TECHNISCHE UNIVERSITÄT BERLIN

MASTERS' THESIS

Gait analysis methods' review, extension and validation using magnetometer-free inertial data

Author:
Carlos TIANA GÓMEZ

Supervisors:
Dr. Thomas SEEL
Daniel LAIDIG

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Control Systems group
Faculty IV - Electrical Engineering and Computer Science

October 9, 2020

Declaration of Authorship

I, Carlos TIANA GÓMEZ, declare that this thesis titled, "Gait analysis methods' review, extension and validation using magnetometer-free inertial data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

TECHNISCHE UNIVERSITÄT BERLIN

Abstract

Institute of Energy and Automation Technology
Faculty IV - Electrical Engineering and Computer Science

Master of Science

Gait analysis methods' review, extension and validation using magnetometer-free inertial data

by Carlos TIANA GÓMEZ

Gait analysis using inertial measurements units (IMU) offers a wider range of capabilities and freedom compared to other foot motion capture techniques, such as optical markers tracking or instrumented treadmills. However, IMU-based gait analysis is still under development and its accuracy is still being reviewed against other ground truth reference systems.

This thesis aims to review and make a compendium of the existing methods, create a documentation for the *gaitt* project, and serve as a reference guide to newcomers. Beyond this objective, among its goals are also to implement a new rest phase detection algorithm and tune one of the parameters of an orientation estimation algorithm previously developed. For these latter two, the experimental framework consists on recorded data 92 trials of subjects walking on an instrumented treadmill, and equipped with optical markers and and IMU on each foot. This provides two ground truth reference frames against which to compare the ones obtained from the IMU recorded data.

After the implementation, the new rest detection method yields very similar results to the existing one, both in gait phases duration, average stride length, and average gait velocity. The tuning of the orientation estimation algorithm was posed to compute the orientation of the IMU more accurately, but actual results from it implementation have proven to be diminishing and give no reason to use it whatsoever.

Acknowledgements

I would like to thank both my supervisors, Dr. Thomas Seel and Daniel Laidig, given their generosity in accepting yet another pupil in the circumstances this thesis had to be done. They have been there whenever I needed and have provided me with sound advice, insightful remarks and much needed guidance during this thesis. They both have had patience and encouraging words whenever I was lost and have definitely made it easier and more engaging. This thesis would not be the same had they not been my supervisors.

I want to acknowledge my family too because, even though they have not had direct contact with the thesis' contents themselves, they have had contact with me. They have offered me help, have cheered me up, and have encouraged me to continue. Their emotional support has been a constant and a major factor in the completion of this thesis.

Contents

Declaration of Authorship iii

Abstract v

Acknowledgements vii

1 Introduction 1
1.1 Motivation . . . 1
1.2 Objectives . . . 1
1.3 Structure . . . 1
1.4 State of the art . . . 2

2 Theory and algorithms 5
2.1 Previous Concepts . . . 5
2.1.1 General gait terms . . . 5
2.1.2 Gait phases . . . 5
2.1.3 Perry gait phases . . . 6
2.2 Gait Phase Detection . . . 7
2.2.1 Unilateral Gait Phase Detection . . . 7
GPD-One . . . 8
GPD-Two . . . 10
GPD-Three . . . 10
2.2.2 Gait Phase Mapping . . . 11
2.3 Gait Analysis . . . 12
2.3.1 Number of steps . . . 12
2.3.2 Step Indices . . . 12
2.3.3 Step Times . . . 13
2.3.4 Stride duration . . . 13
Using unilateral gait phases . . . 13
Using the gait phases of J. Perry . . . 13
2.3.5 Cadence . . . 14
2.3.6 Stride length . . . 14
Sensor frame to earth frame . . . 15
Sensor frame at previous foot-flat to earth frame . . . 16
Velocity in the earth frame . . . 16
Sensor frame to foot frame . . . 16
Position in the foot frame of the previous foot-flat . . . 17
Stride length . . . 17
2.3.7 Velocity . . . 18
2.3.8 Foot position in the frame of the foot at the first foot-flat . . . 18
2.3.9 Euler Angles . . . 18
2.3.10 Maximum pitch angle . . . 18
2.3.11 Maximum swing roll . . . 18

2.3.12 Z-position . . . 18
2.3.13 Lateral deviation . . . 19

3 3D Visualization **21**
3.1 Post-processing: Correction of the direction of movement . . . 21
3.1.1 Stepwise heading correction . . . 22
3.1.2 Starting point correction . . . 22
3.1.3 Overall direction correction . . . 22
3.1.4 End position correction . . . 23
3.1.5 Stepwise z-correction . . . 23
3.2 Creation of a foot mesh . . . 24
3.2.1 Position and orientation of all foot points . . . 24
3.2.2 Avoid the foot going into the ground . . . 25
3.2.3 Toe orientation . . . 26
3.3 Change of coordinate system to babylonjs standard . . . 26
3.4 Camera movement . . . 26

4 Improvements and extensions **29**
4.1 *gaitt* Comprehensive Guide . . . 29
4.1.1 General Overview . . . 29
4.1.2 The Python code . . . 29
4.1.3 The C/C++ code . . . 31
4.1.4 The Matlab® code . . . 31
4.2 Implementation of new rest detection method . . . 34
4.2.1 Velocity in the earth frame . . . 34
4.2.2 High- and Low-pass filtering . . . 34
4.2.3 Intelligent bias removal . . . 36
4.2.4 Grouping and normalization . . . 36
4.2.5 Rest phase detection . . . 36
4.3 Parameter optimization on orientation estimation algorithm . . . 36
4.3.1 Framework . . . 37
4.3.2 IMU inclination . . . 37
4.3.3 Parameter optimization . . . 38

5 Experimental results **41**
5.1 Experimental framework . . . 41
5.2 New rest detection method . . . 42
5.2.1 Discussion . . . 42
5.3 Parameter optimization on orientation estimation algorithm . . . 44
5.3.1 Exploratory analysis . . . 46
5.3.2 Parameter tuning . . . 46
5.3.3 Implementation of the changes . . . 47
5.3.4 Discussion . . . 49
Parameter tuning . . . 49
Implementation of the changes . . . 53
5.4 Comparing left and right foot data . . . 53
5.4.1 Discussion . . . 56

6 Conclusions **59**
6.1 Overview . . . 59
6.2 Future work . . . 60

A Detailed code explanations **61**
A.1 Python code 61
A.2 C/C++ code 66
A.3 Matlab code 76

Bibliography **79**

List of Figures

2.1 Illustration of the gait phases and the influence of the contra-lateral foot on the observed foot's phases. 6
2.2 Illustration of the new Perry gait phases used in this thesis. 7
2.3 Representation of the GPD-One algorithm used on acc or gyr data to distinguish motion phases (1) from rest phases (0). 8
2.4 Representation of the intersection used on the GPD-One algorithm to merge acc and gyr data. 9
2.5 Representation of the GPD-Two algorithm used to detect the beginning of the swing phase using the tilt rate. 10
2.6 Representation of the GPD-Three algorithm, which uses the norm of the acceleration jerk to detect the initial contact. 11
2.7 Illustration of coordinate systems used. Sensor frame at first foot flat is not represented, but it would be very similar to the one at previous foot-flat. 14
2.8 Illustration of the axes used to compute the rotation matrix R. 17

3.1 Illustration of the heading angle δ for any given step. 22
3.2 Illustration of the end correction algorithm. The dashed lines with dots represent the created linear spaces whereas the black and green lines represent current and corrected paths, respectively; and in purple is represented the displacement vector. 24
3.3 Illustration of the process of creating, positioning and orienting the foot mesh. Each subsection corresponds to a picture on this image. . . 25

4.1 Global project's code flowchart. Python code is coloured grey, C/C++ code blue, and Matlab® red. 30
4.2 C/C++ code flowchart. 32
4.3 Detail of gpdFull function, from C/C++ code flowchart. Implementation of GPD algorithm. 32
4.4 Detail of gait_analysis function, from C/C++ code flowchart. Greater part of the gait analysis. 33
4.5 Main Matlab® functions used for 3D visualization 35
4.6 Setup of IMU and optical markers during the trials 37

5.1 Scatter plot of gait phases duration (in %), comparing the new rest detection method against GPD-One. Bisection of the first quadrant marked in red. 43
5.2 Scatter plot of stride length and gait velocity, comparing the new rest detection method against GPD-One. Bisection of the first quadrant marked in red. 44
5.3 Bland Altman plot of gait phases' duration of both rest detection method against ground truth reference system Zebris. 45

5.4 3D Bar plot of the optimal τ_{acc} and acc_{Rating} parameters and the inclination RMSE (in degrees). Their values are in the horizontal grid, the inclination RMSE is the height of the bars. 46
5.5 Box-plot of the inclination RMSE for different τ_{acc} values. The *Individual* refers to the optimal case (each trial with optimal parameters), and acc_{Rating} was fixed to 4. 47
5.6 Box-plot of the inclination RMSE for different τ_{acc} values, computed with an alternative orientation estimation algorithm. The markings on the x-axis refer to the time constant used 48
5.7 Euler Angles of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters 50
5.8 z Euler Angle of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters 51
5.9 x Euler Angle of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters 51
5.10 y Euler Angle of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters 52
5.11 Bland Altman plot of average gait velocity and average stride length comparing the orientation estimation algorithms and τ_{acc} parameters. . 52
5.12 Box-plot of inclination RMSE separated by foot and scatter plot of difference in inclination RMSE between feet, with the average marked in red. 54
5.13 Distances between optical markers by foot, and angle between vectors used to compute the quaternion describing the optical system frame. . 55
5.14 Detail of the y-position of the front optical marker on the left foot, with 2 filtered used and raw data. 56
5.15 Inclination RMSE by foot for 2 filtered and unfiltered optical markers positions. 57
5.16 Overall inclination RMSE and absolute difference between left and right foot for 2 filtered and unfiltered optical markers positions. 58

List of Tables

1.1 Comparison of papers focused on gait analysis. Taken from [1]. 3

2.1 Mapping of unilateral gait phases into Perry gait phases. 12

List of Abbreviations

acc	**acc**elerometer
EA	**E**uler **A**ngles
fff	**f**irst **f**oot-**f**lat
GPD	**G**ait **P**hase **D**etection
gyr	**gyr**oscope
HR	**H**eel **R**ise
IC	**I**nitial **C**ontact
IMU	**I**nertial **M**easurement **U**nit
opp	**op**posite and **p**arallel to each other
opt	**opt**ical marker system
pff	**p**revious **f**oot-**f**lat
pos	**pos**ition
q	**q**uaternion
ref	**ref**erence
RMSE	**R**oot-**M**ean-**S**quare **E**rror
TO	**T**oe-**O**ff
TUB	**T**echnische **U**niversität **B**erlin
vel	**vel**ocity

Chapter 1

Introduction

1.1 Motivation

Walking is an ability that society as a whole takes for granted. The world we live in, the way we interact, our daily lives are conceived with the idea in mind that almost everyone is able to walk. However, and despite what it may actually seem at first glance, walking is a complex task that requires many different functions and movements to work to be performed successfully. Pain, functional malformations, muscle weakness, or loss of sensory function can hinder the proper execution of these tasks and create problems when walking and, thus, in our lives. Therefore, analyzing a patient's gait does not only fulfill the role of correcting and improving the way they walk, but also finding the root of the problem and, thus, contributing to their health.

There are several options to analyze gait data, even some specifically designed for this purpose, such as instrumented treadmills. Optical markers have also been used, since they offer accurate results and can record on 3D too. However, both of these options require a lab and, therefore, can only be used on clinical environments. This presents a problem when analyzing such a common activity as walking, since there are situations that may not arise in these kind of environments, but may be encountered on our daily lives. The miniaturization of Inertial Measurement Units (IMU) offers the possibility of solving this problem and contributing to expand gait analysis to real-life scenarios. In this thesis, the aim is not only to analyze gait data, but also extend work on the existing algorithms to validate them against the results of already tested methods.

1.2 Objectives

The objectives of this thesis are three, and will be dealt in this same order. First and foremost, part of the purpose of this thesis is to be used as an introduction to the *gaitt* project at the Control Systems group of the Technische Universität Berlin (TUB). This way, a newcomer should have an idea of the purpose, structure and algorithms of the project, as well as an overview of the code. Second, to implement and validate a new rest detection method. And third, to tune a new algorithm and test and validate it after.

1.3 Structure

This thesis is organized in 6 Chapters, including this one, which serves as an introduction. Chapter 2 should be the entry point to any newcomer that has never had

any relation to gait analysis. It explains general gait concepts, the gait phase detection method and several gait analysis techniques. Chapter 3 is dedicated to the visualization tool created by student groups within the Control Systems group at TUB. Even though it is not used for the extensions presented in this thesis, it is still an important part of the *gaitt* project and, in line with the first goal presented earlier, it was deemed worthy of including it. Both Chapters 2 and 3 are greatly based on the work of E. Kastenbauer [1], and updated wherever necessary with the information from [2].

Chapter 4 presents and explains the extensions to the *gaitt* project in this thesis, which are the fulfillment of the objectives aforementioned. Their results and a discussion derived from them are finally shown in Chapter 5.

In Chapter 6 are the conclusions of this thesis. It includes a recapitulation of all other chapters and their main content, contribution and findings. It also proposes new lines of future work according to the findings on this thesis.

1.4 State of the art

The capabilities of IMUs and the lower limitations in environment settings for conducting experiments with them, as opposed to ground truth methods, has increased the interest in research of their applications in gait analysis. Here are presented ten papers that deal with the issue, with different approaches: either with different kind or number of sensors, kind of patients studied or ground truth reference methods. They, along with the summarizing table that accompanies this section (Table 1.1), have been directly taken from [1].

The number of sensors used greatly varies across studies. Some papers use, just as this thesis, a sensor on each foot, like [3]; whereas [4] uses only a single sensor. Others, like [5] use two sensors per foot, and [6] uses a whole frame. Some even use different kinds of sensors simultaneously to improve results, such as [7], that uses optical markers tracking and IMU readings combined; or [8], that uses force and momentum sensors along the IMU measurements.

The ground truth reference system also changes depending on the considered study. Whereas some of them use optical markers (see [4], [6], [7], [8] and [9]), others use the force sensors of an instrumented treadmill, such as [5].

The patients studied is also a differentiating point across papers. From the ten here discussed, only four of them had not healthy subjects within their patients. [10] and [11] studied both healthy patients and others affected by Parkinson's Disease, whereas [3] focused only on the latter. Finally, [9] compared data obtained from healthy subjects from trans femoral amputees.

TABLE 1.1: Comparison of papers focused on gait analysis. Taken from [1].

Paper	**Investigated values**	**System Used**	**Reference System**	**Patients**
[3]	Linear acceleration, cadence, step length, step time	IMUs mounted in left and right soles	3D motion capture system	Affected by Parkinson's disease
[4]	Eight phases defined by J. Perry	Three IMUs on each shoe (surface, shank an thigh)	Optical motion analysis system	Healthy
[5]	Gait speed, swing/stance percent, stride length	Two IMUs on each foot	Instrumented treadmill	Healthy
[6]	Clearance, Toe-Off and Heel-Strike	Sensor frame	Optical motion capture	Healthy
[7]	Step length and foot angles	camera + inertial sensors	Measuring with ruler	Not specified
[8]	Lateral foot placement, stride length	Inertial and force/momentum sensors	Optical position measurement	Healthy
[9]	Four gait phases, foot pitch, roll angle	One IMU on the instep of each foot	Optical motion capture	Healthy and trans femural amputees
[10]	Heel-strike, Toe-off, velocity, stride length	'GaitShoe' sensor suite with inertial sensors and other sensor types	BioMotionLab (BML)	Healthy and affected by Parkinson's disease
[11]	Turning angles, stride velocity and stride length	One IMU on each foot	Optical motion capture	Healthy and affected by Parkinson's disease
[12]	Stride time, relative stance, stride length, velocity	One IMU on the right foot	Velocity of instrumented treadmill	Healthy

Chapter 2

Theory and algorithms

2.1 Previous Concepts

2.1.1 General gait terms

There are some concepts related to the gait that will appear throughout this project and are needed to understand some other core concepts. Besides those ones which already have physical meaning (such as position, velocity or orientation), the rest of core definitions and variables that are most used were proposed by Jacquelin Perry in [13]. They are used extensively in the project and, thus, will be used in this thesis. Some of these that may need an initial explanation are:

- Initial Contact (IC): Sometimes also referred to as heel-strike. It is the moment the foot touches the ground.
- Heel Rise (HR): The moment the heel starts to rise from the floor after resting.
- Toe-off (TO): The moment the toe leaves the ground before swinging forward.
- Contra lateral foot: Foot in which the analysis is not focused on, but still influences the gait.
- Single limb support: Complete body weight is on one foot.
- Step: Interval between Initial Contact and Initial Contact of the contra lateral foot.
- Stride: Also called gait cycle. Interval between two consecutive initial contacts if the same foot, which is also equal to two steps.

2.1.2 Gait phases

J. Perry divided the gait cycle into three phases, each of which is in turn divided into tasks. The phases of the observed and contra lateral foot can be seen in Figure 2.1. These phases are:

- Weight acceptance: The body weight is shifted to the examined foot, both feet touching the ground.
    1. Initial contact.
    2. Loading response: Starts with the Initial Contact of the examined foot and ends with the Toe-off of the contra lateral foot.
- Single limb support.

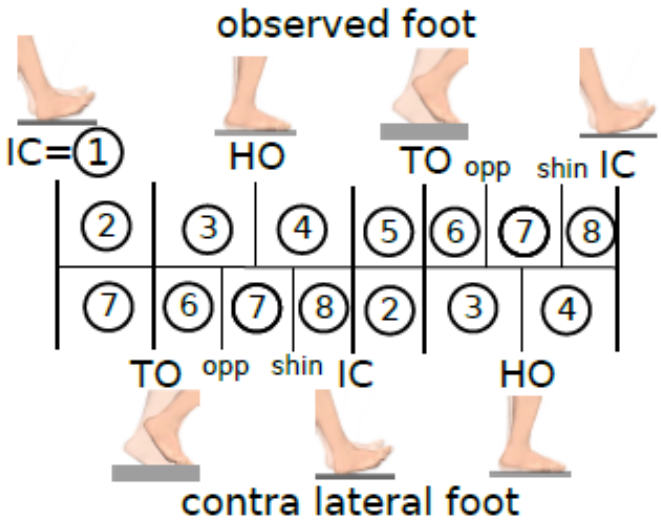

FIGURE 2.1: Illustration of the gait phases and the influence of the contra-lateral foot on the observed foot's phases.

3. Mid stance: Begins with the Toe-off of the contra lateral foot and ends with the shifting of weight to the forefoot.
4. Terminal stance: Starts with the Heel rise and finishes with the Initial contact of the contra lateral foot.

- Limb advancement.

5. Pre-swing: Begins with the Initial Contact of the contra lateral foot and ends with the Toe-off of the observed foot.
6. Initial swing: Starts with the Toe-off and finishes when the feet are opposite and parallel to each other (opp).
7. Mid-Swing: Begins when the feet are opposite to each other and ends when the shin-bone is vertical (shin).
8. Terminal swing: Starts when the shin bone is vertical and ends with the Initial Contact.

2.1.3 Perry gait phases

In this project, the gait cycle division proposed by J. Perry will be slightly modified. Therefore, in the new "Perry gait phases", the Initial Contact will be incorporated to the Loading Response; and the Initial, Mid and Terminal Swing phases will combined into a single Swing phase. The new relationship between the observed foot and the contra lateral one can be seen in Figure 2.2. The resulting gait phases are:

1. Loading response
2. Mid stance
3. Terminal stance
4. Pre swing
5. Swing

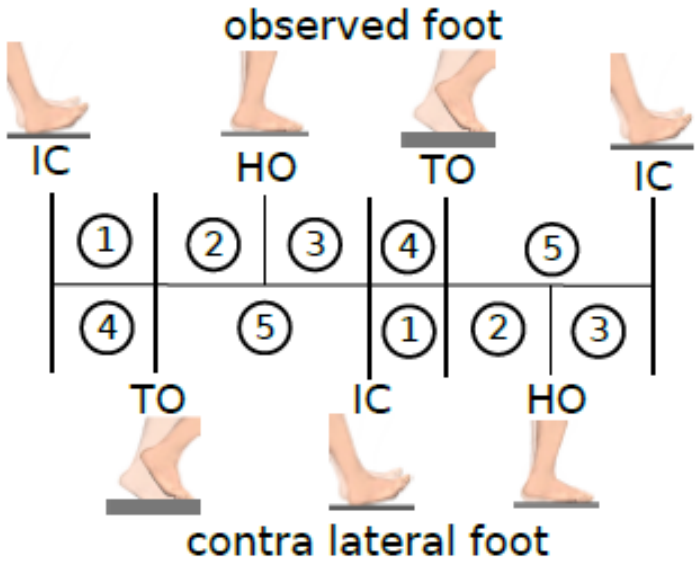

FIGURE 2.2: Illustration of the new Perry gait phases used in this thesis.

