each node in the network are set as in [39]. With respect to the activity threshold $a_t \in [0, 1]$, a value of $a_t = 1$ means that at every iteration a new node is added (if also the firing counter constraint is satisfied) and $a_t = 0$

means that no new node are ever going to be added to the network. In this work, a_t, the firing counter threshold h_t and the learning rates e_b and e_n have been tuned according to the final clustering results. In fact, since the number of expected clusters is known a priori and since I know to which cluster each gait cycle belongs to, the best values for a_t and h_t are the ones maximizing the correct matches. According to this procedure, the assigned values are $a_t = 0.2$, $h_t = 0.245$, $e_b = 0.3$ and $e_n = 0.006$. With this value of h_t, as can be seen in Figure 7.1, a given node is considered to be trained after having fired exactly five times.

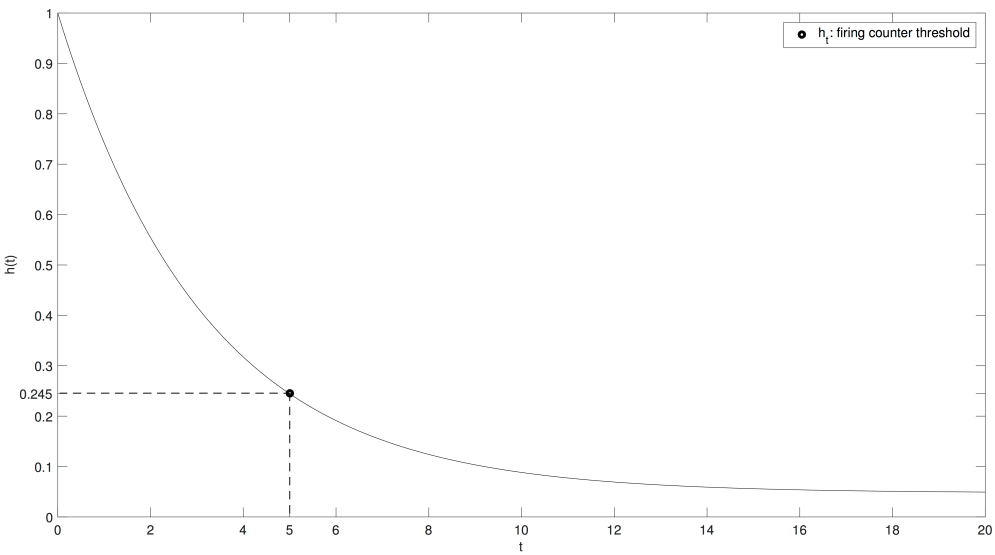


Figure 7.1: Firing counter curve and firing counter threshold h_t.

In Figure 7.2 it is shown an example of the evolution of the GWR network using the accelerometer and video signals gathered from the four different walking acquisitions of one of the volunteer. These data are then put together, processed, rearranged in matrix form, and given as input to the clustering algorithm.

In this example, the maximum allowed age for the edges of the graph is set to $a_{max} = 5$.

The algorithm starts with the initialization phase (ITERATION 1) where two nodes are randomly created and connected with an edge that has age equal to zero.

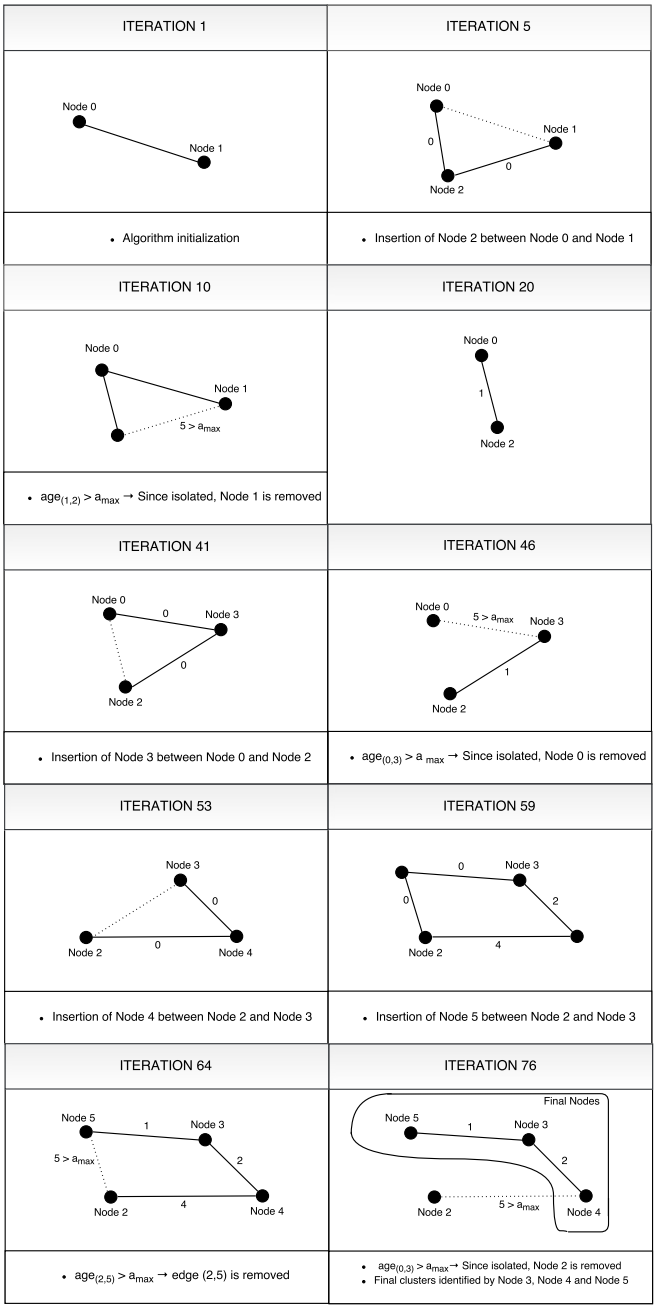


Figure 7.2: Example of the GWR network evolution using accelerometer and video signals.

In ITERATION 5, the activity and the firing counter values of the best node (Node 0) are smaller than the correspondent thresholds. Consequently, Node 2 is added to the network between Node 0 and Node 1. This new node will have an associated weight matrix $\mathbf{w}_2 = (\mathbf{w}_0 + \boldsymbol{\xi})/2$, where $\mathbf{w}_0$ is the weight matrix associated to Node 0 and $\boldsymbol{\xi}$ is the input data matrix at the fifth iteration. Moreover, the connection between Node 0 and Node 1 is removed. After five more algorithm iterations, the age of the edge connecting Node 1 and Node 2 exceeds a_{max} and consequently it is deleted. At this point, Node 1 is isolated from the network and it is permanently discarded. From ITERATION 10 to ITERATION 40, the network setup does not change and the two nodes keep adapting their weight matrices to the input data. In the successive iterations the network grows adding Node 3, Node 4 and removing the starting Node 0. At ITERATION 59, Node 5 enters the network between Node 2 and Node 3. This insertion is different from the others since it occurs between two nodes that were not previously connected. New node insertion, in fact, always happens between the two best matching nodes but it does not imply their connection. In the last iterations, the connections of Node 2 are deleted since they booth exceed a_{max}. After the removal of Node 2 the algorithm ends with a final set of nodes composed by Node 3, Node 4 and Node 5.

These nodes identify the final clusters detected by the algorithm in the input data. Each input sample, in fact, is assigned to one of these three clusters. In particular, it can be noticed that the GWR network is built by creating and deleting different nodes and only the ones that lead to the best representation of the input data are kept.

The weight matrices associated with the final nodes contain six different patterns that define the cluster they represent. These patterns represent a sort of weighted average of all the input samples that have been assigned to a given cluster. In particular, every patter is also strongly influenced by the learning rate parameters and by the firing counter of each node.

The reference patterns associated to the example discussed above can be seen in Figure 7.3. These six patterns are related to the three components of the accelerometer and of the signals extracted from the videos. The ones associated to the "cluster0" represents the regular-walking-style cluster while the other two identify the walks with a right and left limp, respectively.