2.2 Gait Phase Detection

The ultimate goal of the *gaitt* project, which this thesis is part of, is to analyse gait data with the information from the 3D gyroscope and accelerometer of an IMU mounted in each foot. This gait analysis can be achieved via several plots and values that will be presented later on, but the process must start with the gait phase detection, or GPD.

2.2.1 Unilateral Gait Phase Detection

The most common division of gait into phases is the one proposed by J. Perry, which takes into account the involvement of both feet in the gait process. In this project, there is a preliminary division into 4 different phases using only the information of the foot in movement at the moment. These phases are:

1. Rest phase: Whenever the foot is completely flat on the floor.
2. Pre-swing: Phase between heel rise to toe-off.
3. Swing: Interval in which the foot is on the air, from toe-off to initial contact.
4. Loading response: Phase between initial contact to complete rest.

These gait phases differ from the ones named as Perry gait phases in that they are not influenced by the contra lateral foot. Even though this may imply a loss of information, it also eases significantly the gait phase detection. Moreover, and as it will be seen in 2.2.2, the information of the contra lateral foot can still be added afterwards via mapping.

The whole GPD algorithm is composed by three smaller algorithms applied sequentially, named GPD-One, GPD-Two and GPD-Three, each one detecting the beginning-end of one of the phases aforementioned. The result of the GPD algorithm is an array

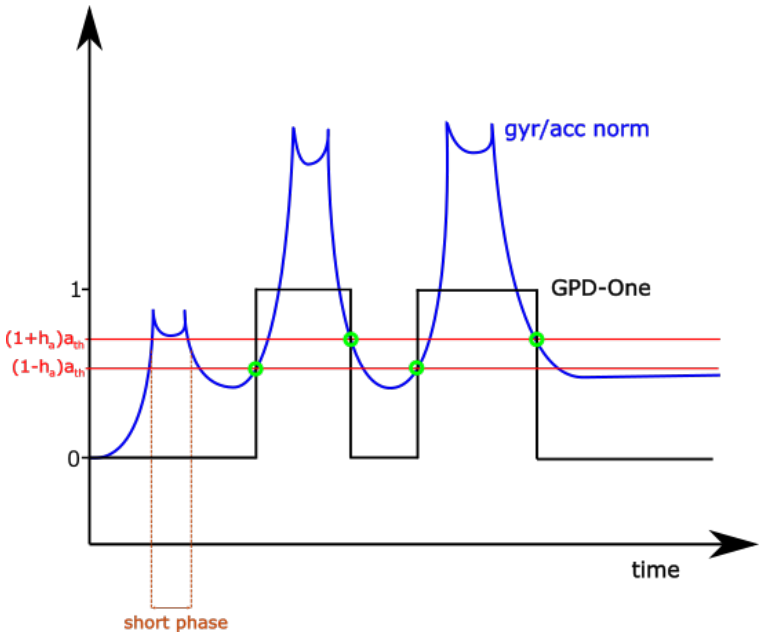

FIGURE 2.3: Representation of the GPD-One algorithm used on acc or gyr data to distinguish motion phases (1) from rest phases (0).

composed of 0-1-2-3 values, which correspond to the gait phases, each being a sample from the IMUs.

GPD-One

This algorithm aims to distinguish between motion and rest phases, and encodes them in an array (*gpdone*) with values 1 and 0, respectively. The result can be seen in Figure 2.3.

Foot-flat detection: First, it computes the norms of the gyroscope and accelerometer readings, and subtracts the value of gravity ($9.81 m/s^2$) to the latter. With each one of them, it performs acausal thresholding by forwards and backwards hysteresis; and threshold a_{th} and hysteresis factor h_a. For the accelerometer norm, the process is as follows:

$$r *_a (t_k) = \begin{cases} 1 & a(t_k) > (1+h_a)a_{th} \\ 0 & a(t_k) < (1-h_a)a_{th} \\ r_a^*(t_{k-1}) & \text{otherwise} \end{cases} \tag{2.1}$$

$$r_a(t_k) = \begin{cases} 1 & r_a^*(t_k) = 1 \\ 0 & a(t_k) < (1-h_a)a_{th} \\ r_a^*(t_{k+1}) & textotherwise \end{cases} \tag{2.2}$$

Then, zero-phases shorter than a defined duration $T_{0,min}$ and one-phases shorter than $T_{1,min}$ are removed.

The same process of acausal thresholding and short phases removal is repeated for gyroscope readings, with threshold ω_{th} and obtaining signal $r_\omega(t_k)$

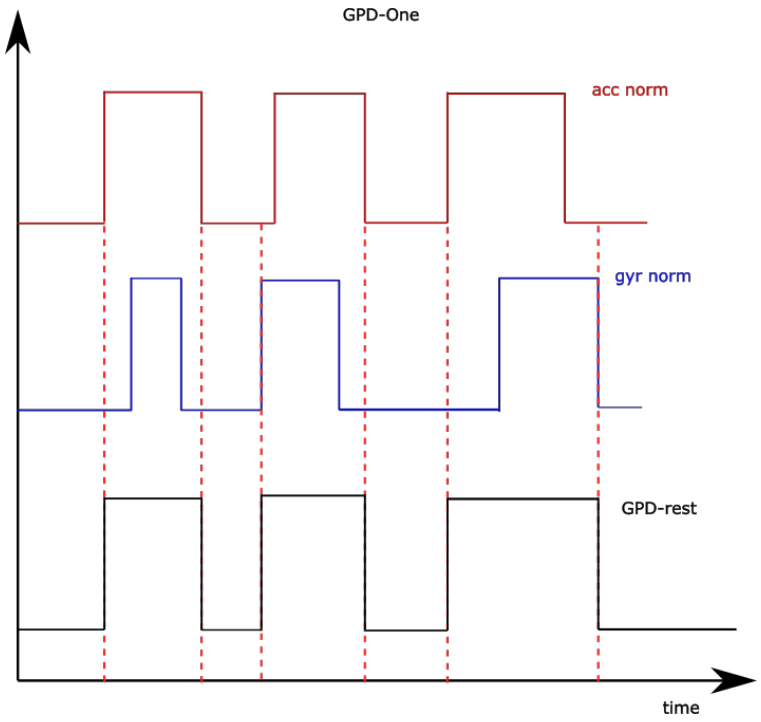

FIGURE 2.4: Representation of the intersection used on the GPD-One algorithm to merge acc and gyr data.

Intersection: The result of this first sub process is two arrays of data, each with its own motion and rest phases. The information from them both is combined into a single array, *gpdone*, in which the resulting rest phases are the intersection of the rest phases of both arrays (it will only be at rest if both and are at rest simultaneously). This also means that if either is in motion, the resulting phase will be in motion too. This is represented in Figure 2.4. Afterwards, the short phases which may have appeared in this process are eliminated again, using a new set of thresholds.

Automatic threshold adaptation: The result of the algorithm is very sensible to the defined thresholds a_{th} and ω_{th}. These values can be found via tuning, and then implemented manually. However, the amplitude of both the accelerometer and the gyroscope signal depends on the gait velocity and other factors. Therefore, an iterative algorithm was developed that was able to tune the thresholds for each trial based on measured data, similar to the one in [14]. The procedure is as follows:

$$a_{th,0} = \frac{1}{2}\left(\max_{t_k \in [t_1,t_n]} a(t_k) + \min_{t_k \in [t_1,t_n]}\right) \tag{2.3}$$

$$T^+ = \{t_k \in [t_1,t_n] | a(t_k) > a_{th,l}\} \tag{2.4}$$

$$T^- = \{t_k \in [t_1,t_n] | a(t_k) \leq a_{th,l}\} \tag{2.5}$$

$$a_{th,l+1} = \frac{w_a}{|T^-|} \sum_{t_k \in T^-} a(t_k) + \frac{1-w_a}{|T^+|} \sum_{t_k \in T^+} a(t_k) \tag{2.6}$$

Where l is the iteration index, w_a is a weight factor and $a_{th,0}$ (the initial value) is the average accelerometer reading's norm. Even though it is an iterative algorithm and should be stopped by convergence, it is actually very cheap computationally-wise, and it is therefore stopped at $a_{th,200}$. As before, the tuning process for ω_{th} is

homologous to the one presented.

GPD-Two

This second process uses the tilt rate of the foot to detect the beginning of the swing phase. First, the tilt rate is defined as:

$$\Gamma = \frac{\int_0^t gyr(\tau)d\tau}{\left\| \int_0^t gyr(\tau)d\tau \right\|_2} \tag{2.7}$$

The tilt rate signal usually presents two local maximums. The start of the swing phase is located somewhere between the first local maximum and the first zero-crossing after it. To find them, the algorithm looks first only at the first half of the swing phase. There, it looks for the maximum value within that window, and then searches for the first value which is greater than half said maximum. Then, it looks for the first zero-crossing after that point.

The start of the swing phase is located somewhere in between the found local maximum and the zero-crossing; it is defaulted to the latter, but it can be adjusted via weighting (i.e.: changed the default place to a given, fixed point of the interval between the maximum and the zero-crossing). It takes then the array from GPD-One and the values from that point until the next foot-flat phase are shifted from 1 to 2. Figure 2.5 displays the result of applying the GPD-Two algorithm.

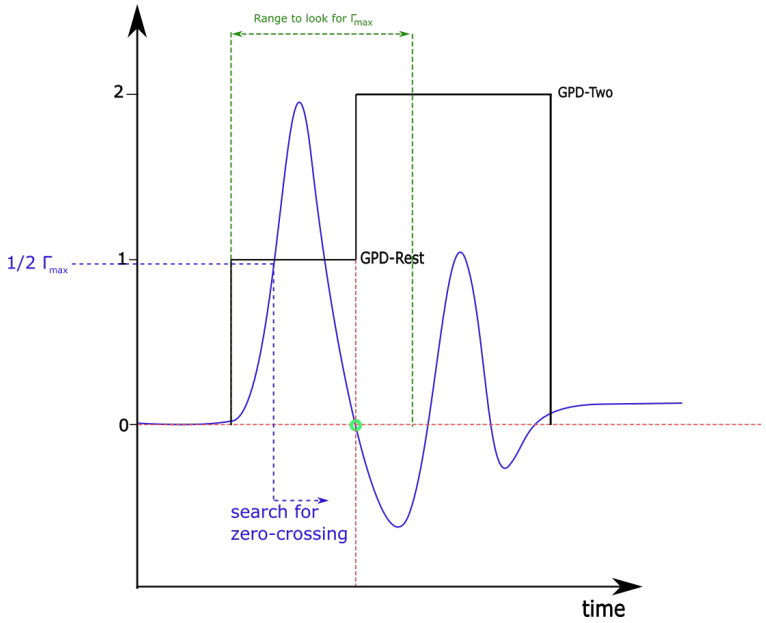

FIGURE 2.5: Representation of the GPD-Two algorithm used to detect the beginning of the swing phase using the tilt rate.

GPD-Three

This third and last phase aims to find the beginning of the loading response, which is marked by the initial contact. It uses the jerk of the acceleration, which is its derivative, computed as:

$$jerk(t) = \frac{d}{dt} acc(t) \tag{2.8}$$

The algorithm looks on the last 30% of the swing phase determined up until this moment (which are those samples where the result of the GPD-Two had a 2 as a value). Within that interval, it searches for the maximum jerk norm, and marks it as the initial contact. All the values from there on until the next rest phase, and those immediate previous values that are higher than a fixed threshold (considered as a percentage of the achieved maximum), are marked as 3 for the GPD. The final result of the GPD algorithm can be seen on Figure 2.6.

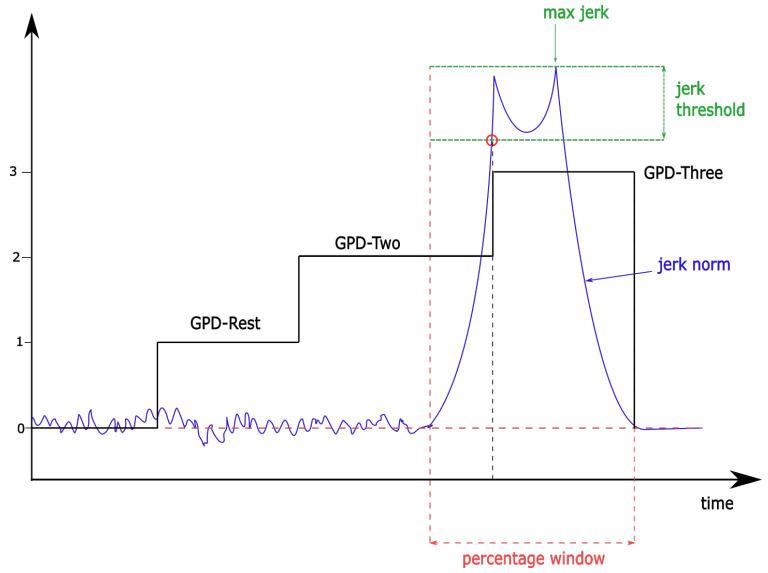

FIGURE 2.6: Representation of the GPD-Three algorithm, which uses the norm of the acceleration jerk to detect the initial contact.

The result, as stated before, is an array of samples which has sequences of 0-1-2-3 values, and form the GPD array. With it, every sample from the IMU can be mapped into the associated gait phase.

2.2.2 Gait Phase Mapping

As it was foreshadowed, the Perry gait phases can still be obtained mapping the information of the studied foot and the contra lateral one. The mapping used is shown in table 2.1.

There are a few corner cases that are worth an explanation. In cases where the subject walks slow (such as an elderly person), it can happen that the Initial contact of the contra lateral foot appears before the Heel rise of the observed foot (cases (3,0) and (0,3)). Those are assigned to the pre-swing phase and loading response, respectively.

The other case is when the subject is walking extremely slow, where moments of double support can appear and, as such, both feet are at rest simultaneously. In this situation, the phases of both feet are (0,0), but it is also the case of a mid-stance. To separate these three cases, an additional variable has been created, *valprev*, that holds the value of the last Perry gait phase. By default, the algorithm assigns the Perry gait phase to -1 (mid stance phase), unless *valprev* has value 1 or 3. Therefore, the three cases are (represented with a sub-index in the Perry gait phases in the table):

- $valprev = 1$: Perry gait phase = 1.
- $valprev = 3$: Perry gait phase = 3.
- *else*: no changes on Perry gait phase.

TABLE 2.1: Mapping of unilateral gait phases into Perry gait phases.

Gait phase of studied foot	0			0	3	1	0	1	1	2	2	2	0	3	3
Gait phase of contra lateral foot	0			2	2	2	3	3	0	3	0	1	1	0	1
Perry gait phase	-1_{else}	1_1	3_3	-1	-1	0	1	1	1	2	2	2	3	3	3

2.3 Gait Analysis

The following section presents several methods to compute variables of interest to perform a gait analysis. These variables are either used as gait analysis tools, to compare different signal acquisition systems, or as intermediate variables for the computation of other values. In those cases where it is necessary to use the gait phases, the unilateral phases will be used unless specified otherwise.

2.3.1 Number of steps

A step is a full sequence of phases 0-1-2-3-0, and it must include all of them and in that order. Therefore, the algorithm counts the number of 0-1-2-3-0 sequences in the GPD array to find the number of steps.

2.3.2 Step Indices

The GPD array contains the associated gait phase of every single sample recorded. However, this information can be summarized into an array of arrays (one for each step), called *stepindices*. These second level arrays are formed by 6 values: the first index of the step, the indexes associated to the first sample of each phase within the step, and the last sample for the step. They are found as follows:

1. Index of the first sample of the step. Found via:
   (a) Find number of samples on the foot-flat phase before the step:
   $$num_{0\_samples} = index_{start1} - index_{start0} \tag{2.9}$$
   Being $start_1$ and $start_0$ the first sample of the pre-swing phase and the first sample of the foot-flat phase, respectively.
   (b) Find the index of the first sample of the step:
   $$index_{stepStart} = \left\lceil index_{start1} - \frac{num_{0\_samples}}{2} \right\rceil \tag{2.10}$$

2.-5. Index of the first sample of the pre-swing/swing/loading response/rest phase.

6 Index of the last sample of the step, found via:
   (a) Find the number of samples in the foot-flat phase after the step, as in step 1.

(b) Find the index of the last sample of the step:

$$index_{stepStop} = \left\lceil index_{start0} + \frac{num_{0\_samples}}{2} \right\rceil \tag{2.11}$$

2.3.3 Step Times

With the previous information and knowing the sampling rate, it is possible to compute the duration of each phase at each step. This produces again an array of arrays (*steptimes*), being the content of each array(all measures in seconds):

1. Time at the beginning of the step.
2. Duration of the rest phase.

3.-5. Duration of the. pre-swing/swing/loading response phase.

6. Time at the end of the step.

The duration of each phase can be easily computed sing the following formula:

$$duration_{phase} = \frac{index_{phase\_end} - index_{phase\_start}}{rate} \tag{2.12}$$

Where $index_{phase\_start}$ and $index_{phase\_end}$ are the start and end indices for the studied phase, respectively, that are found in the previous result, *stepindices*.

Recalling Equations 2.10 and 2.11, there is a rounding in both of them. This causes that, whenever the number of samples in rest phase is uneven, the index of the last sample of a step and the first sample of the next step do not coincide. This means that there can appear a gap between two consecutive steps. To solve this, the duration of this gap is computed and split evenly across the rest phases of the two involved steps. Note too that the rest phase has been split in two, so it is necessary to merge the duration of the two halves.

2.3.4 Stride duration

With the information of the previous results, *steptimes* and *stepindices*, the duration of every stride can be computed.

Using unilateral gait phases

The duration of each stride, using the unilateral gait phase definition, can be computed using just *steptimes* and a simple summation of the duration of each gait phase within it. Moreover, it is useful to compute too the percentage of the stride duration corresponding to each gait phase, done by:

$$percentage_{gaitphase} = \frac{duration_{gaitphase}}{duration_{total}} 100 \tag{2.13}$$

Using the gait phases of J. Perry

Since when this was created the step duration had not yet been described, the percentage duration of the Perry gait phases (described in Section 2.1.3) can be computed using the indices of each phase found in Section 2.3.2. Given that the sampling

rate is constant, the proportion of the duration of each phase in number of samples over the total number of samples in a step is the same as if it were computed with duration instead. Therefore, the samples of each phase can be known via the mapping described in Section 2.2.2, and its respective duration is just a division of the number of samples in each phase with the number of (valid) samples in the step.

2.3.5 Cadence

Cadence is defined as the number of steps per minute, and each stride is two steps. Therefore, knowing the duration of each stride, cadence for the stride i can be computed as:

$$cadence_i = \frac{60s}{\frac{1}{2} \cdot stride_{duration}} \tag{2.14}$$

2.3.6 Stride length

The coordinate system of the IMUs mounted on the feet do not need to align with the coordinate system of the foot, which requires some kind of coordinate axes transform in order to achieve the desired results. To describe each necessary axis and make the transformations necessary, quaternions will be used. The used frames, shown in Figure 2.7, are presented below:

- Sensor frame
- Global earth frame
- Foot frame
- Sensor frame at previous foot-flat (pff)
- Sensor frame at first foot flat (fff), necessary for further analysis later on Section 2.3.8

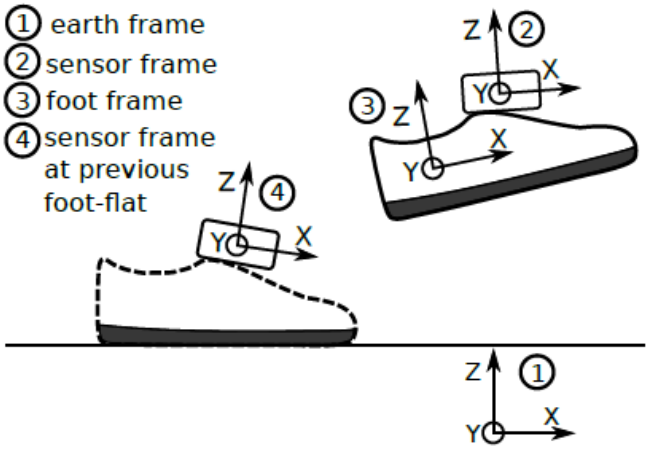

FIGURE 2.7: Illustration of coordinate systems used. Sensor frame at first foot flat is not represented, but it would be very similar to the one at previous foot-flat.

Stride length is nothing more than the difference in positions at the beginning and end of the stride; but these need to be computed first. Thus, a position estimation algorithm is used to retrieve these positions in the appropriate frames (note that, due the IMUs being mounted on the foot too and it being on movement, not even their frames on two consecutive foot-flats have to be the same). The steps of this algorithm are the following:

Sensor frame to earth frame

The quaternion that gives the rotation between the sensor and the earth frames, q_{earth}^{sensor}, must be computed by an orientation estimation algorithm. Currently, there are two existing methods: the one that will be explained here, which is the default one; and another one similar to the one described in [15].

The process first starts with an arbitrary initial quaternion $q(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$, and then performs strapdown integration from the gyroscope readings:

$$q_\omega(t_k) = q_\omega(t_{k-1}) \otimes \left[\cos\left(\frac{T_s}{2} \|\omega(t_k)\|\right) \quad \frac{\omega(t_k)^T}{\|\omega(t_k)\|} \sin\left(\frac{T_s}{2} \|\omega(t_k)\|\right) \right]^T \tag{2.15}$$

Afterwards, the measured acceleration can be converted to an inertial frame using that same quaternion:

$$a_\omega(t_k) = q_\omega(t_k) \otimes a(t_k) \otimes q_\omega(t_k)^{-1} \tag{2.16}$$

In this new frame, the gravitational acceleration will always pull from the same direction, not matter the sensor orientation. Moreover, when integrating, the acceleration and the deceleration will cancel each other out. Using this property, each component of this new acceleration $a_\omega(t_k)$ is filtered using a zero-phase acausal filter using a moving average filter with a window size of T_a forwards and backwards. Once filtered, and assuming that the velocity changes from two consecutive samples will be small, the filtered acceleration will be dominated by gravity. When moving it back to sensor frame:

$$a_f(t_k) = q_\omega(t_k)^{-1} \otimes a_\omega(t_k) \otimes q_\omega(t_k) \tag{2.17}$$

It is possible to correct the inclination of the gyroscope strapwdown integration quaternion $q_\omega(t_k)$ by using this acceleration $a_f(t_k)$ as a vertical reference.

Taking $q_a(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$, the acceleration can once again be brought to sensor frame:

$$a_r(t_k) = q_a(t_{k-1}) \otimes q_\omega(t_k) \otimes a_f(t_k) \otimes (q_a(t_{k-1}) \otimes q_\omega(t_k))^{-1} \tag{2.18}$$

Then, the inclination can be corrected:

$$n(t_k) = a_r(t_k) \times \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \tag{2.19}$$

$$\alpha(t_k) = \arccos(\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \frac{a_r(t_k)}{\|a_r(t_k)\|} \tag{2.20}$$

$$q_a(t_k) = q_a(t_{k-1}) \otimes \left(\cos\left(\frac{\alpha(t_k)}{2}\right) \quad \frac{n(t_k}{\|n(t_k)\|} \sin\left(\frac{\alpha(t_k)}{2}\right)\right) \tag{2.21}$$

And, finally, by quaternion multiplication between the gyroscope strapdown integration quaternion and the accelerometer correction quaternion, the desired quaternion can be computed:

$$q_{earth}^{sensor}(t_k) = q_a(t_k) \otimes q_\omega(t_k) \tag{2.22}$$

Sensor frame at previous foot-flat to earth frame

From the previous result, q_{earth}^{sensor}, and the information contained in *stepindices*, it is possible to obtain the quaternion that transforms from the middle of the previous foot-flat to the earth frame, $q_{earth}^{sensor\_pff}$.

Velocity in the earth frame

The velocity in earth frame can be obtained from the accelerometer readings, that are in sensor frame. They are transformed into earth frame using the previously computed quaternions, as such:

$$q_{acc\_earth}(t) = q_{earth}^{sensor(t)} q_{acc\_sens}(t) q_{sensor(t)}^{earth} \tag{2.23}$$

Being $q_{acc\_sens}(t) = [0, a_x(t), a_y(t), a_z(t)]^T$, and $a_i(t)$ the acceleration in the global i direction. The velocity in earth frame can now be determined by integrating the newly transfomred acceleration:

$$vel\_earth(t) = \int_0^t acc\_earth(\tau) d\tau \tag{2.24}$$

A linear drift correction is applied too to this velocity, so that it is effectively 0 in the middle of the next foot-flat (nff), with the following formula:

$$vel_i^{corrected}(t) = v_i(t) - \frac{t}{t_{nff}} v_i^{nff} \tag{2.25}$$

Where $v_i(t)$ is the velocity on the i axis ($i \in x, y, z$), and v_i^{nff} is the velocity at the next foot-flat, or $v(t_{nff})$.

Sensor frame to foot frame

To get the quaternion that relates the foot frame to the sensor one, the rotation matrix composed by the X, Y and Z axes of the foot frame on the sensor frame needs to be computed first. The process, which can be seen in Figure 2.8, is the following one:

1. Z: During foot-flat phases, as the foot is still and lying on the ground, the only acceleration present is (or should be) gravity. Therefore, the average of the accelerometer z readings during that phase can be equaled is the same as the one of the foot frame.

2. X': This is an auxiliary, but necessary, axis, used to later compute the actual X axis. This axis points in the forward direction of movement, and is obtained using the velocity in earth frame $vel\_earth(t)$. To do so, it computes the normalized average of this velocity on the middle 50% window time interval of the swing phase (when the foot is moving forward, leaving the first and the last 25% outside the window).

3. X and Y: Since Z and X' form a plane, it is possible to find the remaining axes using by forcing orthogonality:

$$Y = Z \times X' \tag{2.26}$$

$$X = Y \times Z \tag{2.27}$$

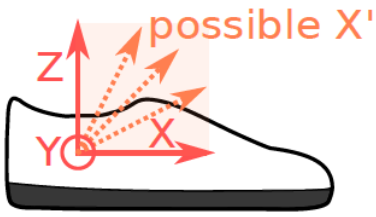

FIGURE 2.8: Illustration of the axes used to compute the rotation matrix R.

From these 3 axes it is possible to obtain a rotation matrix, R = [X, Y, Z], and from it the quaternion q_{foot}^{sensor}.

Position in the foot frame of the previous foot-flat

The multiplication of the quaternions $q_{earth}^{sensor\_pff}$ and the inverted quaternion q_{foot}^{sensor} yields $q_{earth}^{foot\_pff}$, which is the quaternion between the frame of the foot at previous foot-flat and the earth frame. With it, it is possible to determine the velocity in the foot frame at the previous foot-flat:

$$vel_{footframe\_pff}[k] = vel_{earth} \otimes q_{foot\_pff}^{earth} \tag{2.28}$$

This velocity can be integrated to obtain the position, but since its frame changes at every instant, the operation is:

$$pos_{foot\_pff\_i}[k] = h \sum_{n=s}^{k} vel_{footframe\_pff\_i}[n] \quad \text{for } i \in \{x, y, z\} \tag{2.29}$$

Being s the number of strides.

Stride length

Now it is possible to know the x and y coordinates of the foot at the middle of the foot-flat, referenced to the previous foot-flat. Thus, the distance is:

$$dist = \sqrt{(x_k - x_s)^2 + (y_k - y_s)^2} \tag{2.30}$$

2.3.7 Velocity

Knowing the stride length and duration, it is possible to compute the velocity (in *m/s*) at every stride *i* with just a simple division:

$$vel[i] = \frac{stride\_length[i]}{stride\_duration[i]} \tag{2.31}$$

2.3.8 Foot position in the frame of the foot at the first foot-flat

The procedure is the same as the one used when computing the position in the foot frame of the previous foot-flat in 2.3.6, but referencing the first foot-flat instead.

2.3.9 Euler Angles

Euler angles can be derived from any of the available quaternions. There are three that have been deemed interesting and, thus are computed and then saved. These are (EA stands for Euler Angles):

- EA^{foot}_{earth}
- $EA^{sensor}_{sensor\_pff}$
- $EA^{foot}_{foot\_pff}$

2.3.10 Maximum pitch angle

The third component of the Euler angle $EA^{foot}_{foot\_pff_3}$ stores the pitch angle. Thus, from all samples *i*, the maximum value is found and transformed from radians to degrees. This will give and idea of the mobility of the subject.

$$pitch[i] = \frac{180}{\pi} \cdot EA^{foot}_{foot\_pff_3}[i] \tag{2.32}$$

2.3.11 Maximum swing roll

The second component of the same Euler angle as before, $EA^{foot}_{foot\_pff_2}$, encodes the roll of the foot. In this case, the maximum is only looked at during swing phases, instead of the whole movement.

$$roll[i] = \frac{180}{\pi} \cdot EA^{foot}_{foot\_pff_2}[i] \tag{2.33}$$

2.3.12 Z-position

It takes the Z-position in foot frame at pff for every step and corrects the linear drift that may appear. To do so, it is assumed that the subject is walking on level ground and it is imposed that the final global Z-position (*endPos*) at every step is 0.

$$Z[i] = Z_{old}[i] - endPos \cdot \frac{i}{N_s - 1} \tag{2.34}$$

Where $i \in [0, N_s - 1]$ and N_s is the number of steps.

Besides that, a moving average filter is applied to smooth results, and the maximum Z-position at every step is saved too.

2.3.13 Lateral deviation

The lateral deviation from the straight path between the start and end point of a stride can be computed for every sample. Both the start (0), end (N, number of samples on the step), and current position (i), referenced to the pff can be found as in section 2.3.6. The formula for lateral deviation of sample i is:

$$latdev_i = \frac{|(y_N - y_0) \cdot x_i - (x_N - x_0) \cdot y_i + x_N \cdot y_0 - y_N \cdot x_0|}{stride\_length} \tag{2.35}$$

As with the Z-position, the lateral deviation is smoothed with a moving average filter and then the maximum value of every step is saved too.

Chapter 3

3D Visualization

Within the scope of the *gaitt* project is also a tool to visualize the gait recorded from a given trial. The program was created by two student project groups at TU-Berlin, and it is based on the *babylonjs* framework by JavaScript. This visualization displays the movement of the foot in four different angles and grants the user some control over the simulation, such as: change the reproduction speed, start and stop the simulation, play it backwards, and move around the cameras. No changes were done in this thesis, but it was deemed part of the project and, thus, worthy of being included.

This 3D visualization uses then the gait analysis data obtained in Chapter 2 and re-processes it, creates a foot mesh, positions it using that data, and creates the gait animation. The data handling part is done in Matlab®, while the visualization program itself is written in babylonjs, based on the JavaScript framework.

The changes that need to be made are:

1. Change the data into the babylonjs standard.
2. Show the movement of the entire foot, not just the sensor.
3. Prevent the foot from moving into the ground.
4. Make the toes have a realistic movement: give them their own orientation.
5. Align the direction of movement of both feet to each other and to a global coordinate system.
6. Correct, as much as possible, the drift in sensor data.

3.1 Post-processing: Correction of the direction of movement

As stated above, there is a series of corrections that can be performed to improve the visualization. Unless stated otherwise, these corrections are optional and have to be initialized by their corresponding flags when first executing the simulation. Besides this initialization, the starting position of both feet in global coordinate system and the position of the sensor, heel, mid-foot and toe (the foot's key points) in global coordinate system must be provided too. Finally, the foot's middle orientation (q_{earth}^{foot}), the sensor position (*pos_earth*), the gait phases (GPD), the step indices of each foot, and the sampling rate are loaded from the gait analysis results.

3.1.1 Stepwise heading correction

This algorithm was created by Daniel Laidig. The aim is to find the overall walking direction and make it the x-axis of a global coordinate system, all under the assumption that the subject walks along a straight path. Afterwards, the positions and orientations are transformed into this new global reference frame. The process is as follows:

1. Find the heading of each step and the corresponding heading angle, and save it into the variable δ (represented in Figure 3.1
2. Low-pass filter δ into δ_{LP}.
3. With δ_{LP}, find the correction quaternion q_{delta} as follows:

$$q_{delta} = \left[\cos\left(\frac{-\delta_{LP}}{2}\right) \quad 0 \quad 0 \quad \sin\left(\frac{-\delta_{LP}}{2}\right)\right] \tag{3.1}$$

4. Differentiate position to find velocity in each step.
5. Correct velocity and orientation with the computed quaternion q_{delta}.
6. Integrate the corrected velocity to find the new position.

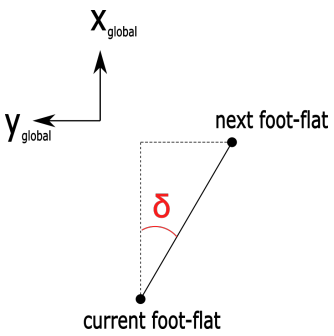

FIGURE 3.1: Illustration of the heading angle δ for any given step.

3.1.2 Starting point correction

Since in the previous section 3.1.1 the position of both feet was corrected, and it was assumed that each started at the origin of the global coordinate system, the result would be that both feet would walk over the same line, stepping one on top of the other. To avoid this, the distance between them is measured and its half is added or subtracted (depending on the foot) so as to separate them from the middle line.

3.1.3 Overall direction correction

Note that this correction is not used by default, due to it being unnecessary when the other corrections are applied. The goal of this correction is to ensure that the subject is walking along an approximately straight line along the global x-axis. To achieve this, the vector between the start and end position is computed; and with it the angle that represents the deviation from a straight line is computed and used to correct the position. The process is as follows:

$$\vec{dir}_{original} = pos_{end} - pos_{start} \tag{3.2}$$

$$\vec{dir}_{corrected} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{3.3}$$

$$angle_{correction} = \arccos\left(\frac{\vec{dir}_{original} \cdot \vec{dir}_{corrected}}{\|\vec{dir}_{original}\| \cdot \|\vec{dir}_{corrected}\|}\right) \tag{3.4}$$

The position can be adjusted by rotation it $angle_{correction}$ around the global vertical z-axis; whereas the orientation correction is achieved by means of quaternion multiplication.

3.1.4 End position correction

There are several assumptions that are taken in the algorithm used to correct the end position. Firstly, it is assumed that the end y-difference (the lateral deviation from the middle of the straight path followed) is the same for both feet. The initial x-position for both feet is considered the same, and it equals the average between the two of them after the previous corrections are applied. Lastly, the global z-position is assumed to be 0.

Since the drift depends on velocity, and this changes at each moment of a step, the position correction will be done based on location. First, 2 linear spaces are created with 100.000 samples each: one from the corrected start to the corrected end (i.e.: a straight line between both desired positions with 100.000 points in between), and the other from the current start to the current end (i.e.: a straight line from the actual start to the actual end, with 100.000 points too, that approximately resembles the actual path taken by the subject). For each point (sample) of the path, it finds the sample on the current space that is closest to it and, from it, finds the displacement vector to the corrected space (the vector that relates the chosen point in current space to the comparable point in the corrected space). Afterwards, this vector is applied to the point in the path, effectively moving and correcting it.

Figure 3.2 shows the process of the end position correction. In black, the current path, and in green, the corrected path. The purple arrows represent the displacement vector; first found between both linear spaces (dashed lines with multiple dots), and then applied to one position on the current path.

3.1.5 Stepwise z-correction

This correction is always done, no matter the flags selected. It corrects the drifts in the z-position under the assumption that the ground is level. The algorithm finds the beginning, end, and middle point of the rest phase, where it is assumed that the foot is standing still. Then, it performs a linear drift correction over the movement period by setting that middle point of the rest phase to zero. Also, to avoid discontinuities, the last value of the correction is subtracted to all z-positions in all following steps. The formula to do so is:

$$z\_pos_{new} = z\_pos_{old} - \frac{t}{n} \cdot z_{s,t} - \left(\sum_{i=0}^{st-1} z_i\right) \tag{3.5}$$

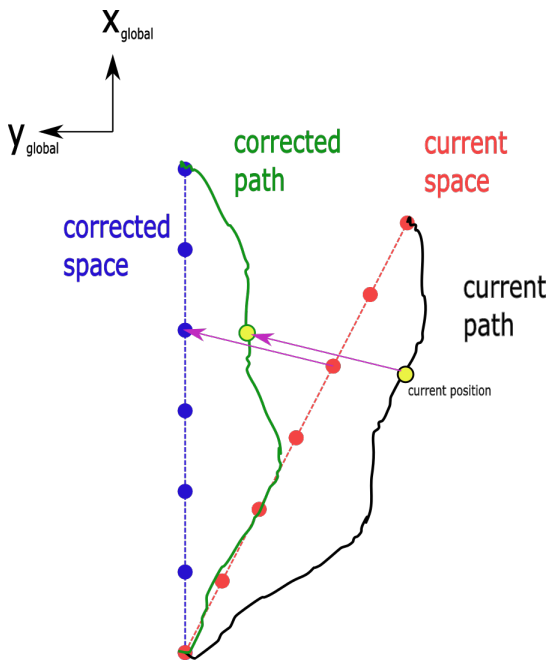

FIGURE 3.2: Illustration of the end correction algorithm. The dashed lines with dots represent the created linear spaces whereas the black and green lines represent current and corrected paths, respectively; and in purple is represented the displacement vector.

Where z_s is the (uncorrected) z-position in the middle of the next rest phase after step s; st is the current step, $t \in [1, n]$ is the number of samples within it, and n is the number of samples between the end of the rest phase of the current step and the beginning of the next one. Note that, since the value of t is clipped between 0 and n, if the movement has not yet started it will have a value of 0, whereas if that has already finished, it will have a value of n.

3.2 Creation of a foot mesh

Up until now, the position of a single point of a foot was specified, but it is not sufficient to accurately position and orientate (and, thus, visualize) the whole foot. Therefore, more points are necessary, which are: the heel, the mid-foot, and the tip of the toes. Since the corrections performed previously only dealt with the position and orientation of the sensor, the rest of the points of the foot will have to be rotated and positioned too, while taking care that they are, at all times, above the ground. Also, note that as the toes can move differently than the rest of the foot, they will need their own orientation. The whole process can be seen in Figure 3.3

3.2.1 Position and orientation of all foot points

The middle orientation quaternion was already loaded when initializing all necessary variables in section 3.1. This quaternion will allow to rotate all the points of the foot mesh to the adequate orientation. The vector relating the difference in the sensor position between the original one and the newly rotated one, computed by simple subtraction between both positions, will allow to translate the all the other points of the mesh to their actual position.

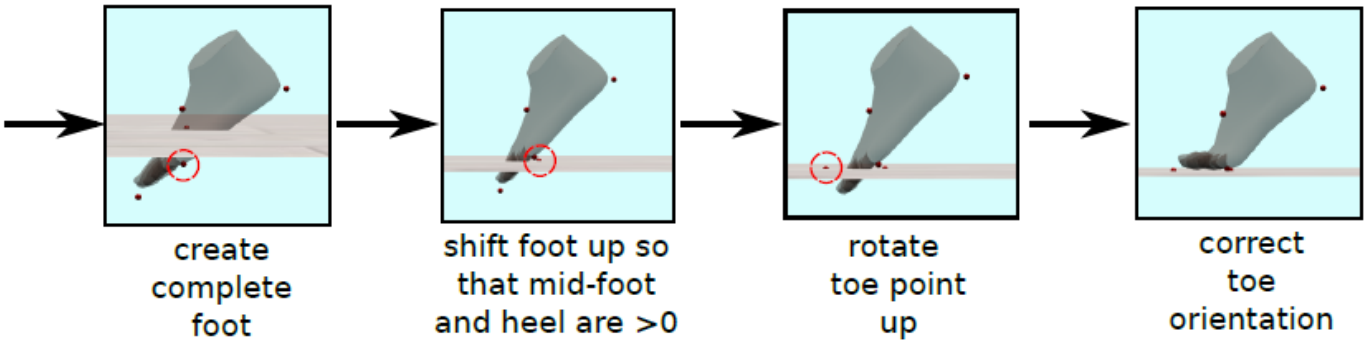

FIGURE 3.3: Illustration of the process of creating, positioning and orienting the foot mesh. Each subsection corresponds to a picture on this image.

3.2.2 Avoid the foot going into the ground

The z-position correction done in 3.1.1 only took into account the position of the sensor, but it could be that other parts of the foot still lie beneath the ground. Thus, all key points barring the toes (that will be changed later on), are shifted upwards in the z direction until all of them are above the ground.

Once all other points are corrected, the toes' orientation needs to be changed, fit the actual gait movement. To achieve so, is is interpreted that, whenever the toes are under the ground (and all points are already above it), they should be rotated - their orientation will be different than that of the rest of the foot -. In this case, they will be rotated upwards so they lie flat on the ground. There are three geometric assumptions that allow to achieve it:

1. The ratio between the global x and y distance between the mid-foot and the toe position remains constant. This is true because the direction of the toes and the mid-foot in the horizontal plane should be the same.

$$x_{dist} = x_{toe\_global} - x_{midfoot\_global} \tag{3.6}$$

$$y_{dist} = y_{toe\_global} - y_{midfoot\_global} \tag{3.7}$$

$$ratio = \frac{x_{dist}}{y_{dist}} = constant \tag{3.8}$$

2. The distance between the mid-foot and the toe point of the foot mesh, which is called the toe length, remains also constant:

$$length_{toe} = \sqrt{x^2_{dist\_new} + y^2_{dist\_new} + z^2_{dist\_new}} = constant \tag{3.9}$$

3. The new z-coordinate distance is the opposite of the global mid-foot z-position, in order for the tip of the toe to stay above the ground.

$$z_{dist\_new} = -z_{midfoot\_global} \tag{3.10}$$

With these conditions, x_{dist_new} and y_{dist_new} can be computed:

$$y_{dist\_new} = \pm\sqrt{\frac{length_{toe}^2 - z_{dist\_new}^2}{1 + ratio^2}} \tag{3.11}$$

$$x_{dist\_new} = y_{dist\_new} \cdot ratio \tag{3.12}$$

Where the sign of y_{dist_new} is the same as the original one. Finally, the new position of the foot can be found via:

$$pos_{toe\_new} = pos_{midfoot\_global} + \begin{pmatrix} x_{dist\_new} \\ y_{dist\_new} \\ z_{dist\_new} \end{pmatrix} \tag{3.13}$$

3.2.3 Toe orientation

Even though all key points have been already positioned, the orientation of the points in the toes still has to be corrected for the visualization. The steps to achieve it are:

1. For each sample, set toe orientation to to mid-foot orientation.
2. Determine current y-axis of the foot frame by rotation the global y-axis with the quaternion of the mid-foot orientation.
3. With the heel, mid-foot and toe key points find the correction angle between the toe segment and the main foot.
4. With this angle and the y-axis, the correction quaternion can be computed. With it, the orientation of the points of the toes can be adjusted for every time instant.