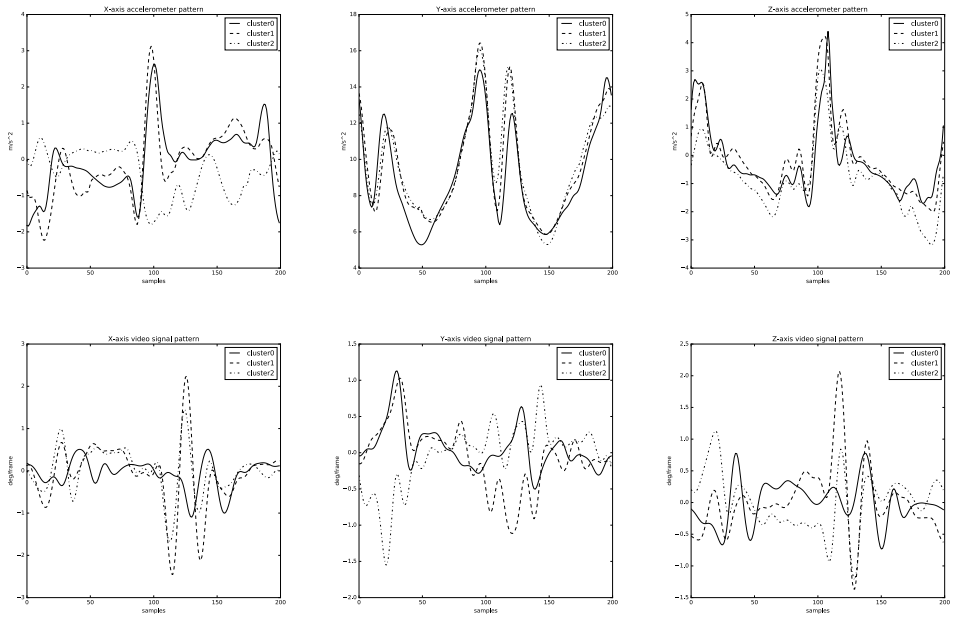


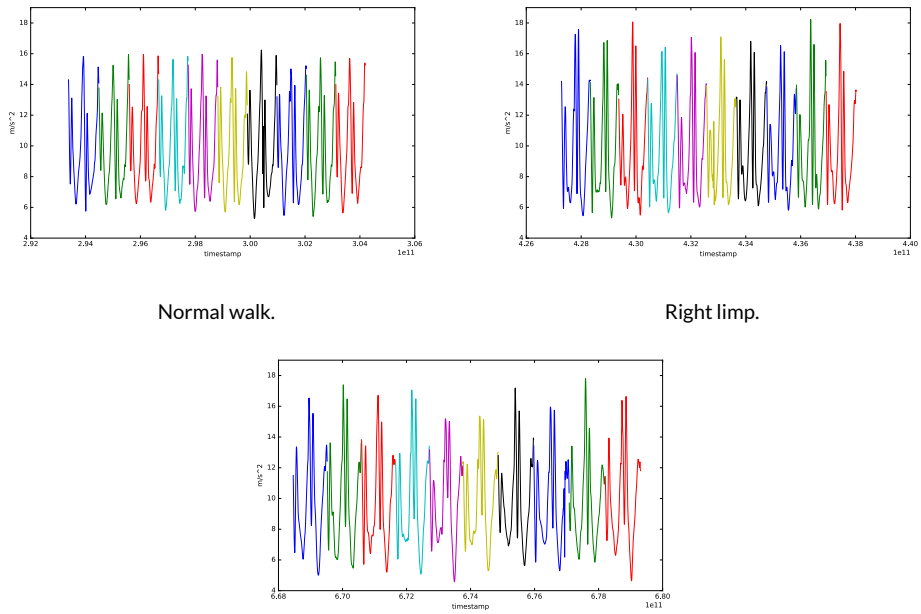
Figure 7.3: Reference patterns associated to the clusters discovered by GWR algorithm using accelerometer and video signals. "Cluster0" is related to the regular walking style, "cluster1" is related to the walk with a right limp and "cluster2" is related to the walk with a left limp.

As expected, it can be seen that the curves associated to "cluster0" are almost symmetric. This emphasize the fact that, during a regular walk of a healthy person, the signals extracted from the left and right step are almost the same. For what concerns the other two clusters, not surprisingly, this symmetry is absent, especially in the patterns extracted from the video signals.

7.2 Performance Analysis of the GWR Algorithm with Different Combinations of Input Signals

In this section I present and evaluate the results obtained from the GWR algorithm focusing upon the benefits that the signals extracted from the video can bring if combined with the accelerometer ones. As already discussed, in fact, in most gait analysis or human identification works, the classical signals included in the study are the ones coming from accelerometer and gyroscope.

In Figure 7.4 are reported three examples of y-axis accelerometer signals corresponding to the three considered walking styles. The gait cycles are extracted with the techniques discussed in Chapter 5 and are represented by different colors in the figures.



Normal walk.

Right limp.

Left limp.

Figure 7.4: Y-axis acceleration signals for the three different walking styles. Different gait cycles are represented by different colors.

It can be seen that there are no significant differences between the analyzed

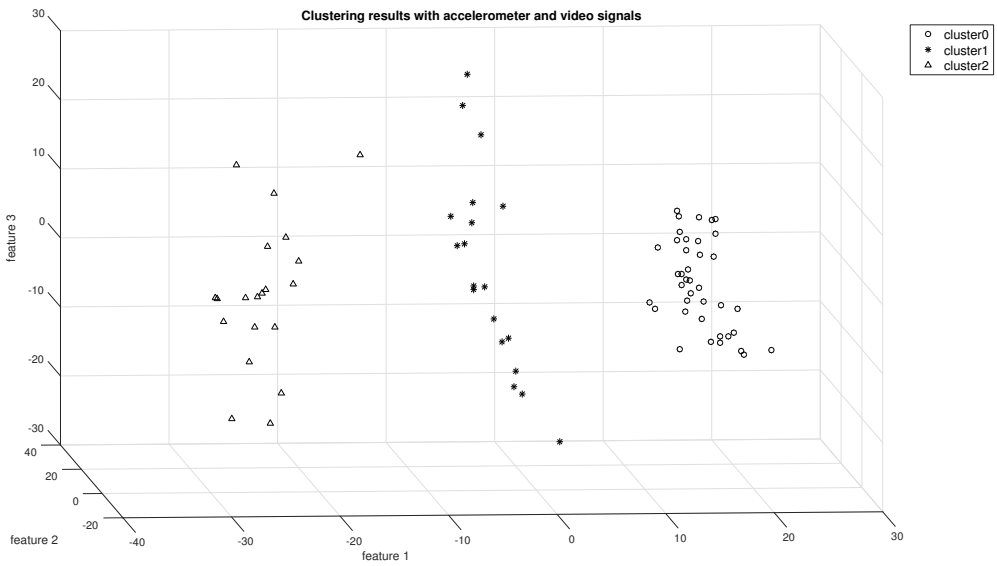


Figure 7.5: Clustering results obtained with accelerometer and video signals.

signals. In fact, the waveform and the range values are almost the same for all the walks. This fact underlines that differentiating the walking acquisitions finding meaningful clusters that effectively represent the input space is not a trivial problem.

Figure 7.5 shows how the input gait samples of a given user are clustered by the GWR algorithm using the accelerometer components together with the signals extracted from the reference videos. Since every gait sample is represented by a 6×200 matrix, to visualize them a dimensionality reduction technique has to be applied. The tool I utilized is Principal Component Analysis (PCA) [61].

PCA is a dimensionality-reduction algorithm that can be used to reduce a large set of variables to a small set that still contains most of the information of the large one. It is a mathematical procedure that transforms a number of correlated features into a number of uncorrelated ones, called principal components, where each pair of new distinct features has 0 covariance. Features are ordered with respect to how much of the variance of the data each component captures. The first feature captures as much of the variance of the data as possible and, subject to the orthogonality requirement, each successive attribute captures as much of

the remaining variance as possible.