3.3 Change of coordinate system to babylonjs standard

The coordinate system used by *babylonjs* uses a different coordinate system than the one used up until now. The change of frames should be:

1. The x-coordinate has to be the new third coordinate.
2. The y-coordinate has to be the new first coordinate.
3. The z-coordinate has to be the new second coordinate.

Positions are changed by just switching the order of coordinates, whereas the orientations are shifted by quaternion multiplication with two normalized constant correction quaternions defined within the code.

3.4 Camera movement

The cameras that follow the feet during gait need also position and orientation so that the visualization is successful. In this case, whenever only one foot was selected, it was chosen the mid-foot position as the one the cameras must follow. If it

is of interest to follow both feet simultaneously, the chosen point is the average between both mid-foot positions. Since it can cause problems of smoothness, a moving average filter is performed over a window of 1 second.

Chapter 4

Improvements and extensions

4.1 *gaitt* Comprehensive Guide

Part of the purpose of this thesis is to serve as a comprehensive guide to the *gaitt* project for newcomers. The intent is not to explain or further extend gait concepts or analysis techniques (those were already discussed in the previous chapter) but rather present the used and tested code in an structured way and serve as an entry point to the project.

It is important to point out that this project also makes use of some of the bigdata project's tools. Among them, the most remarkable here is the use of DataFile. These is a custom created file format saved in .mat files (in this project, .gaitt.mat) that is able to store many different kinds of data. For more information regarding this and other tools, refer to the *bigdata* project documentation.

4.1.1 General Overview

The code is written in Python, C/C++ (with Cython scripts being the nexus between both) and Matlab® , with some files encoded in .json format too. The Python files are mostly related to processing and creating different kind of files, organizing and formatting the data, creating plots and calling several sub-processes, such as the C/C++ and Matlab® functions. The code written in C/C++ is related to gait phase detection and gait analysis, and the Matlab® deals with the 3D representation of the foot and gait movement: creating and positioning the foot mesh, correcting the movement for visualization, and saving it into .json files for its later use. Appendix A contains tables with a more detailed description of the functions used.

The overview of the whole code can be appreciated in Figure 4.1. The Python files are coloured grey, the C/C++ are coloured blue, and the Matlab® are coloured red. The figure represents a flowchart of the function calls, in a nested structure, with the file they belong to stated atop the coloured rectangle. Thus, represented sequential functions indicate a function call within the previous function; whereas parallel function calls indicate successive function calls within the function they stem from (for instance, checking the Matlab® code, the functions *align_direction*, *footV2*, *find_ground*, *quaternionMultiply* and *append_to_json* are called one after the other inside the *create_json* function).

4.1.2 The Python code

The entry point is demo.py, that accepts several types of inputs, and proceeds accordingly:

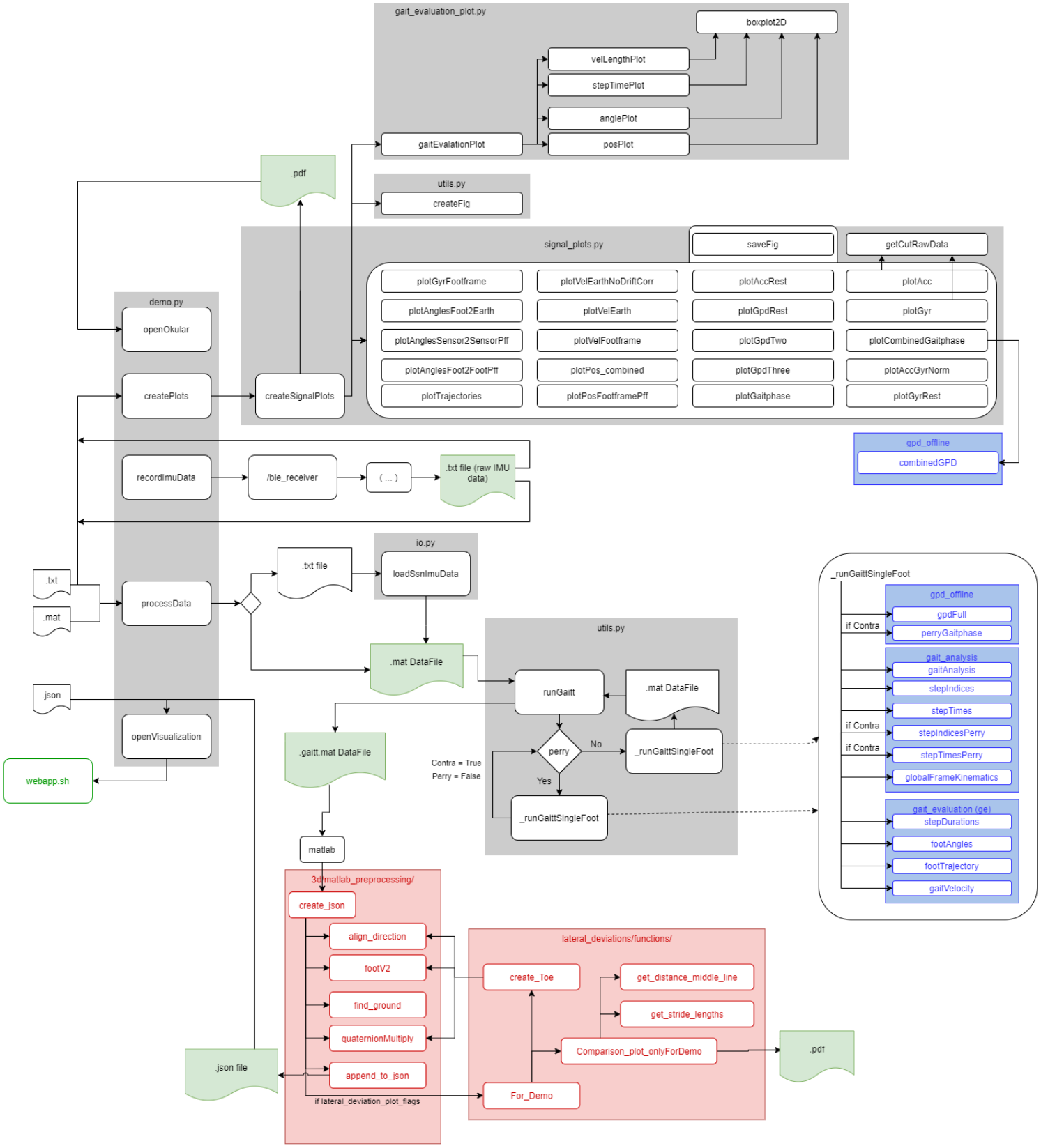

FIGURE 4.1: Global project's code flowchart. Python code is coloured grey, C/C++ code blue, and Matlab® red.

- IMU data streams: It records data received on real time from active IMUs.
- raw IMU data files: recorded information from a IMU trial in .txt format.
- .mat data files: recorded information already saved in an appropriate .gaitt.mat DataFile.
- .json files: Already processed information containing all necessary data for simulation in a .json file.

Regardless of the input, the demo will run all necessary scripts and will end with the simulation, and save important files while doing so. Therefore, all available inputs beyond the first one are also byproducts of the demo, which saves time when doing re-runs (so, for example, when entering a .txt file, it will create a .json file as well during the demo, which can be used as an input too on another run).

The code itself deals mostly with receiving the inputs and preparing them so the data they contain to those required by the GPD and gait analysis algorithms, and later on for the 3D visualization on Matlab®. It is in charge of retrieving this information too and saving it into appropriate files (.gaitt.mat, .json, or others); and plotting and saving some variables of interest contained in this data. These plots are saved in a .pdf file too so they are easier to access and read. For more information regarding the functioning of any of the existing functions, refer to A.1.

4.1.3 The C/C++ code

The C/C++ code is the implementation of the algorithms presented in Chapter 2: the GPD and gait analysis. The overview of the C/C++ code can be observed in Figure 4.2, along with all the useful outputs. This schematic represents the most important functions called, this time in a sequential manner (so, gait_analysis is called after gpdFull, not within it). For the sake of readability, constant inputs (such as the sampling rate) have been omitted, and some arrows have coloured (they have no particular meaning). Versions of the functions that include the Perry gait phases, instead of the 4 originally defined in this project, have been omitted too, since their functioning, besides the mapping, is mostly identical.

Figures 4.3 and 4.4 represent detailed flowcharts for the greatest functions among the C/C++ code. The former depicts the GPD algorithm, with the corresponding sub-algorithms (GPD One, GPD Two and GPD Three) remarked; and the latter shows the gait_analysis function, which constitutes a great part of the gait analysis tools used. As before, both are in sequential structure and coloured arrows are just to make it easier to follow them (a function inside another one indicates that it is called within the larger one). Moreover, the output variables are enclosed in red rectangles. Those will be the ones retrieved by the Python code too. For more information regarding the functioning of any of the C/C++ code, refer to A.2.

4.1.4 The Matlab® code

The Matlab® is in charge of preparing 3D visualization of the gait. To do so, it takes all the already processed gait data. It positions appropriately key foot points, creates the mesh and rotates it to a natural position. It also corrects errors that may have appeared during processing, such as drift in walking, so the visualized process

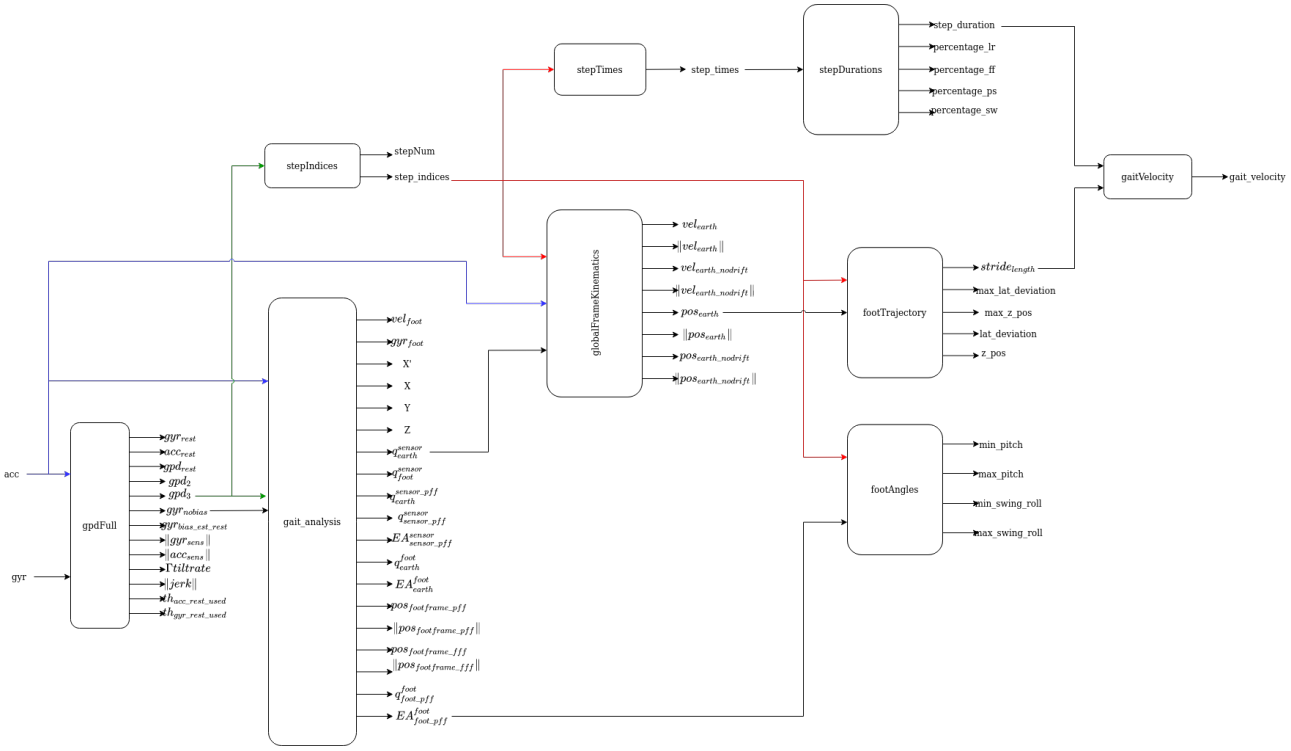

FIGURE 4.2: C/C++ code flowchart.

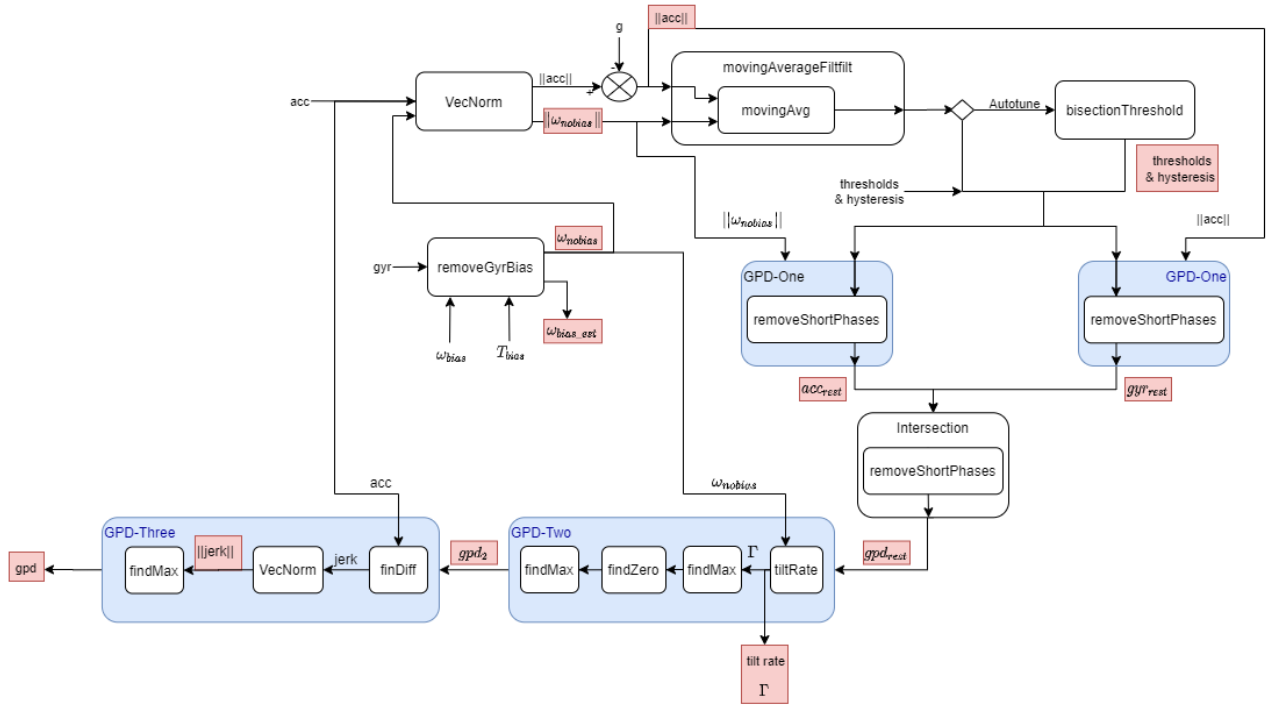

FIGURE 4.3: Detail of gpdFull function, from C/C++ code flowchart.
Implementation of GPD algorithm.

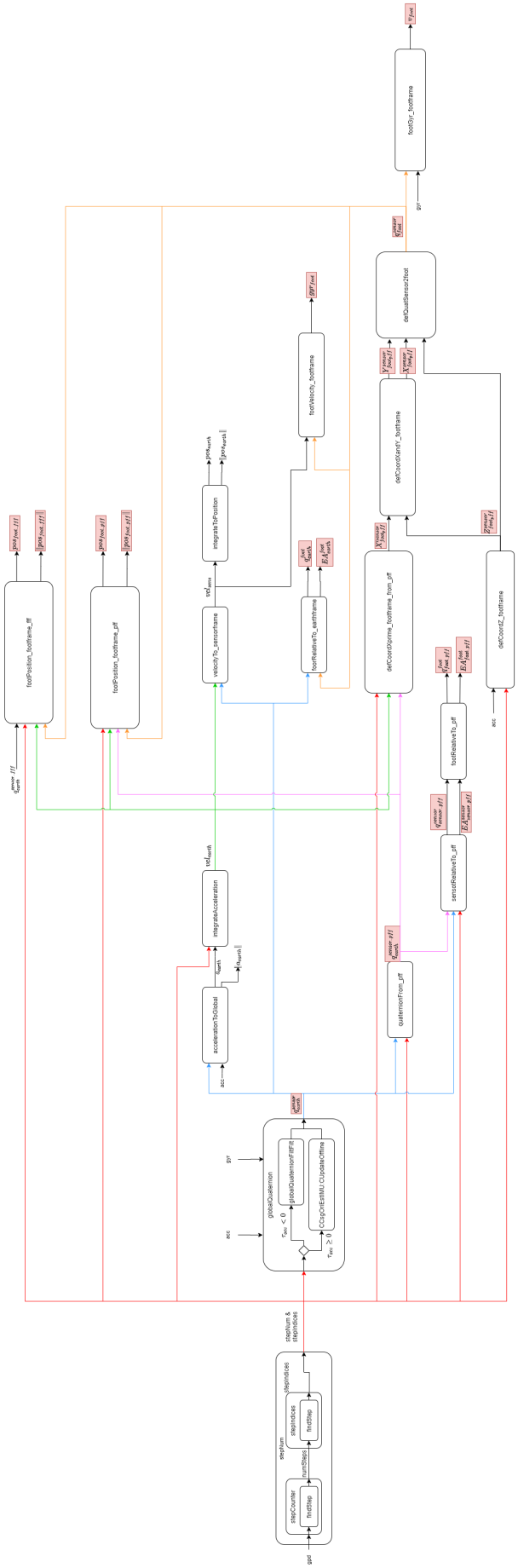

FIGURE 4.4: Detail of gait_analysis function, from C/C++ code flowchart. Greater part of the gait analysis.

matches a regular gait pattern. Finally, it creates the .json code where this whole data is stored (the foot, the gait itself, and the elements for visualization). Since there are fewer functions in the this piece of code, and the relationships between them can already be observed in 4.2, Figure 4.5 represents the inputs/output of the main key Matlab® functions used. In A.3 there is a more detailed description of the main Matlab® code used.

There is a change of notation in the transition from C/C++ to Matlab, and all variables are now separated by foot (rather that retrieved directly from the corresponding foot from the DataFile):

- step_indices → step_ind_left (or right)
- gpd array → gait_phase_left (or right)
- quat_foot2earth → foot_middle_orientation_left (or right)
- pos_earth →sensor_position_left (or right)

Moreover, some functions need inputs from only one foot or from both of them. Whenever only one foot is passed as input/output, it has been marked as L / R or [...]_l_[...] (/ r); whereas if both feet are passed, it has been marked as [...]_left (& right). The same is true for functions called within other functions: if they are called twice, once for each foot (L / R) or only once and take both feet as arguments (L & R).

4.2 Implementation of new rest detection method

The rest gait detection method used was presented already in section 2.2.1. It used the norms of the gyroscope and accelerometer readings to detect rest phases. Here, a newer method is implemented that uses the velocity on earth frame, vel_{earth}. The method itself was developed by the Control Systems group at TUB, and has been taken from [16].

4.2.1 Velocity in the earth frame

The first step is transforming the accelerometer readings, that are in sensor frame, to the global earth frame. It is done with the quaternion q_{earth}^{sensor}, found from the orientation estimation algorithm introduced in Section 2.3.6, and Equation 2.23, repeated here for the sake of comprehension:

$$q_{acc\_earth}(t) = q_{earth}^{sensor(t)} q_{acc\_sens}(t) q_{sensor(t)}^{earth} \tag{4.1}$$

Once in the earth frame, the effect of the acceleration due to gravity is subtracted. Then, this new acceleration can be integrated to achieve the desired velocity.

4.2.2 High- and Low-pass filtering

The resulting velocity is not yet suitable to operate with. First, it is filtered to eliminate the bias from the acceleration. This high-pass filter is tuned so that it keeps the high frequency data while erasing the low frequency drift. Then, a low pass filter is applied too to smooth the velocity peaks.