Every input matrix is vectorized obtaining a reference input vector of 1200 samples that is given as input to the PCA algorithm. In order to correctly visualize the results, the number of PCA features is set to three. Obviously, introducing a dimensionality reduction in the data, leads to an information loss. The extracted features, however, are still able to successfully describe the data since the clusters detected by the GWR algorithm can be easily identified even with visual inspection.

Using the accelerometer and video signals, as reported in Table 7.1, the number clusters detected by the algorithm matches the number of expected ones: a cluster that represents the normal walk, another representing the walk with a limp on the right leg and the third related to the walk with a limp on the left leg.

Expected number of clusters	Number of found clusters
3	3

Table 7.1: Expected and found number of clusters using accelerometer and video signals.

However, even if the number of detected and expected clusters is the same, it does not mean that the clustering procedure has completely succeed. It could be, in fact, that some input samples may be assigned to a wrong cluster. So, in order to compute the accuracy results, the cluster assigned to each input sample must be compared with the ground truth. In this case, the clustering results perfectly match the ground truth for every of the five users considered in this study. The final accuracy reached with accelerometer and video signals is then 100%.

Figure 7.6 shows the clustering results obtained with the accelerometer and gyroscope data. As underlined by Table 7.2, in this case the GWR algorithm is not able to detect all the expected clusters.

Expected number of clusters	Number of found clusters
3	2

Table 7.2: Expected and found number of clusters using accelerometer and gyroscope signals.

From the plot below, in fact, only two well-separated clusters can be observed. By comparing the clustering results with the ground truth, it can be seen that the

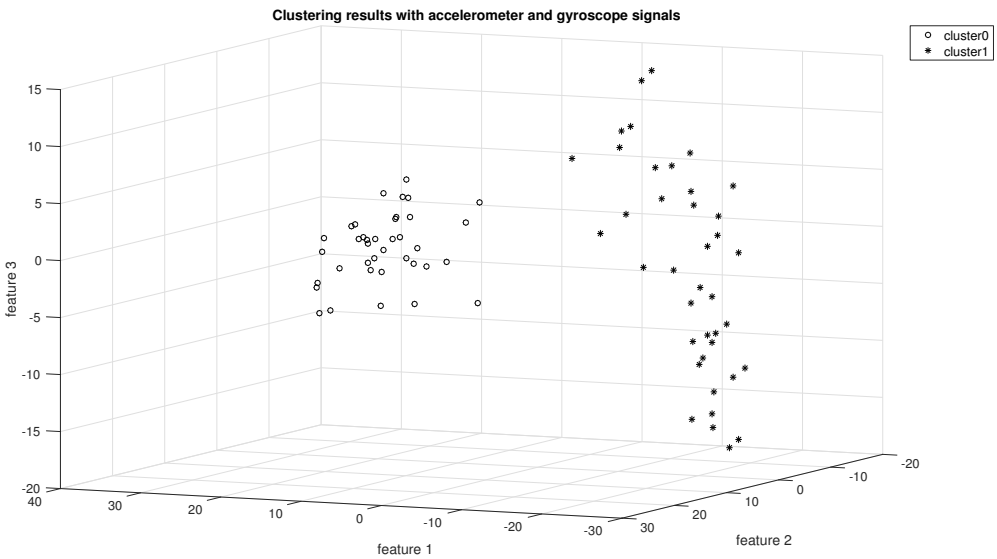


Figure 7.6: Clustering results obtained with accelerometer and gyroscope signals.

input gait samples related to the normal walk are all correctly assigned to "cluster0". However, the left and right limp clusters are not differentiated by the algorithm since the associated samples are all assigned to the same "cluster1".

From these results it comes out that the approach that includes accelerometer data and signals extracted from input videos can describe the input data in a more precise manner, leading the clustering algorithm to discover all the expected clusters without any classification error. On the other hand, the classical approach that sees the accelerometer data combined with the gyroscope ones leads to the detection of only two clusters, the one related to the normal walk and the other that includes all the samples coming from a generic walk with a limp.

The clustering performance of the proposed system is also evaluated in a different scenario. Considering only the acquisitions coming from the normal walks, it is analyzed whether the algorithm is able to recognize which gait samples belongs to a certain user. Also in this case, the results coming from the combination of accelerometer data and video signals are compared against the ones related to the combination of accelerometer and gyroscope signals. To the best of my knowledge, an unsupervised human identification approach similar to the one proposed

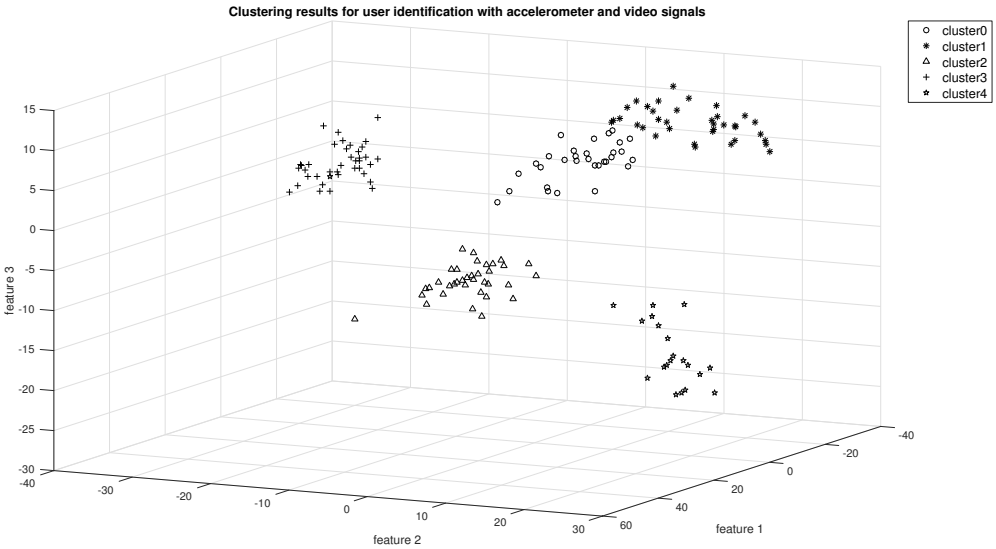


Figure 7.7: Clustering results obtained in the user identification scenario with accelerometer and video signals.

in this work has never been investigated in the literature.

Figure 7.7 presents the results achieved utilizing the accelerometer and video signals. The parameters of the GWR algorithm are set as described in Section 7.1 and, also in this case, PCA is performed in order to visualize the input data. As reported in Table 7.3, all the five expected clusters corresponding to the five users are successfully identified by the algorithm. Besides, even if "cluster0" and "cluster1" are so close to each other, the algorithm is still able to split them. Also in this user identification scenario, the accuracy of the system, assessed comparing the obtained results with the ground truth, shows great performance since all the input samples are correctly assigned to the correspondent cluster.

Expected number of clusters	Number of found clusters
5	5

Table 7.3: Expected and found number of clusters using accelerometer and video signals in the user identification scenario.

As before, also in this case the results are investigated when the video signals are replaced by the gyroscope data. Figure 7.8 shows that, even in the user identification scenario, the system is not able to perfectly characterize the input data

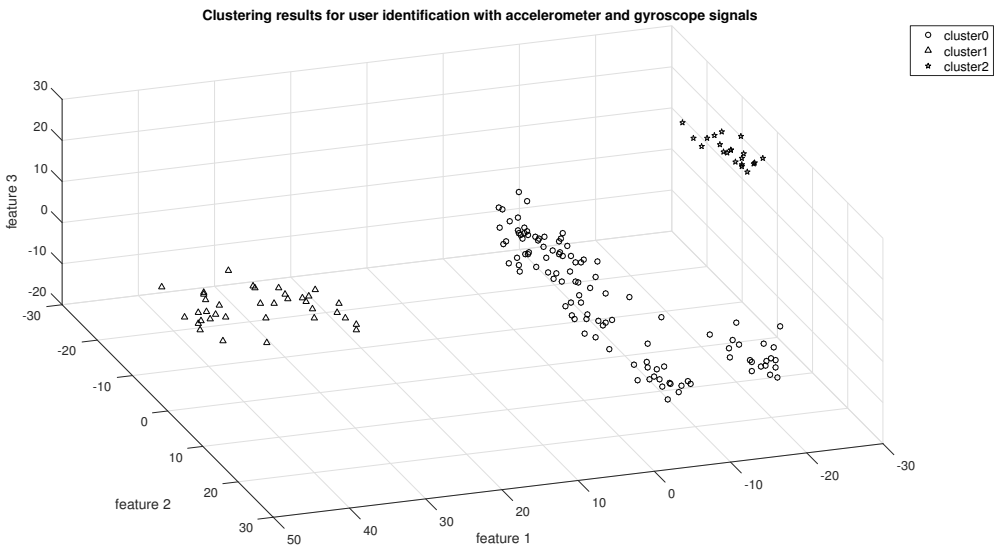


Figure 7.8: Clustering results obtained in the user identification scenario with accelerometer and gyroscope signals.

when the gyroscope is included in the computation. Even with visual inspection the data are not well-separated and only three main clusters can be recognized. The number of clusters identified by the algorithm, in fact, does not matches the number expected ones (see Table 7.4).

Expected number of clusters	Number of found clusters
5	3

Table 7.4: Expected and found number of clusters using accelerometer and gyroscope signals in the user identification scenario.

Analyzing the ground truth, it can be seen that the gait samples corresponding to the second and fourth user are correctly assigned to two different clusters (”cluster1” and ”cluster2”). On the other hand, the input samples corresponding to the first, third a fifth user are all misclassified and assigned to a unique ”cluster0”.

These results confirm that the approach including accelerometer and gyroscope data is overcome by the one including the signals extracted from the reference video. The combination of these signals, in fact, can effectively describe the input

in such a way that all the expected clusters can be identified by the algorithm in both scenarios.

7.3 Gait parameters

In this section I present the parameters estimated from the extracted gait samples. After each sample is assigned to a certain cluster, in fact, the stance and swing time values together with the step velocity are computed using the techniques described in Chapter 5.

I recall that the stance time is defined as the time in which the reference foot is in contact with the ground at the beginning of each gait cycle. The swing time, instead, is the time in which the reference foot swings forward between one episode of ground contact and the next one.

In the three tables below I report the average value and standard deviation of the parameters estimated from the input samples of every user. It can be seen that there are no substantial changes in the parameters value of User 1 and User 3 for the different walking styles. As for User 0, instead, it can be noticed a decrease of the swing time of around 9% in the case of walking with a limp acquisitions. In User 2 and User 4 limp parameters, there is a noticeable decrease in the step velocity value of around 23% and 18%, respectively. Moreover, as opposite to the results obtained for User 0, User 2 has an increased swing time value (around 10%) in the case of the walks with a limp.