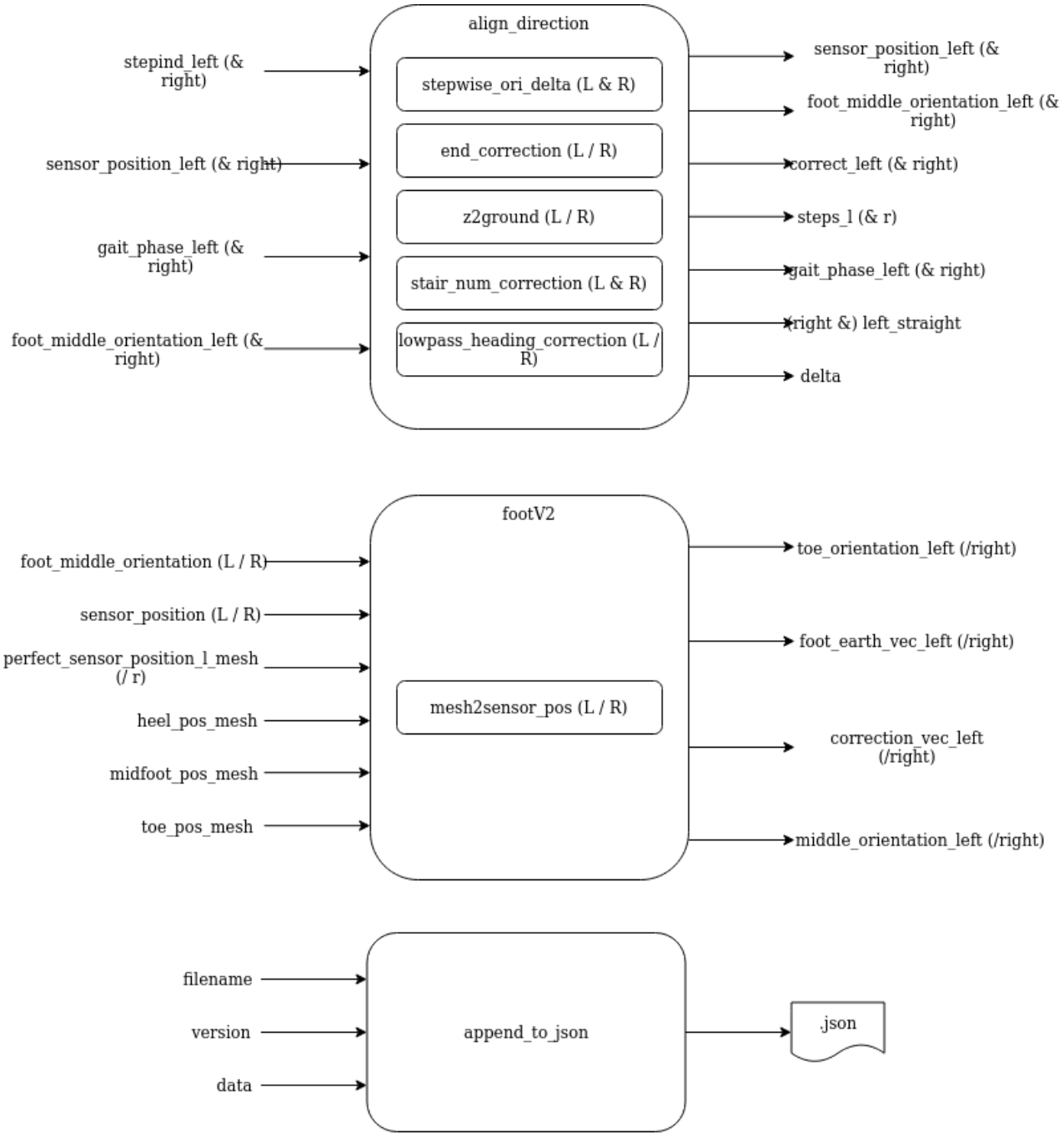

FIGURE 4.5: Main Matlab® functions used for 3D visualization

4.2.3 Intelligent bias removal

The velocity is now adequately filtered and in the correct frame. To determine the rest phases, there has to be a clear difference of the velocity values between movement and rest phases across all three velocity coordinates. Thus, it is crucial to eliminate this bias so the aforementioned difference can be observed. This is done in three steps with an intelligent bias removal algorithm.

On the first step a moving average is created by the use of a highly smoothing low pass filter, forwards and backwards. This double passing ensures that the time offset due to the low-pass is neutralized. The resulting moving average does not lie on the rest phase due to the speed fluctuations of the steps, but it is offset from it.

On the second step, a sliding time window is used to find whether that speed offset is above of below the moving average. This sliding time window ensures that the fluctuations are observable even when the subject changes directions, as they will appear on a different axis or there will be a zero crossing.

On the third step, the resulting signal is composed by the velocity on the rest and motion phases. This new signal is clean of noise, drift and also the velocity fluctuations due to the steps. When the three steps are repeated once more on this recently obtained signal, the final moving average closely resembles the actual rest phase: in the obtained signal, rest phases will have values near 0 with very small deviations.

4.2.4 Grouping and normalization

The velocity in x and y axes, that correspond to the horizontal plane, are combined together into a single axis by means of the Euclidean distance (which is also, by definition, positive semidefinite). To the remaining axis, z, is applied the Euclidean norm, removing the negative values as well. Finally, both signals velocities are normalized into to the $[0, 1]$ range - this avoids amplitude issues and allows using a single constant threshold -.

4.2.5 Rest phase detection

These last two signals are, once again, low-pass filtered forwards and backwards to remove any kind of disturbances caused by trembles in walking. To these two resulting, final signals, a threshold is applied. Whenever a the signal below the threshold, a rest phase is detected, marked as 0; and greater values, corresponding to motion phases, are marked as 1. This yields a binary signal with where every sample is classified as either rest (0) or movement (1), just like the original GPDOne.

4.3 Parameter optimization on orientation estimation algorithm

Previous sections have dealt with ways to compute and how to improve gait analysis and evaluation data with the existing working framework. However, the quality of the processing algorithms has not yet been discussed. In this section, the orientation estimation algorithm of the IMU will be tuned. Having a well oriented sensor is crucial because many algorithms used later on depend on it, either directly using

FIGURE 4.6: Setup of IMU and optical markers during the trials

this orientation or indirectly when translating coordinate frames using quaternions derived from it.

Besides the orientation estimation algorithm explained in section 2.3.6, there is another available option. This second orientation estimation algorithm is not the default one, however, it still can be useful. Within the work of this thesis is the optimization of the τ_{acc} and acc_{Rating} parameters used by this algorithm.

4.3.1 Framework

To assess the goodness of results, a reference system must be used to compare the orientation to a ground truth. In this case, there were two: an instrumented treadmill and optimal markers for motion tracking. The latter consisted in optical markers mounted at several places of the foot. Among them, there were three placed over each IMU mounting, which allowed to capture the position and orientation of the IMUs and, thus, set the basis for a comparison. The instrumented treadmill cannot be used to compute orientations, but it will be useful later on to check the implementation of this algorithm; since it can provide velocity or stride length measure to compare. An exemplary photo of the IMU and optical markers setup is shown in Figure 4.6.

4.3.2 IMU inclination

The IMU readings used are the acceleration and gyroscope, the magnetometer is purposely left aside so that the orientation estimation can still be used even under the presence of non-homogeneous magnetic fields, such as in clinical environments. The gyroscope readings' bias is removed and the result is used to synchronize the optical and IMU measurements.

To judge the impact of the τ_{acc} and acc_{Rating} parameters, the chosen measure has been the error between the inclination measure by the IMU and the one measure by the reference system. First, the quaternion ${}^{IMU}q^{sensor}_{earth\_sensor}$ is computed by the orientation estimation algorithm , which translates sensor frame to the global earth frame; and the quaternion ${}^{opt}q^{opt}_{earth\_opt}$, which translates the sensor frame obtained from the three optical markers on the IMU to the global earth frame. The left superscript indicates the capture system used to obtain this orientation, whereas the *earth_subscript* indicates the global reference frame, which may not be equal for both systems. Therefore, the the following operation is necessary in order to have a common ground across both systems:

$$ {}^{opt}q^{sensor}_{earth\_sensor} = q^{earth\_opt}_{earth\_sensor}\ {}^{opt}q^{opt}_{earth\_opt}q^{sensor}_{opt} \tag{4.2} $$

Where $q^{earth\_opt}_{earth\_sensor}$ is the quaternion that relates both global reference systems and q^{sensor}_{opt} relates their inertial frames (in fact, the quaternion computed is q^{opt}_{sensor}, but by quaternion properties, $q^{sensor}_{opt} = (q^{opt}_{sensor})^*$). Hence, there is a quaternion for the IMU inertial frame that translates it to the same global earth frame, captured by the two different kind of sensors.

Since the IMU and the optical capture system may have different sampling rates, the obtained quaternions $q^{opt}_{earth\_opt}$ - one for each sample - are interpolated so that their timestamps match the ones obtained from the IMU data.

Finally, these quaternions are multiplied to compute the rotation between both frames (q_{diff}), the one obtained from the IMU sensor and the one from the optical markers system, as in the equation below:

$$ q_{diff} = {}^{IMU}q^{sensor}_{earth\_sensor}\ {}^{opt}q^{sensor}_{earth\_sensor} \tag{4.3} $$

The inclination component of this rotation constitutes the difference in the inclination computed by the IMU sensor and the optical markers system or, in other words, the error committed by the IMU sensor when taking the optical markers measure as the ground truth reference. Thus, there is a measure for the goodness of orientation estimation algorithm and the impact of the τ_{acc} and acc_{Rating} parameters. The inclination is computed as:

$$ inclination = 2 \cdot \arccos\left(\sqrt{q^2_{diff_3} + q^2_{diff_0}}\right) \tag{4.4} $$

Where q_{diff_i} represents the i^{th} component of $q_{diff} = [q_{diff_0}, q_{diff_1}, q_{diff_2}, q_{diff_3}]$.

4.3.3 Parameter optimization

The optimization is performed for each trial (see Section 5.1 for more information on this), but the parameters will not be changed for each of them. Since the goal of the optimization is finding a good combination of both parameters that fits all trials, the results from all the optimizations will be analyzed and discussed to find adequate τ_{acc} and acc_{Rating} without compromising the orientation obtained on each one.

The optimization is performed in two steps. First, an heuristic search is done, with the aim of finding good initial values for the numerical optimization method used after. In this search, lower and upper bounds and step increments are specified

for each parameter; and a value grid is constructed from it. Then, for each possible τ_{acc} and acc_{Rating} combination within the grid, a cost function is computed. Afterwards, an optimization problem solver is run, having as method L-BFGS-B and as initial value the solution from the heuristic search. The cost function is the same for both methods: they try to find the minimum inclination RMSE.

Chapter 5

Experimental results

In this chapter, the results from the algorithms presented in Sections 4.2 and 4.3 will be presented. The trials performed, the evaluation method and plots displaying the results will be discussed.

5.1 Experimental framework

The data set used for results' assessment has been the same in both experimental cases. This data set was recorded and provided by FH Joanneum Graz. It consists on 92 trials of patients (students) walking on a instrumented treadmill for a few minutes. Having a treadmill that is able to record gait data too is useful because it creates a reference system against which it is possible to compare gait analysis and evaluation results. The trials are grouped in 23 groups of 4 categories, depending on speed and movement capabilities:

1. Walking at 1,5 km/h.
2. Walking at 3 km/h.
3. Walking at 3 km/h with right knee flexion blocked.
4. Walking at 5 km/h.

All subjects had an IMU attached over each foot, which captured the data to be tested. Besides the IMUs, there were also optical markers placed on the foot and, concretely, 3 atop each IMU. This allows creating another reference system with which compare the position and orientation of the IMU itself, rather that gait analysis results. This also allows assessing whether the results derived from IMU data can be considered meaningful or, on the contrary, the captured data was not accurate enough.

The technical details of the equipment used are the following:

1. Instrumented Treadmill: Zebris Rehawalk:
   - Treadmill: h-p-c Mercury Med Treadmill, walking speed: 0–22 km/h in 0,1 km/h steps, walking surface: 150 cm x 50 cm.
   - Pressure measuring platform: FDM-THM-M-3i / 120 Hz, sensor area: 108.4 cm x 47.4 cm, 7168 sensors.
2. IMU: PABLO® by Tyromotion GmbH, Graz, Austria
   - Rate: 110 Hz.

- Dimensions: 56 mm x 34 mm x 21 mm.
- Bluetooth Wireless data transmission.

3. Optical sensors: Vicon.
   - Rate: 120 Hz.

5.2 New rest detection method

To assess the impact of the new rest detection method, the average duration of the four gait phases (in percentage), the average stride length and the average gait velocity were chosen. They have been compared against the ones obtained with the previous rest detection method, GPD-One. To compare them, they have been plotted together in scatter plots; and, as an indicative measure, the bisection of the first quadrant has been plotted too. The results can be observed in Figures 5.1 and 5.2.

Since this comparison might shed some light into the differences between these methods, but not their actual accuracy, it is also presented the comparison against a reference system, in this case, the Zebris data from the instrumented treadmill. Results are presented in Figure 5.3 on a Bland Altman plot. This kind of plot displays the average gait phase duration (in percentage of the step duration) for each trial in the horizontal axis, and the difference between them and the gait duration measured with the reference system on the vertical axis. Also displays the overall average value (thick grey line), and the 95% confidence intervals (1.96 times the standard deviation, thinner grey lines). Values over the overall average are overestimated whereas under it are underestimated.

5.2.1 Discussion

The new rest detection method presents differences with respect to the original one, GPD-One. It can be observed in Figure 5.1 that the rest and swing phases duration are shorter when computed with the new method (as it is below the bisection of the first quadrant); whereas the pre-swing phase is longer. The only phase in which they have similar length is the loading response phase.

The reduction in the rest phase duration and the increment in the pre-swing phase share the same explanation. Recall Section 2.2.1: the GPD-One algorithm first created the division between rest phases and motion phases, preemptively marked as pre-swing before further classification. Therefore, the new method seems to detect the motion before the GPD-One, which leaves a shorter rest phase and a longer motion phase. It is noteworthy how, the longer the pre-swing phase (and shorter rest phase), the most notable the differences between both systems are. Also the new method provides a wider range of these gait phases duration than the original one, seen by its higher dispersion across the vertical axis (values within $[0.3, 1.5]$ seconds in pre-swing phase) compared to the horizontal one (values within $[0.2, 1.1]$ seconds in pre-swing phase).

The explanation behind the decrement in time of the swing phase duration is not so clear. It is surprising how the swing phase remains almost constant across all trial when computed with the new method (almost all values around 0.2 seconds),

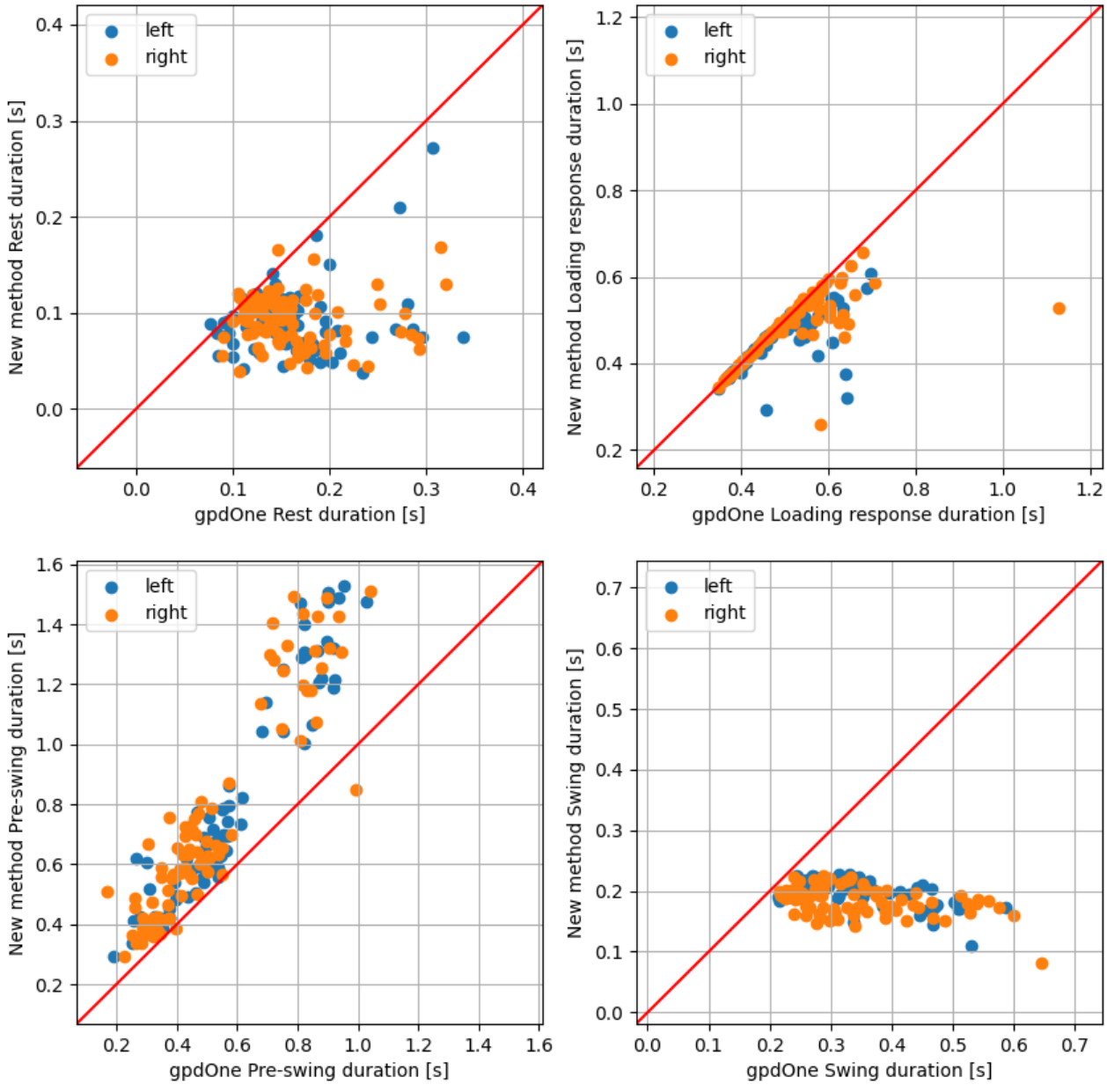

FIGURE 5.1: Scatter plot of gait phases duration (in %), comparing the new rest detection method against GPD-One. Bisection of the first quadrant marked in red.

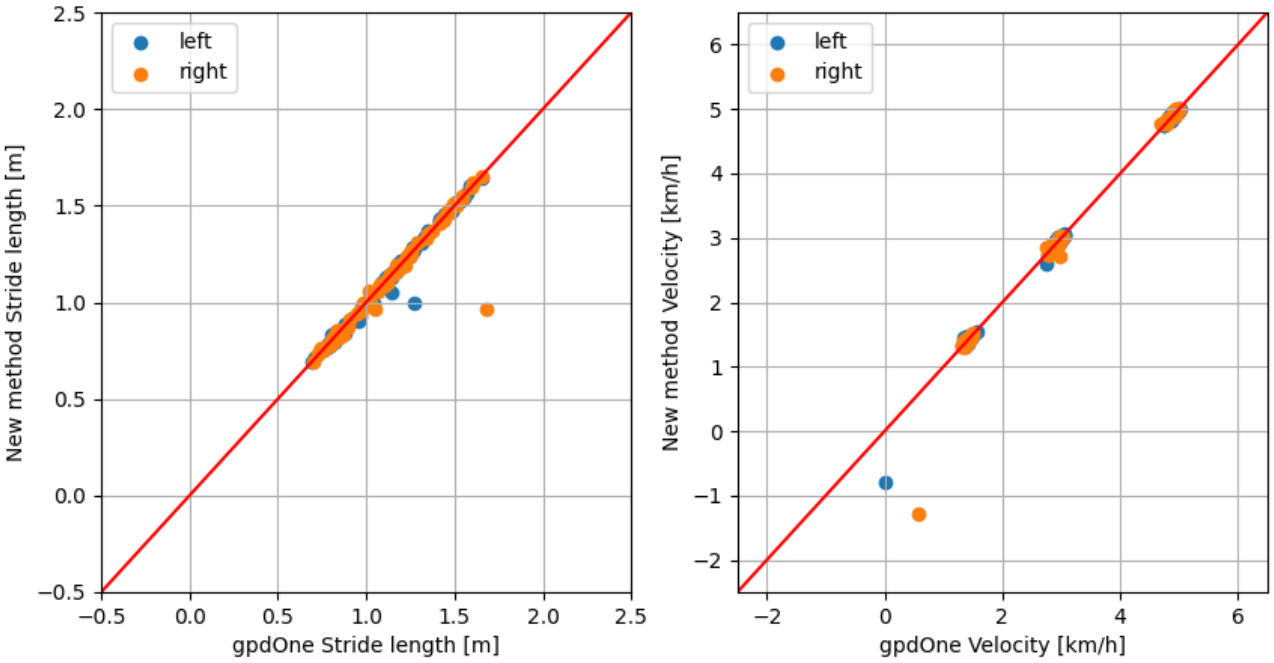

FIGURE 5.2: Scatter plot of stride length and gait velocity, comparing the new rest detection method against GPD-One. Bisection of the first quadrant marked in red.

whereas there is much more variability when using GPD-One. This is remarkable because the subjects were walking at different velocities across all trials.