User ID / Gait parameters	User 0	User 1	User 2	User 3	User 4
Stance time	0.44 ± 0.007 s	0.40 ± 0.007 s	0.42 ± 0.006 s	0.39 ± 0.01 s	0.39 ± 0.01 s
Swing time	0.73 ± 0.01 s	0.67 ± 0.01 s	0.70 ± 0.01 s	0.69 ± 0.02 s	0.62 ± 0.01 s
Step velocity	1.00 ± 0.06 m/s	1.08 ± 0.06 m/s	1.06 ± 0.03 m/s	0.92 ± 0.04 m/s	1.11 ± 0.04 m/s

Table 7.5: Gait parameters extracted from normal walking samples.

User ID / Gait parameters	User 0	User 1	User 2	User 3	User 4
Stance time	0.41 ± 0.01 s	0.41 ± 0.01 s	0.45 ± 0.01 s	0.37 ± 0.03 s	0.43 ± 0.04 s
Swing time	0.66 ± 0.014 s	0.67 ± 0.02 s	0.77 ± 0.02 s	0.67 ± 0.04 s	0.66 ± 0.03 s
Step velocity	1.11 ± 0.03 m/s	1.06 ± 0.03 m/s	0.8 ± 0.08 m/s	0.89 ± 0.06 m/s	0.92 ± 0.07 m/s

Table 7.6: Gait parameters extracted from right-limp walking samples.

User ID / Gait parameters	User 0	User 1	User 2	User 3	User 4
Stance time	0.42 ± 0.03 s	0.40 ± 0.01 s	0.43 ± 0.01 s	0.42 ± 0.02 s	0.43 ± 0.02 s
Swing time	0.67 ± 0.03 s	0.67 ± 0.01 s	0.79 ± 0.02 s	0.67 ± 0.03 s	0.72 ± 0.05 s
Step velocity	1.11 ± 0.09 m/s	1.12 ± 0.06 m/s	0.82 ± 0.08 m/s	0.83 ± 0.07 m/s	0.88 ± 0.06 m/s

Table 7.7: Gait parameters extracted from left-limp walking samples.

8
Conclusions and Future Work

In this thesis, I have presented a gait analysis system based on inertial and video signals acquired from a smartphone located in an ad-hoc made chest support during different walking sessions. In particular, the proposed system exploits accelerometer and gyroscope data together with signals extracted from a reference video to estimate useful gait parameters and identify meaningful clusters in the walking data. After a preprocessing and a cycle segmentation phase, an adapted version of the GRW unsupervised clustering algorithm is applied. This algorithm, in addition to identify clusters in the input data, returns a representative pattern for each of them.

The performance of the proposed clustering algorithm has been investigated in two different scenarios using different combinations of input signals. At the beginning, the system tries to distinguish between three different walking styles performed by a single user. Then, different walking samples coming from different users are used as input to detect whether the algorithm was able to assign each sample to the correspondent user.

Results showed that the combination of accelerometer and video signals performs much better than the combination of accelerometer and gyroscope data, leading to a perfect classification in both scenarios.

As future work, the proposed system could be exploited in researches aimed at studying the evolution of the Parkinson's syndrome, analyzing if particular

gait patterns occur, identifying the stage of the disease,etc.. Moreover, it could be included in a personal health counseling application able to find user-related walking patterns, track them over time and monitor monthly or weekly changes.

References

[1] A. Fusiello, *Visione Computazionale. Tecniche di Ricostruzione Tridimensionale.* Milano: Franco Angeli, 2013.

[2] J. Holmström, "Growing Neural Gas Experiments with GNG, GNG with Utility and Supervised GNG," Master's thesis, Uppsala University.

[3] M. Gadaleta and M. Rossi, "Idnet: Smartphone-based gait recognition with convolutional neural networks," *CoRR*, vol. abs/1606.03238, 2016. [Online]. Available: http://arxiv.org/abs/1606.03238

[4] N. Neverova, C. Wolf, G. Lacey, L. Fridman, D. Chandra, B. Barbello, and G. W. Taylor, "Learning human identity from motion patterns," *CoRR*, vol. abs/1511.03908, 2015. [Online]. Available: http://arxiv.org/abs/1511.03908

[5] S. J. Morris and J. A. Paradiso, "Shoe-integrated sensor system for wireless gait analysis and real-time therapeutic feedback," pp. 2468, 2469, October 2002.

[6] M. Shoaib, S. Bosch, O. D. Incel, H. Scholten, and P. J. M. Havinga, "Complex human activity recognition using smartphone and wrist-worn motion sensors," *sensors*, 2016. [Online]. Available: www.mdpi.com/journal/sensors

[7] O. D. Incel, M. Kose, and C. Ersoy, "A review and taxonomy of activity recognition on mobile phones," *Springer Science + Business Media New York*, 2013.

[8] Lockhart, P. J.W., W. T., and G.M., "Applications of mobile activity recognition," pp. 1054–1058, 2012.

[9] S. O. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Prentice Hall, 2008.

[10] M. Yuwono, B. Moulton, S. Su, B. G. Celler, and H. Nguyen, "Unsupervised machine learning method for improving the performance of ambulatory fall detection systems," *BioMedical Engineering OnLine*, 2012. [Online]. Available: http://www.biomedicalengineering-online.com/content/11/1/9

[11] B. Longsta, S. Reddy, and D. Estrin, "Improving activity classification for health applications on mobile devices using active and semi-supervised learning."

[12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[13] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*, 2nd ed. Springer, 2009. [Online]. Available: http://www-stat.stanford.edu/~tibs/ElemStatLearn/

[14] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system-a case study," DTIC Document, Tech. Rep., 2000.

[15] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2, 1995, pp. 1137–1145.

[16] T. Martinetz, "Competitive hebbian learning rule forms perfectly topology preserving maps," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN-93), Amsterdam*, S. Gielen and B. Kappen, Eds. Heidelberg: Springer, 1993, pp. 427–434.

[17] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[18] P. Siirtola and J. Röning, "User-independent human activity recognition using a mobile phone: Offline recognition vs. real-time on device recognition," in *Distributed Computing and Artificial Intelligence*. Springer, 2012, pp. 617–627.

[19] J. J. Guiry, P. van de Ven, and J. Nelson, "Classification techniques for smartphone based activity detection," in *Cybernetic Intelligent Systems (CIS), 2012 IEEE 11th International Conference on.* IEEE, 2012, pp. 154–158.

[20] J. J. Guiry, P. van de Ven, J. Nelson, L. Warmerdam, and H. Riper, "Activity recognition with smartphone support," *Medical engineering & physics*, vol. 36, no. 6, pp. 670–675, 2014.

[21] O. D. Incel, M. Kose, and C. Ersoy, "Online human activity recognition on smart phones," 2012.

[22] H.-H. Hsu, C.-T. Chu, Y. Zhou, and Z. Cheng, "Two-phase activity recognition with smartphone sensors," 2015.

[23] M. E., Papandrea, M., Lane, and Campbell, "Pocket, bag, hand, etc. automatically detecting phone context through discovery," 2010.

[24] Park, J.g., Patel, A., Curtis, D., Ledlie, J., and Teller, "Online pose classification and walking speed estimation using handheld devices," pp. 113–122, 2012.

[25] J. Guo, X. Zhou, Y. Sun, G. Ping, G. Zhao, and Z. Li, "Smartphone-based patients' activity recognition by using a self-learning scheme for medical monitoring," *Springer Science + Business Media*, 2016.

[26] M. O. Derawi, "Accelerometer-based gait analysis, a survey," *Nor Informasjonssikkerhetskonferanse NISK*, 2010.

[27] M. Murray, "Gait as total pattern of movement," *American Journal of Physical Medicine*, vol. 46, 1967.

[28] M. Murray, A. B. Drought, and R. C. Kory, "Walking patterns of normal men," *Journal of Bone ans Joint Surgery*, vol. 46, 1964.

[29] J. Perry, "History of the study of locomotion." [Online]. Available: http://www.clinicalgaitanalysis.com/history/modern.html

[30] H. M., L. H., and M.-N. R., "Test-retest reliability of trunk accelerometric gait analysis," 2004.

[31] S. Nishiguchi, M. Yamada, K. Nagai, S. Mori, Y. Kajiwara, T. Sonoda, K. Yoshimura, H. Yoshitomi, H. Ito, K. Okamoto, T. Ito, S. Muto, T. Ishihara, and T. Aoyama, "Reliability and validity of gait analysis by android-based smartphone," *Telemedicine and e-Health*, vol. 18, pp. 292–296, March 2012.

[32] J. Mantyjarvi, M. Lindholm, E. Vildjiounaite, S. Makela, and H. Ailisto, "Identifying users of portable devices from gait pattern with accelerometers," pp. 973–976, March 2005.

[33] M. Derawi and P. Bours, "Gait and activity recognition using commercial phones," vol. 39, pp. 137–144, November 2013.

[34] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

[35] F. Miao, Y. He, J. Liu, Y. Li, and I. Ayoola, "Identifying typical physical activity on smartphone with varying positions and orientations," 2015.

[36] S. D. Din, A. Godfrey, and L. Rochester, "Validation of an accelerometer to quantify a comprehensive battery of gait characteristics in healthy older adults and parkinson's disease: Toward clinical and at home use," 2015.

[37] J. McCamley, M. Donati, E. Grimpampi, and C. Mazza, "An enhanced estimate of initial contact and final contact instants of time using lower trunk inertial sensor data," pp. 316–318, 2012.

[38] B. Fritzke, "A growing neural fas network learns topologies," 1995.

[39] S. Marsland, J. Shapiro, and U. Nehmzow, "A self-organising network that grows when required," 2002.

[40] D. Trojaniello, A. Cereatti, and U. D. Croce, "Accuracy, sensitivity and robustness of five different methods for the estimation of gait temporal parameters using a single inertial sensor mounted on the lower trunk," pp. 487–492, 2014.

[41] S. Jiang, B. Zhang, G. Zou, and D. Wei, "The possibility of normal gait analysis based on a smart phone for healthcare," in *Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, August 2013.

[42] D. Qin and M.-C. Huang, "A smartphone based gait monitor system," 2015.

[43] J. G. Lim, "Unsupervised clustering for abnormality detection based on the tri-axial accelerometer," 2009.

[44] P. Gupta, G. Ramirez, D. Y. Lie, R. T. Dallas, E. Banister, and A. Dentino, "Mems-based sensing and algorithm development for fall detection and gait analysis," *MOEMS-MEMS*, 2010.

[45] J. Chen, K. Kwong, D. Chang, J. Luk, and R. Bajcsy, *Engineering in Medicine and Biology Society*, pp. 3551–3554, 2005.

[46] V. S. N. Selvabala and A. B. Ganesh, "Implementation of wireless sensor network based human fall detection system," *Procedia Engineering*, pp. 767–773, 2012.