Regarding the stride length and the gait velocity, there is no difference with respect to the original method of rest phase detection. The three velocities at which the subjects were forced to walk are clearly visible too.

Finally, looking at the last Figure 5.3, some conclusions can be drawn too. First, that there are not really big differences across both gait rest detection methods, results are almost identical in stance and pre-swing duration; so differences spotted before might actually be negligible. The loading response phase is slightly overestimated in some cases with the new method, the GPD-One presents overall better results. On the contrary, the swing phase is sometimes underestimated when using the new method, whereas, as before, the original method has more compact results. Overall, the average difference between the reference system Zebris and either method is close to 0%, with the greatest confidence intervals being $\pm 5\%$ of the average value.

5.3 Parameter optimization on orientation estimation algorithm

As stated in Section 4.3, the other available orientation estimation algorithm needs tuning of the τ_{acc} and acc_{Rating} parameters. The goal was to achieve inclination RMSE results of around 4º at most to consider them as *good* results.

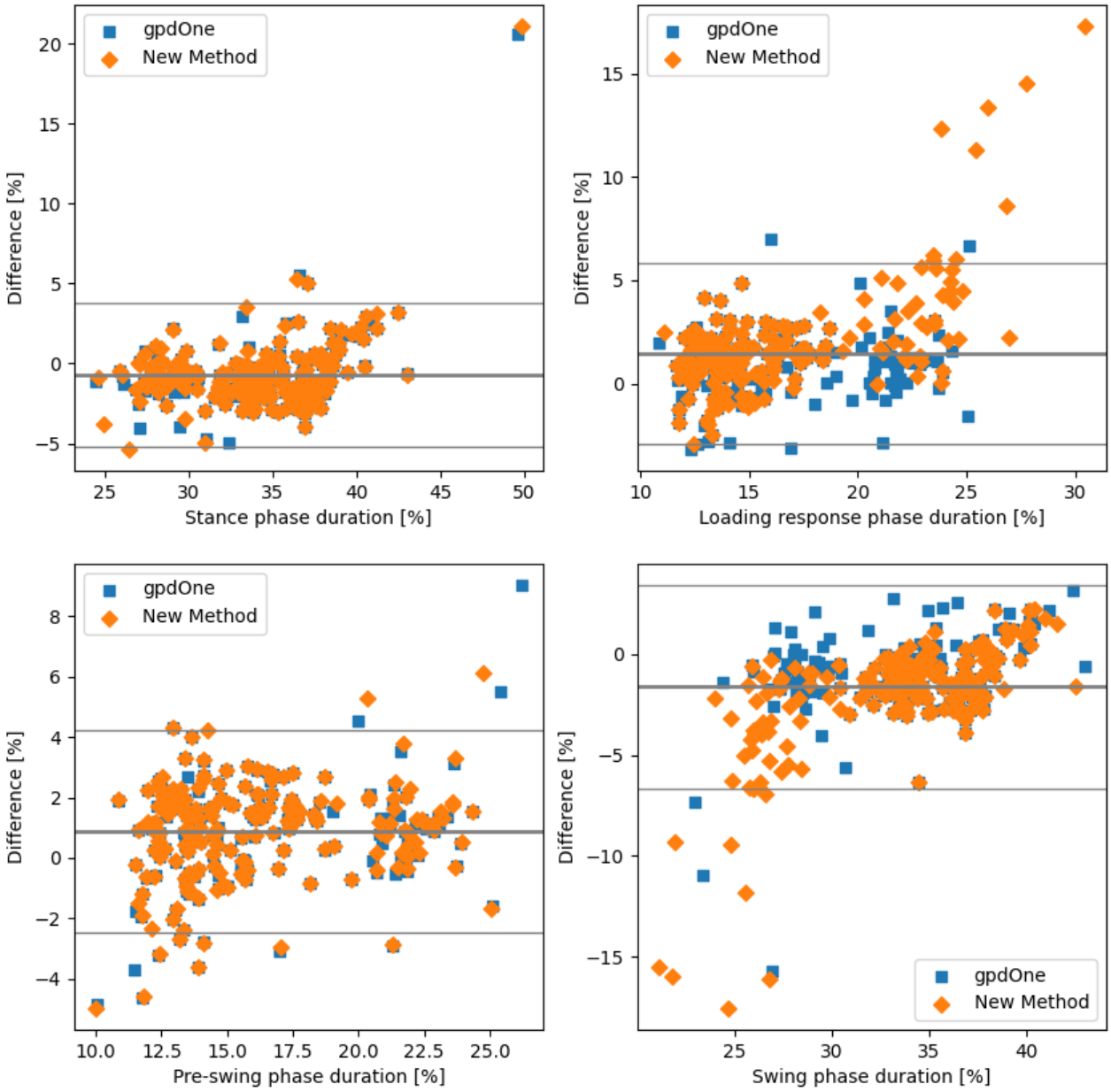

FIGURE 5.3: Bland Altman plot of gait phases' duration of both rest detection method against ground truth reference system Zebris.

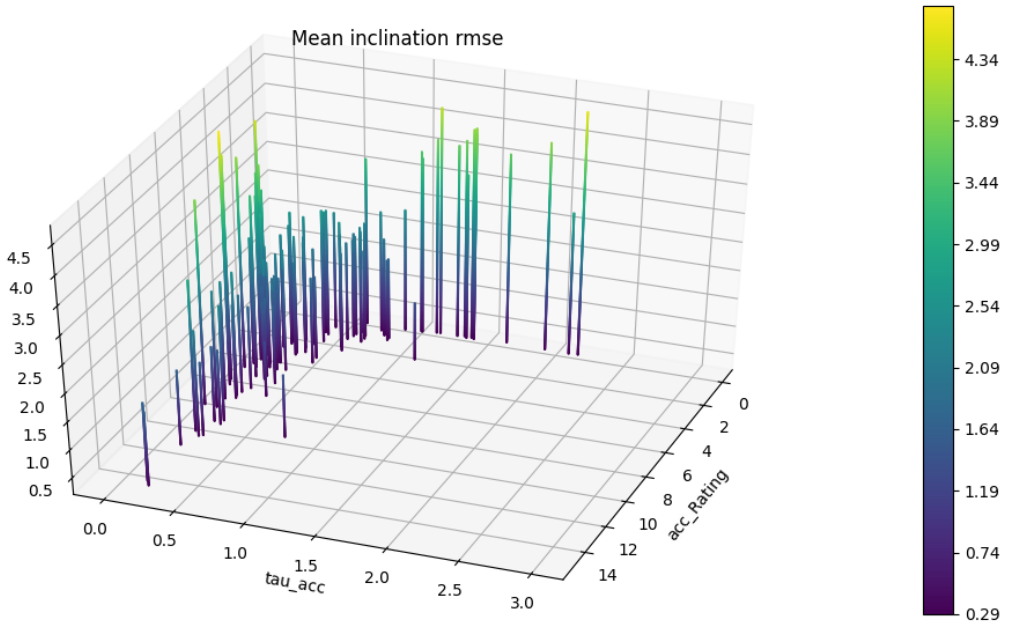

FIGURE 5.4: 3D Bar plot of the optimal τ_{acc} and acc_{Rating} parameters and the inclination RMSE (in degrees). Their values are in the horizontal grid, the inclination RMSE is the height of the bars.

5.3.1 Exploratory analysis

Firstly, as an indicative approach, the optimal parameters and the associated inclination errors of each trial were plotted all together in Figure 5.4. It is a 3D bar plot where the horizontal axes are the τ_{acc} and acc_{Rating} parameters; and the height (and colour) of the bars show the inclination error. It is possible then to see in which areas of the grid do the lower (the desirable ones) or higher inclination RMSE results concentrate.

5.3.2 Parameter tuning

Seeing the high variability of the acc_{Rating} parameter - that will be discussed later on on Section 5.3.4 -, it was deemed more appropriate to focus on the other parameter, τ_{acc}. Therefore, the acc_{Rating} was fixed to 4 (an even number around the average optimal value) and different values of τ_{acc} around 0.1 were used to compute the inclination RMSE. The τ_{acc} values were the range [0.09, 0.16] and the more extreme values 0.01 and 0.5. Since there are many trials and the parameter range is wide, the results were plotted on Figure 5.5.

The aforementioned figure is a box-plot. This kind of plot displays in a box the values within the 25% - 75% interval, each whisker represents the lower (0% - 25%) and upper (75% - 100%) quartile, and the average value is marked with an orange line inside each box. Outliers are represented by dots outside. This, it is possible to see the average value, the four quartiles (and thus, have an idea of the dispersion and min/max values) and the number and value of outliers. In the horizontal axis

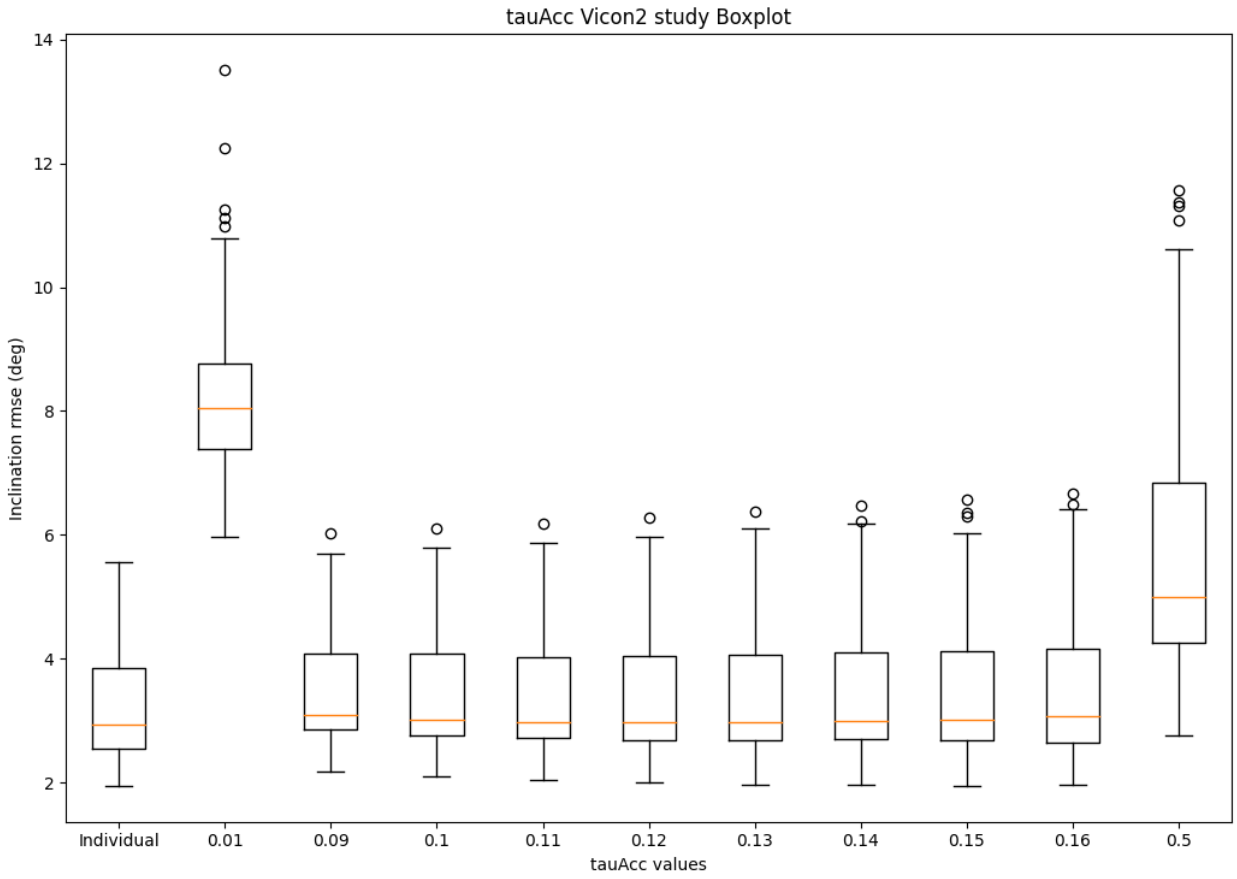

FIGURE 5.5: Box-plot of the inclination RMSE for different τ_{acc} values. The *Individual* refers to the optimal case (each trial with optimal parameters), and acc_{Rating} was fixed to 4.

are marked the τ_{acc} values associates to each box (*Individual* refers to the optimal case), and in the vertical axis is the inclination RMSE in degrees.

Besides the stated τ_{acc} and acc_{Rating} values, the optimal case was represented too. This encompasses the values obtained from the optimization solution and, thus, the parameters' values change for each trial. This was included only as a reference of the best possible case and as a framework for comparison.

When τ_{acc} is set to negative values, the alternative orientation estimation algorithm is selected. In that case, instead of τ_{acc}, the parameter is a Time Constant. As a measure of comparison, the same plot has been created for several Time Constant values, whose results are in Figure 5.6.

5.3.3 Implementation of the changes

The orientation estimation algorithm with the newly tuned parameters was tested against the same algorithm with old parameters, and against the alternative orientation estimation algorithm available.

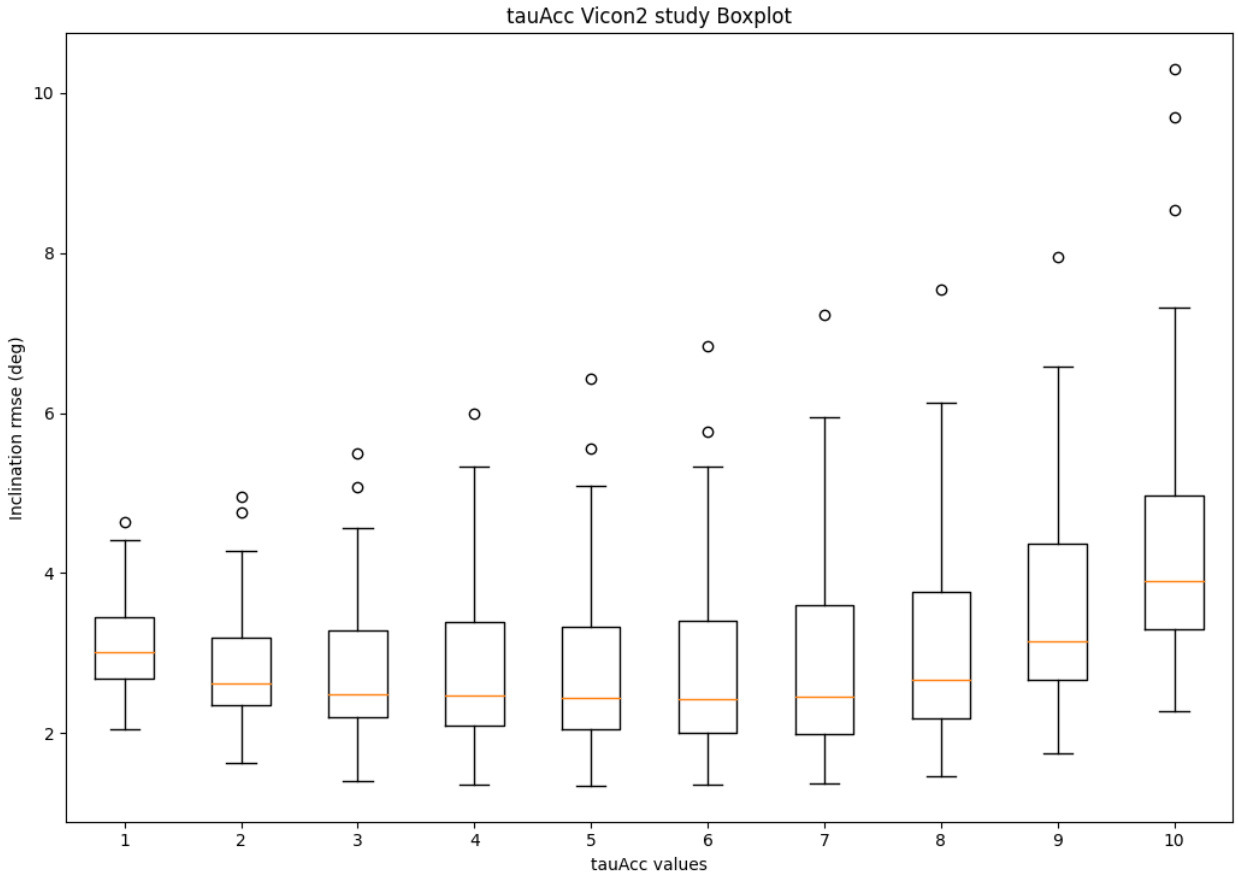

FIGURE 5.6: Box-plot of the inclination RMSE for different τ_{acc} values, computed with an alternative orientation estimation algorithm. The markings on the x-axis refer to the time constant used

Figure 5.7 shows the Euler Angles of the q_{earth}^{sensor} quaternion, computed with the three different possibilities aforementioned. This is the result of the 18^{th} trial at 5 km/h, which is considered a representative one. Since it is hard too see the details in those plots, Figures 5.8, 5.9 and 5.10 provide close-ups of selected regions that were considered interesting and meaningful.

Afterwards, and as as final evaluation, the three methods were also compared one against the other taking as measures the average stride length and the average gait velocity (the gait phase duration is not affected by the changes in this section, thus, comparing them would be useless). The results are presented in a scatter plot in Figure 5.11.

5.3.4 Discussion

Parameter tuning

The first conclusion the plot on Figure 5.4 is that the inclination RMSE values all are below 4.34^{o}, which are considered good results. Even though these are the best possible results, there is still some slack.

Also, it can be seen that the optimal results spread almost all over the acc_{Rating} parameter studied range [0, 15], whereas they are much concentrated on the τ_{acc} range: most of them are below the 0.5 mark, and almost all below 1. Therefore, it seems that the acc_{Rating} parameter has a lesser impact on achieving the minimal inclination RMSE, whereas τ_{acc} is restricted to a narrow interval. Hence, efforts were centered on tuning the this last parameter.

The plot on Figure 5.5 represents the values studied for the tuning. It is seen how results within a small τ_{acc} interval ([0.09, 0.16]) are acceptable, but they diminish as soon as the parameter is either too low or too high. Within the acceptable interval, results are very similar. It appears that, the greater the value, the more dispersion there is: the average remains almost identical, the lowest value decreases very slightly, but the ceiling (maximum inclination RMSE) increases and more outliers appear. Moreover, all these results are just slightly worse than the optimal ones, which indicates that choosing any of those values for τ_{acc} still grants good results. Lower τ_{acc} are considered better because the negative impact of a higher inclination RMSE ceiling is deemed to outweight the positive impact of a lower inclination RMSE floor. Since there is very little distinction in the results obtained with the two smallest τ_{acc} , and to keep things easier, the chosen τ_{acc} value to implement has been 0.1. acc_{Rating} has been left at 4.

Regarding the results obtained with the alternative algorithm (Figure 5.6), it can be seen that there is a great variation depending on the value of the Time Constant used. For lower values (mainly $T = 1$, $T = 2$ and $T = 3$), when best results are achieved, inclination RMSE results are better than the ones obtained with the studied algorithm and tuned parameters: the average is below 3 degrees and the ceiling does not reach the 6 degrees mark, even when taking into account the outliers.