[47] N. H. Chehade, P. Ozisik, J. Gomez, F. Ramos, and G. Pottie, "Detecting stumbles with a single accelerometer," pp. 6681–6686, 2012.

[48] M. El-Gohary, J. McNames, K. Chung, M. Aboy, A. Salarian, and F. Horak, "Continuous at-home monitoring of tremor in patients with parkinson disease," *Biosignal 2010: Analysis of Biomedical Signals and Images*, pp. 420–424, 2010.

[49] M. Yuwono, S. W. Su, B. D. Moulton, and H. T. Nguyen, "Gait episode identification based on wavelet feature clustering of spectrogram images," August 2012.

[50] M. Yuwono, A. M. A. Handojoseno, and H. T. Nguyen, "Optimization of head movement recognition using augmented radial basis function neural network," August 2011.

[51] Y.Tao and H. Building, "Building a visual tracking system for home-based rehabilitation," pp. 343–348, 2003.

[52] H. Charif and S. McKenna, "Activity summarisation and fall detection in a supportive home environment," 2004.

[53] J.Gao, H.Wactlar, and A.Hauptmann, "Combining motion segmentation with tracking for activity analysis," *AFG*, 2004.

[54] L. Wang, "Abnormal walking gait analysis using silhouette-masked flow histograms," 2006.

[55] S. Ullman, *High-level vision: Object recognition and visual cognition.* Boston, MA, USA: MIT Press, 1996.

[56] D. Marr, *Vision. A computational investigation into the human representation and processing of visual information.* San Francisco: Freeman, 1982.

[57] J. Y. Aloimonos and D. Shulman, *Integration of Visual Modules: An Extension of the Marr Paradigm.* San Diego, CA, USA: Academic Press Professional, Inc., 1989.

[58] O. Faugeras, *Three-dimensional Computer Vision: A Geometric Viewpoint.* Cambridge, MA, USA: MIT Press, 1993.

[59] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94

[60] Z. Zhang, "Eight-point algorithm," in *Computer Vision.* Springer, 2014, pp. 239–240.

[61] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[62] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[63] T. Hastie and W. Stuetzle, "Principal curves," *Journal of the American Statistical Association*, vol. 84, no. 406, pp. 502–516, 1989.

[64] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[65] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999. [Online]. Available: http://doi.acm.org/10.1145/331499.331504

[66] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley, 1990.

[67] B. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster analysis*, 5th ed. Wiley, 2011.

[68] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[69] Z. Huang, "Clustering large data sets with mixed numeric and categorical values," in *In The First Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 1997, pp. 21–34.

[70] P. J. Groenen and K. Jajuga, "Fuzzy clustering with squared minkowski distances," *Fuzzy Sets and Systems*, vol. 120, no. 2, pp. 227–237, 2001.

[71] S. Xiang, F. Nie, and C. Zhang, "Learning a mahalanobis distance metric for data clustering and classification," *Pattern Recognition*, vol. 41, no. 12, pp. 3600–3612, 2008.

[72] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: Fuzzy c-means algorithm," *Computers and Geoscience*, 1984.

[73] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[74] D. A. Keim and A. Hinneburg, "Clustering techniques for large data sets—from the past to the future," in *Tutorial Notes of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining, ser. KDD '99. New York, NY, USA: ACM, 1999, pp. 141–181. [Online]. Available: http://doi.acm.org/10.1145/312179.312189

[75] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," *SIGMOD Rec.*, vol. 27, no. 2, pp. 94–105, Jun. 1998. [Online]. Available: http://doi.acm.org/10.1145/276305.276314

[76] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[77] J. D. Banfield and A. E. Raftery, "Model-based gaussian and non-gaussian clustering," *Biometrics*, pp. 803–821, 1993.

[78] G. Celeux and G. Govaert, "Gaussian parsimonious clustering models," *Pattern Recognition*, vol. 28, no. 5, pp. 781–793, 1995.

[79] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika*, vol. 32, no. 3, pp. 241–254, 1967.

[80] G. Karypis, E.-H. Han, and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," *Computer*, vol. 32, no. 8, pp. 68–75, 1999.

[81] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and semi-supervised clustering: a brief survey," in *A Review of Machine Learning Techniques for Processing Multimedia Content*, 2004.

[82] S. Petrovic, "A comparison between the silhouette index and the davies-bouldin index in labelling ids clusters," in *Proceedings of the 11th Nordic Workshop of Secure IT Systems*, 2006, pp. 53–64.

[83] J. C. Bezdek and N. R. Pal, "Cluster validation with generalized dunn's indices," in *Artificial Neural Networks and Expert Systems, 1995. Proceedings., Second New Zealand International Two-Stream Conference on.* IEEE, 1995, pp. 190–193.

[84] X. L. Xie and G. Beni, "A validity measure for fuzzy clustering," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 13, no. 8, pp. 841–847, 1991.

[85] J. C. Stanley, "Computer simulation of a model of habituation," *Nature*, vol. 261, pp. 146–148, May 1976. [Online]. Available: http://dx.doi.org/10.1038/261146a0

[86] P. D. Welch, "The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on audio and electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[87] G. Bradski, *Dr. Dobb's Journal of Software Tools*.

[88] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001. [Online]. Available: http://www.scipy.org/

[89] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[90] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[91] W. Zijlstra and A. L. Hof, "Assessment of spatio-temporal gait parameters from trunk accelerations during human walking," November 2003. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/14654202

[92] Google, "Android developers," 01 2017. [Online]. Available: http://developer.android.com