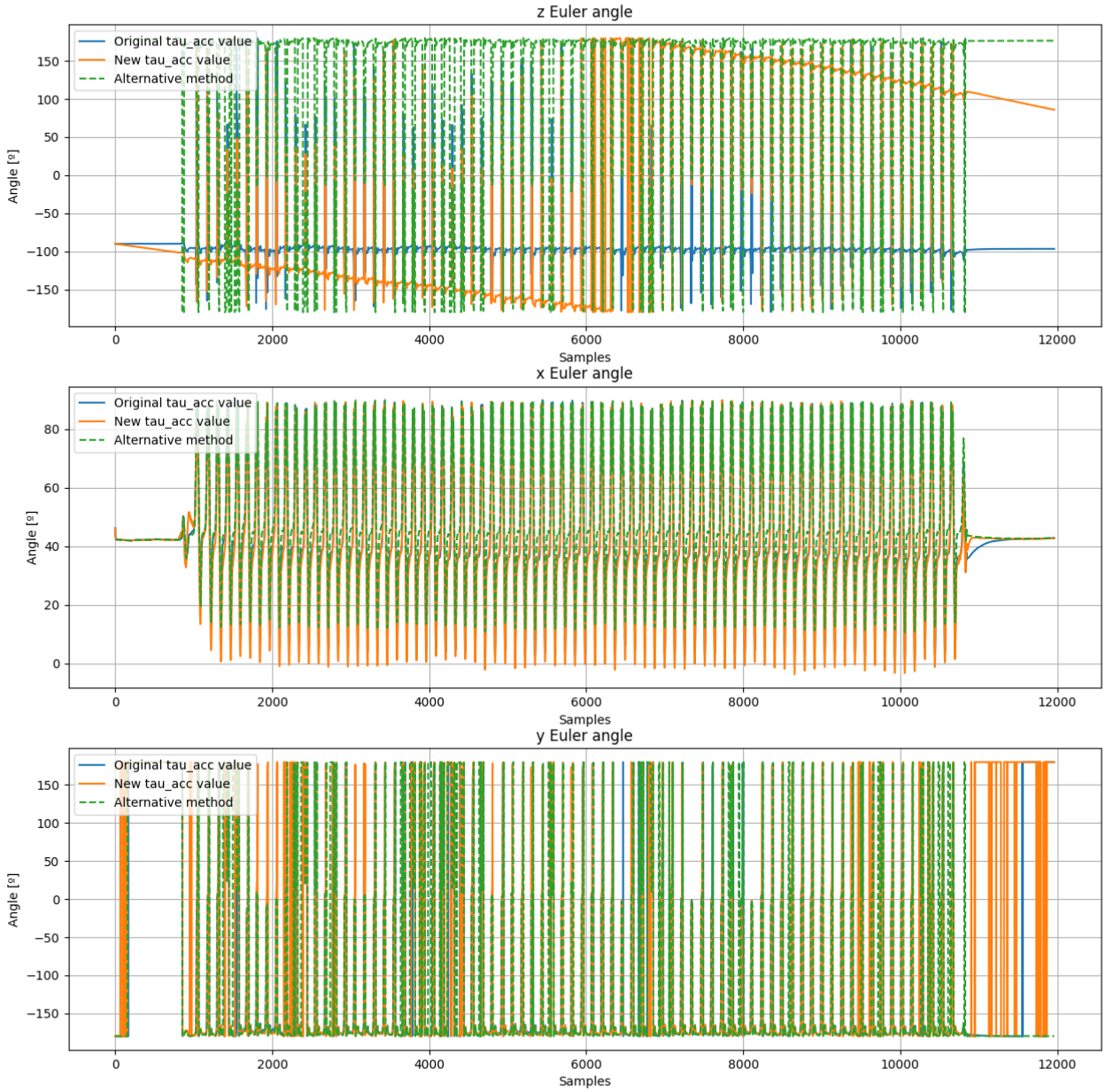

FIGURE 5.7: Euler Angles of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters

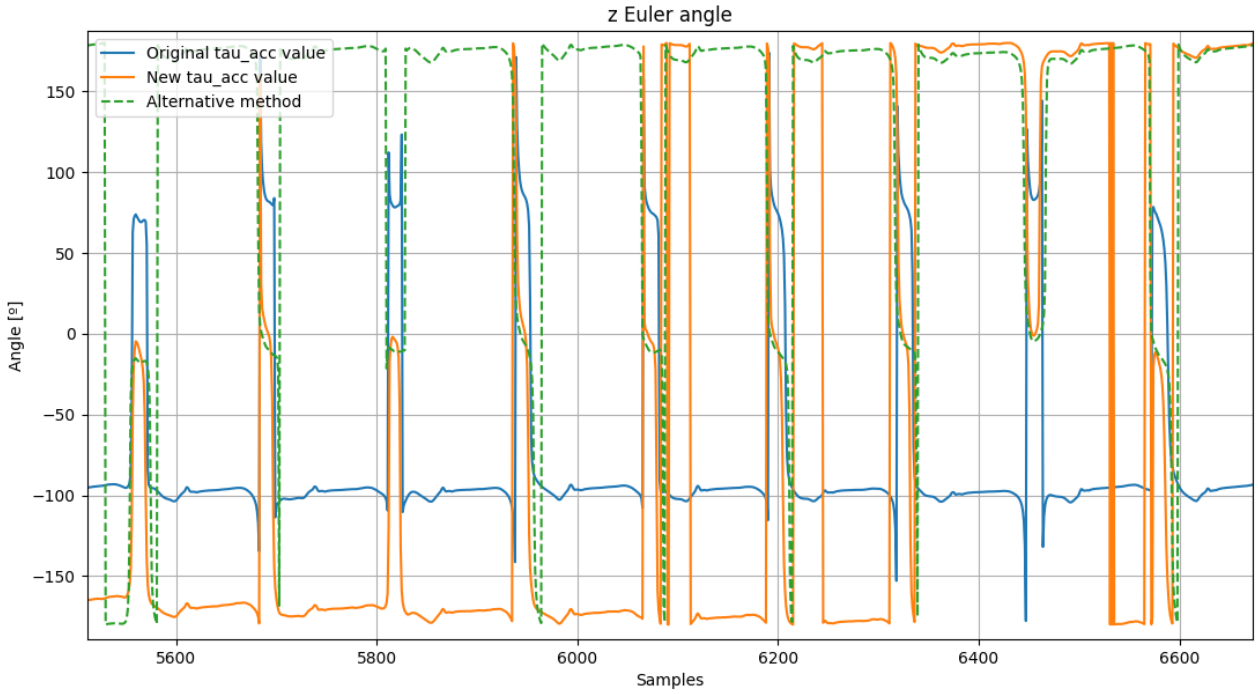

FIGURE 5.8: z Euler Angle of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters

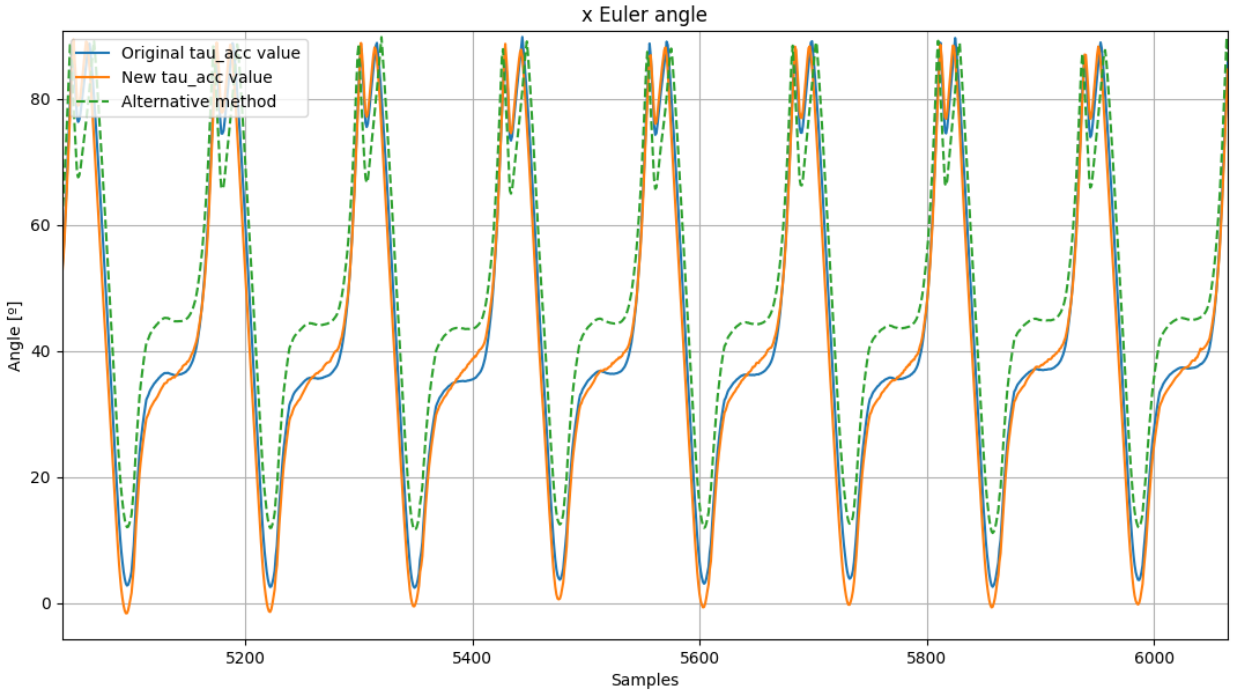

FIGURE 5.9: x Euler Angle of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters

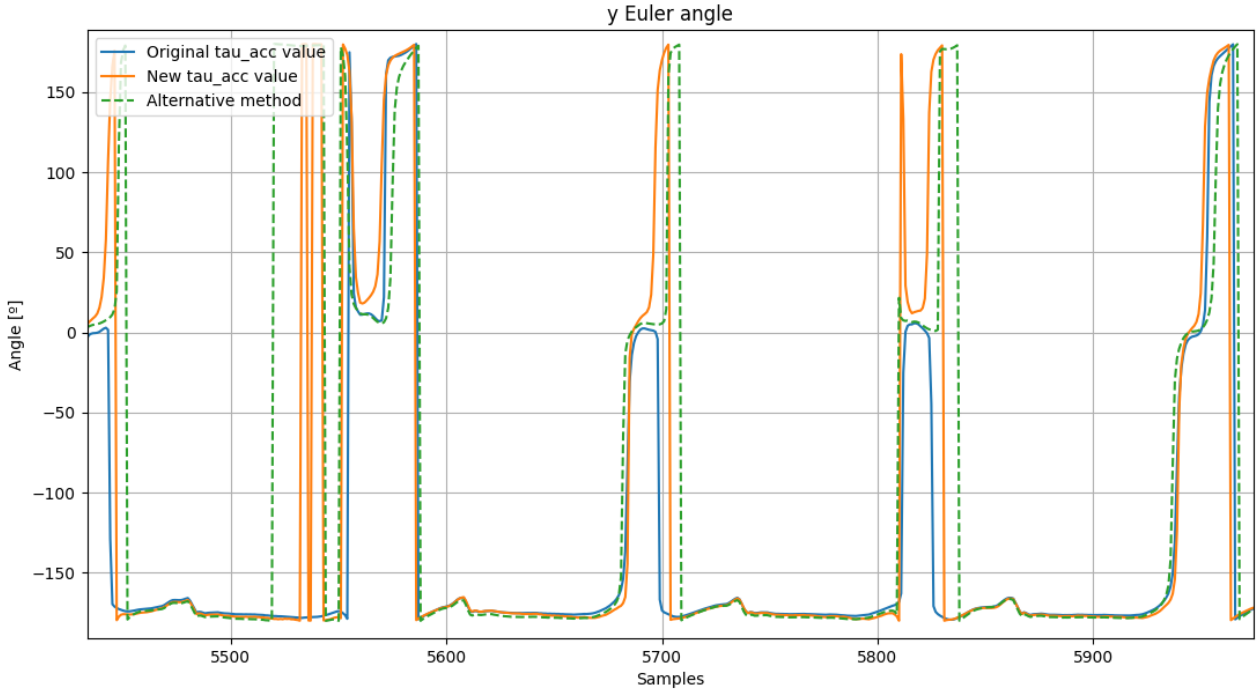

FIGURE 5.10: y Euler Angle of the q_{earth}^{sensor} quaternion computed with different orientation estimation algorithms and τ_{acc} parameters

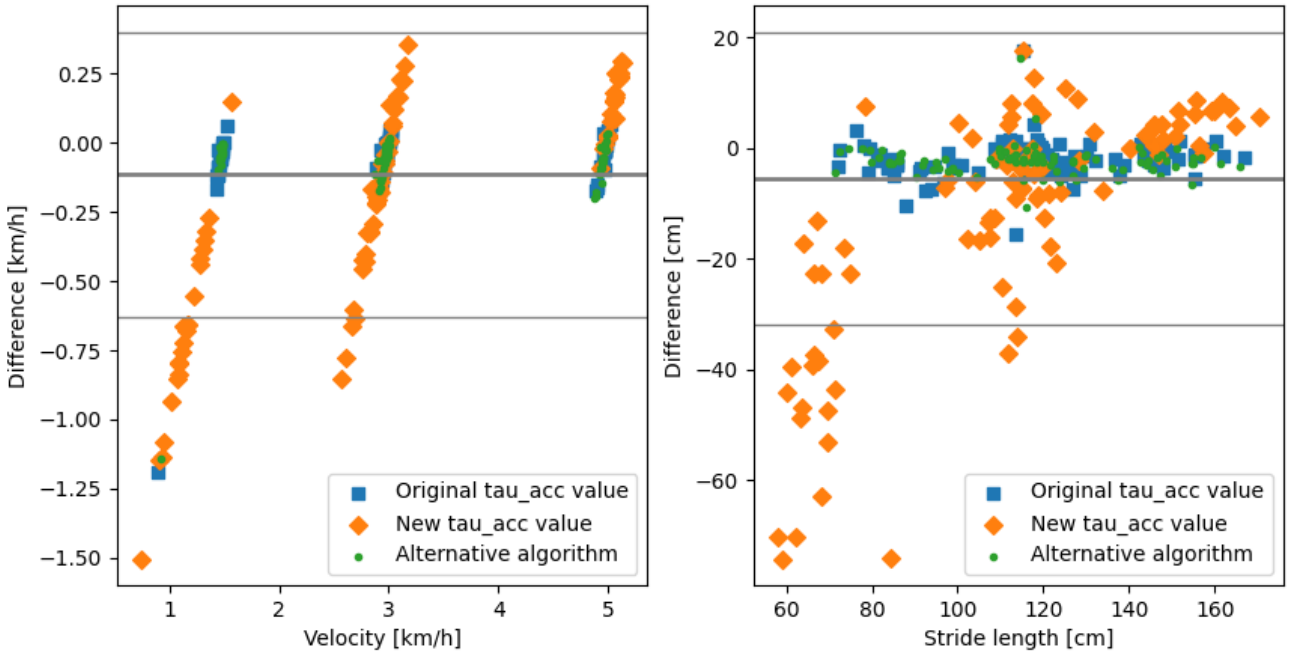

FIGURE 5.11: Bland Altman plot of average gait velocity and average stride length comparing the orientation estimation algorithms and τ_{acc} parameters.

Implementation of the changes

Since here all three algorithms will be discussed heavily, and to avoid mixing up them and easing the understanding of these conclusions, the algorithms will be renamed as follows:

- Algorithm 1: Original orientation estimation algorithm with original parameters.
- Algorithm 2: Original orientation estimation algorithm with tuned parameters ($\tau_{acc} = 0.1$ and $acc_{Rating} = 4$).
- Algorithm 3: Alternative orientation estimation algorithm.

The results from the implementation are far from expected. Plots on Figure 5.7 and its close-ups, Figures 5.8, 5.9 and 5.10, show that there is great disparity across the three options studied. The x Euler Angle is very similar on all three algorithms; however, the other two are vastly different. On z Euler Angle, Algorithm 1 and Algorithm 3 have a similar behaviour in shape, being the only difference that the latter acquires a wider range of angles. With Algorithm 2, it appears the there is some time of calibration in which it resembles Algorithm 1 and then, suddenly, the original algorithm tries to imitate Algorithm 3. Once it reaches that point, it steadily declines back to Algorithm 1 behaviour. Finally, on the y component of the Euler Angle the Algorithms present varying results. It seems that Algorithm 2 is the most regular with evenly spaced peaks, and Algorithm 3 displays a similar behaviour (but not so regular); whereas Algorithm's 2 tends to fall short often. All in all, taking Algorithm 3 as a reference, it seems that Algorithm 2 tries to match its results, but not quite achieving them.

When looking at the plots on Figure 5.11, it is clear that Algorithm 2 presents the worst results. Both Algorithm 1 and Algorithm 3 have differences near 0 on each measure (in their respective units), a much lower variation, and more similar values. What is more, the velocity of the treadmill on all trials was fixed to 1.5, 3, or 5 km/h. On the Velocity Bland Altman plot, both Algorithm 1 and Algorithm 3 have very precise points very close to those values, whereas Algorithm 2's results are more spread.

In conclusion: despite good results on inclination RMSE compared to an optical system and a hint of improvement looking at the Euler Angles of the quaternion q_{earth}^{sensor}, the actual results of its implementation are diminishing and give no reason whatsoever to used the tuned parameters on the orientation estimation algorithm.

5.4 Comparing left and right foot data

During the development of the previous Section, it was discovered that the data for the left and right foot presented differences. Since the chosen measure, the inclination RMSE, was computed for both feet simultaneously, the difference in the data could have affected these results. Therefore, the inclination RMSE was computed once again, but this time for each foot separately. For this analysis, the chosen τ_{acc} and acc_{Rating} values were the optimal ones. These results are shown in Figure 5.12. On the left, a box-plot shows the inclination RMSE results computed for each foot; and on the right, the absolute difference in inclination RMSE for each trial, along

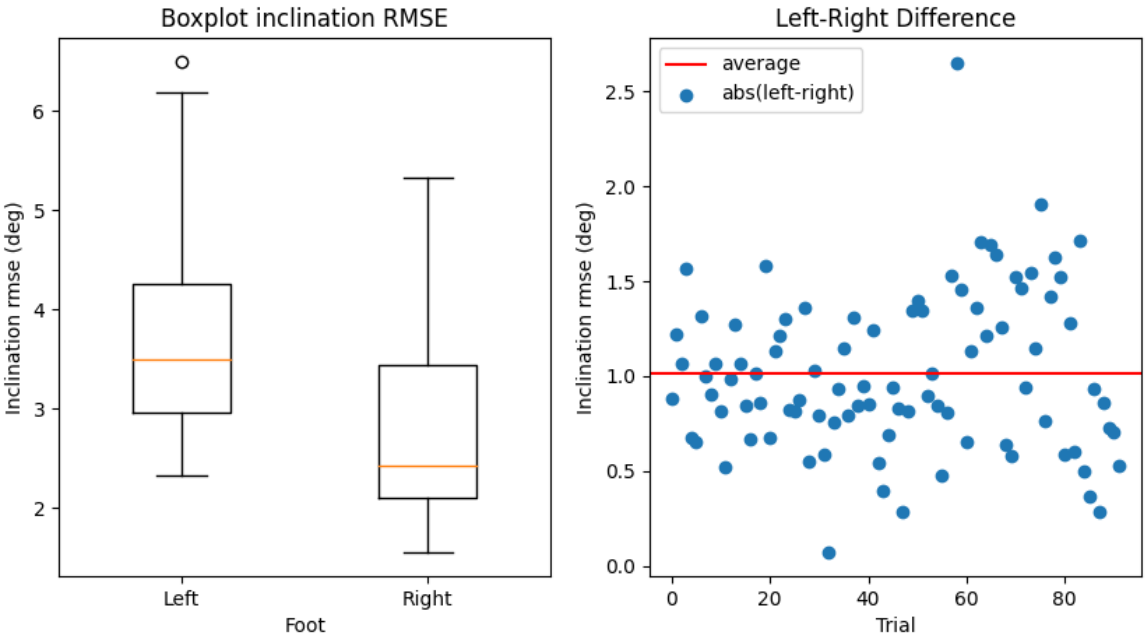

FIGURE 5.12: Box-plot of inclination RMSE separated by foot and scatter plot of difference in inclination RMSE between feet, with the average marked in red.

with the average value of this difference as a red horizontal line.

To discover the root of this issue, the distances between the optical markers were plotted, as well as the angle between the vectors used to compute the quaternion that represents the orientation of the optical markers. Close-ups of the results of a single, representative trial are shown in Figure 5.13.

In light of these results (that will be discussed more thoroughly next on Section 5.4.1), and to improve the results, the position of the optical markers was filtered. Two kinds of filters were applied: a third order low pass filter forwards and backwards, varying the cutoff frequency; and a moving median filter with different window sizes. For each them, the top performers - in terms of correcting the inclination RMSE disparity between both feet - were the low pass one with a cutoff frequency of 6Hz and the moving median with a window size of 5 samples. A detail of the filtered and unfiltered position of the y-coordinate of the front marker of the left foot from a representative trial can be seen in Figure 5.14.

The results are shown in Figures 5.15 and 5.16. The former shows a box-plot that displays the inclination RMSE separated by foot for the results with the two chosen filters as well as unfiltered data, whereas the latter presents also in a box-plot form the average results for each foot as well as the absolute difference in inclination RMSE between left and right foot.

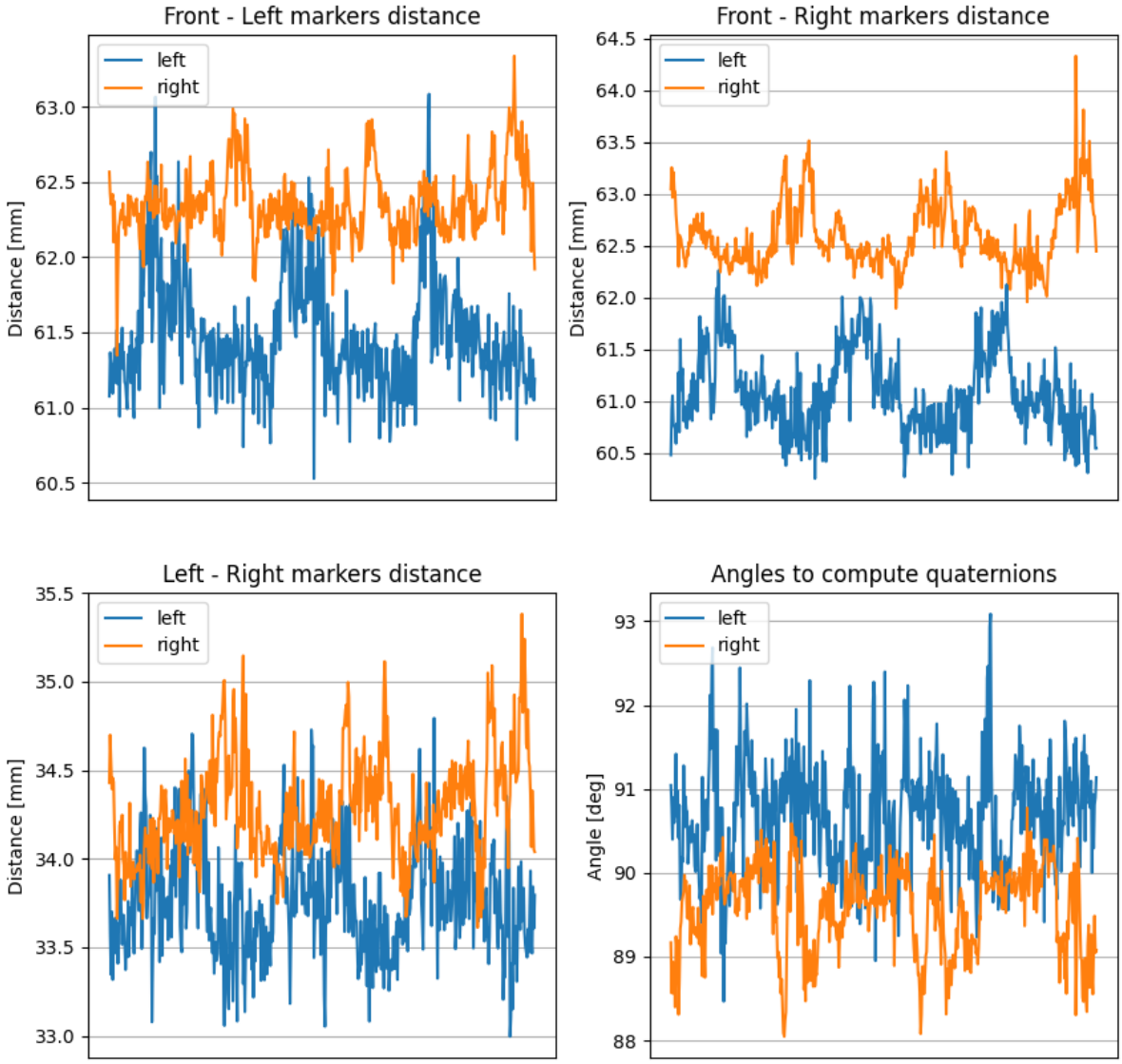

FIGURE 5.13: Distances between optical markers by foot, and angle between vectors used to compute the quaternion describing the optical system frame.

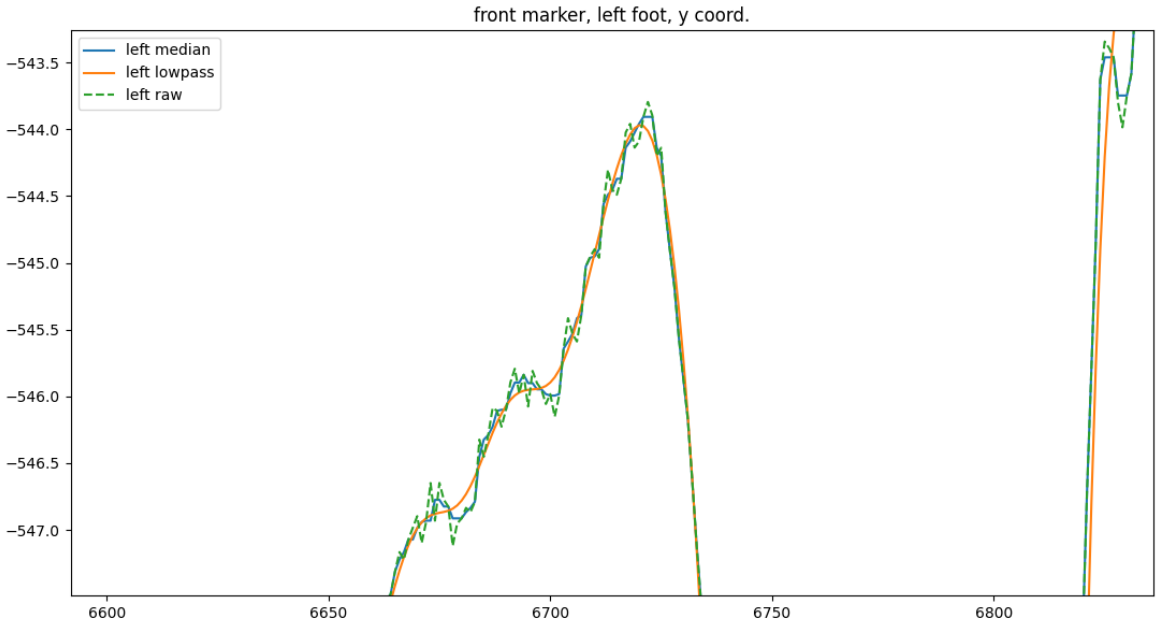

FIGURE 5.14: Detail of the y-position of the front optical marker on the left foot, with 2 filtered used and raw data.

5.4.1 Discussion

First and foremost, it is observed in Figure 5.12 that there is indeed a major difference between both feet. In this case, the left foot presents higher (and, thus, worse) inclination RMSE values consistently. On average, the results for the left foot are over 1º higher that those of the right one.

Figure 5.13 can give some insight into why is this happening. As it can be seen, the distances between optical markers of the left foot (in blue) tend to be noisier than those of the the right one. This is specially visible on the Front-Left distance (top-left subfigure). However, it should be noted that those differences are between 1mm and 2mm, which are rather low. The same kind of increased noise can be appreciated on the angle between vectors used to compute the quaternion than represents the optical markers' frame. In this case, it is more preoccupying, since a noisier signal implies a more unstable frame as well. All in all, it seems that either the optical markers where not as accurately fixed on the left foot as in the right one; or that the left foot IMU strap is not as well positioned as the right one. These may cause slight, but unwanted variations on the positions of the optical markers that must be solved, as it is used as a reference system.

To try to avoid this noise, a proposed solution was filtering the positions of the optical markers. This allows smoothing them and avoiding unwanted trembles, and at the same time can achieve more similar results between both feet. As seen in Figures 5.15 and 5.16, the low pass filter with cutoff frequency of 6Hz is the best option. It improves considerably the left foot's results, reducing the average gap between left and right inclination RMSE to below 0.5º. Moreover, it also improves right foot's results, albeit in a lesser manner. This shows that filtering is actually a good decision

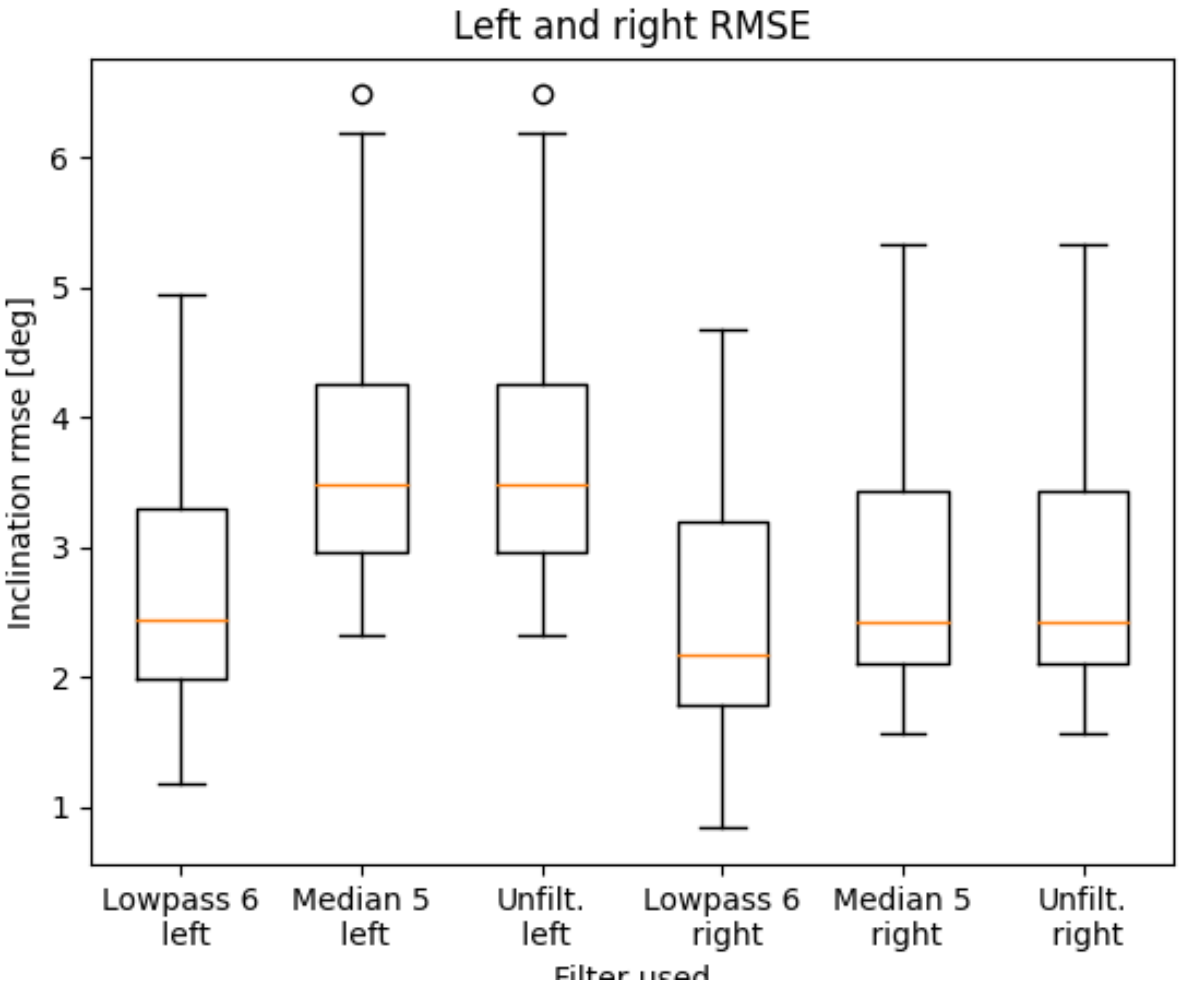

FIGURE 5.15: Inclination RMSE by foot for 2 filtered and unfiltered optical markers positions.

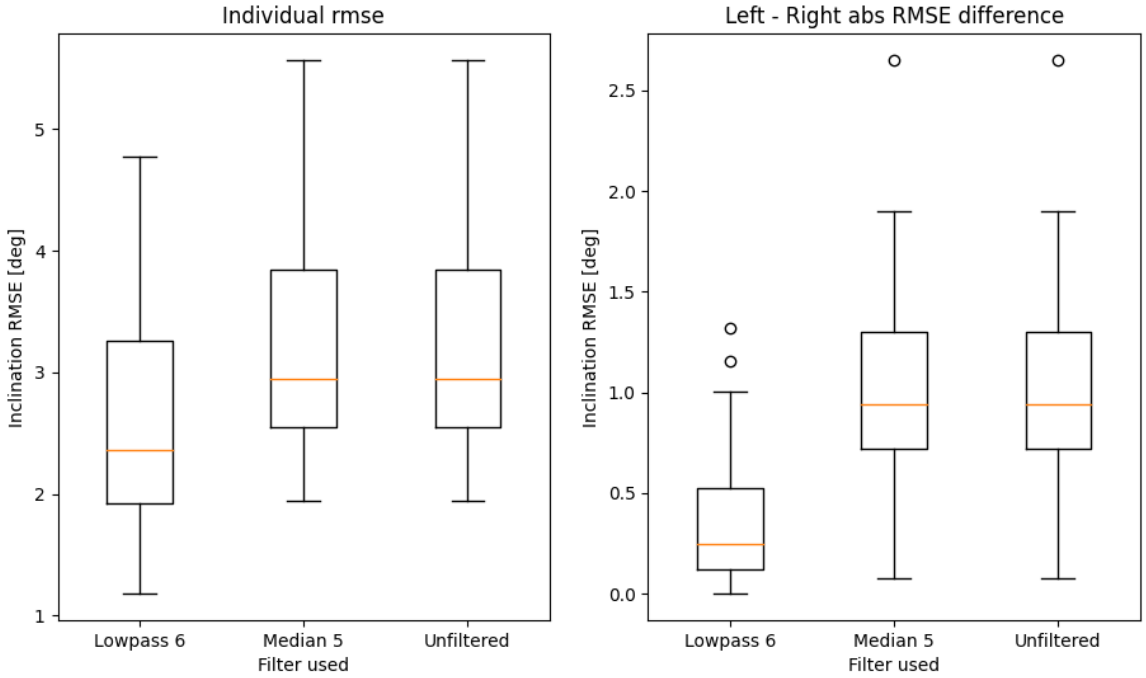

FIGURE 5.16: Overall inclination RMSE and absolute difference between left and right foot for 2 filtered and unfiltered optical markers positions.

even when no such discrepancy between left and right foot is observed, as it can correct outliers and even out the inherent noise.

Chapter 6

Conclusions

6.1 Overview

This thesis presents itself as an entry point for newcomers to the *gaitt* project. It describes the general gait terms, gait analysis methods, and provides an overview to its actual implementation. Also, it tests the implementation of a novel motion detection algorithm and tunes one of the parameters of an already existing orientation estimation method. Both cases were evaluated using a data set composed of 92 trials with different walking patterns and evaluated against a ground truth reference.

Chapter 1 serves as an introduction to the thesis: its motivation and the goals are presented. This defines and articulates this thesis' content.

Chapter 2 explains the core gait terms, some the gait phase detection algorithm and the gait analysis methods. Its aim is to introduce crucial points to those unfamiliar with gait analysis and its methods; explain current algorithms, as well as become a sort of documentation for the code that performs this analysis.

Chapter 3 introduces the gait visualization tool, also part of the *gaitt* project, and presents its most important algorithms and its capabilities. Despite its little connection to other sections, it is still a relevant part of the *gaitt* project and is worthy of an explanation. It is the ultimate gait analysis tool as it provides accurate visual representation of pre-recorded gait data.

Chapter 4 presents the main contributions of this thesis to the *gaitt* project. It is the attainment of its goals; where the extensions and improvements are explained, as well as the methods and reasoning behind them. Here is an overview of the organization of the project's code, the implementation of the new motion detection method, and the tuning of the τ_{acc} parameter from one of the orientation estimation algorithms.

Chapter 5 provides the results from the experimental set up to validate the new methods. It summarizes the experimental framework, provides insight into its set up and presents the actual results from the validation of each practical extension. The new rest detection method provides very similar results to those of GPD-One; whereas the implementation tuning of the orientation estimation, despite promising first results, proved to be diminishing. Finally, a disparity between left and right foot results is studied and corrected using a low-pass filter.

6.2 Future work

The results from the improvements to the *gaitt* project this thesis has proposed have been underwhelming and far from expected. Despite introducing a new rest detection method in Section 4.2, an improvement over the existing one has yet to be seen. There can be several reasons behind this issue: the current method is a better option, some of the parameters of the new method need more tuning, or better results cannot be achieved when using inertial measurement units. This should be addressed, because it conditions the future lines of work regarding gait analysis.

The other method tested, presented in Section 4.3, yielded even worse results. Despite seemingly improvements on the orientation estimation of the sensor, the actual results derived from its implementation have been worse that the existing alternatives. The reasoning behind this is not clear and should be investigated further; otherwise, the parameters used on this algorithm become useless.

As a final point, in Section 3.1.4, it is mentioned the creation of two linear spaces as a part of the procedure. Each of these linear spaces is formed by 100.000 samples. At a sampling rate of 110Hz, this can can cover up to 15 minutes of data recordings without losing samples, whereas if the frequency doubles to 220Hz, the time is reduced to 8,33 min. Even though trials are not that long yet, it may become an issue in the future. This can be solved by using a number of samples which depends on the trial duration and sampling rate (which are both known values), instead of a fixed value; thus being more flexible.

Appendix A

Detailed code explanations

A.1 Python code

main demo.py		
Inputs	**Outputs**	**Algorithm**
- IMU data - .txt / .mat file - .gaitt.mat DataFile - .json file	- .txt file - .gaitt.mat DataFile - .json file - Webapp visualization - PDF plots (optional)	Saves info into .txt file (recordImuData) Processes data: Performs GPD, gait analysis and computes several gait parameters. Opens Matlab, performs orientation, position and direction correction and saves into a .json Open visualizer and shows gait. Creates several plots, saves them on PDF and displays them on Okular

processData (demo.py)		
Inputs	**Outputs**	**Algorithm**
- filename: DataFile - matlab: Matlab path - (Opt.) noLatdevPlots: Lateral deviation plots (True/-False, save on PDF)	- .gaitt.mat DataFile with all info - Creates .sync.png plot, not returned - Creates .json, not returned - (optional) Creates PDF with several plots and saves it, not returned	Reads .txt, formats information appropriately (can synchronize it too) and saves it into a DataFile, plots gait and saves it as image. Runs GPD, gait analysis and computes gait parameters, and saves it into a .gaitt.mat. Opens Matlab, performs orientation, position and direction correction and saves into a .json

loadSsnImuData (io.py)		
Inputs	**Outputs**	**Algorithm**
- data: file to read (.txt or similar, or .gz) - (Opt.) rate: IMU rate - (Opt.) sync: synchronize data (T/F) - (Opt.) syncPlotAx: plt.plot, where to plot if synced. - (Opt.) mapping: dict, IMUs mapping (left/right)	- Formatted and filled DataFile - (Opt.) updated synced plt.plot inputted.	Reads input file and writes all its information into a DataFile in standard format. If sync, data from both IMUs is synced. If syncPlotAx is passed, updates it with synced plot

syncSsnImuData (io.py)		
Inputs	**Outputs**	**Algorithm**
- data: DataFile - (Opt.) syncPlotAx: plt.plot, where to plot - (Opt.) imus: tuple, IMUs' order.	- Modified DataFile, not returned	Synchronizes two data streams within a single DataFile.

runGaitt (utils.py)		
Inputs	**Outputs**	**Algorithm**
- data: DataFile with gait information - (Opt.) perry: T/F, perform perry's gpd or unilateral - (Opt.) params: dict, several parameters (such as thresholds) used in GPD, gait analysis and gait parameter's computation. Any number of them can be specified.	- Modified DataFile, not returned	Modifies DataFile and completes it with GPD, gait analysis and several gait parameters. Includes default values for input parameters not specified. If perry is selected, computes and stores contralateral foot's GPD, analysis and parameters.

_runGaittSingleFoot (utils.py)		
Inputs	**Outputs**	**Algorithm**
- inputs: user-defined parameters (such as thresholds) used in GPD, gait analysis and gait's parameters computation - Data: DataFile with gait information	- New DataFile (returned) with all the results	Performs, sequentially: - GPD (perry's GPD too if data available) - Gait Analysis - Other values: - Step Indices (if perry too) - Step Times (if perry too) - Global Frame Kinematics - Step Durations (perry too) - Foot angles (pitch & roll) - Stride length - Z position - Lateral deviation - Gait velocity

openVisualization (demo.py)		
Inputs	**Outputs**	**Algorithm**
- jsonfilename: name .json file containing all the necessary data for the visualization.	- None	Copies the file into another directory and opens webapp visualization.

createPlots (demo.py)		
Inputs	**Outputs**	**Algorithm**
- filename: Existing DataFile name - data: .gaitt.mat DataFile containing all the gait info to plot	- .pdf File containing all the plots	Reads the data and creates several plots from it, saves them into a .pdf file with the same name and title as the filename passed and saves it in the same directory as filename

createSignalPlots (*signal_plots.py*)		
Inputs	**Outputs**	**Algorithm**
- data: .gaitt.mat DataFile containing the gait data to be plotted - title: title for gait evaluation plots - plotFilename: .pdf filename - (Opt.) perry: Plot perry phases on gait evaluation (T/F/both) - width: width of figures - (Opt.) aggregatedDetailsWriter: adds title and plot name	- None	Creates several plots (each one is a function call) from the data and saves them into a .pdf file

plot- (*signal_plots.py*)		
Inputs	**Outputs**	**Algorithm**
- ax: Matplotlib figure - data: gaitt.mat DataFile containing all gait data - time: sample times - rate	- None. Plots on the inputted figure	Creates the corresponding plot on the input figure. All but plotCombinedGaitphase use time.

getCutRawData (*signal_plots.py*)		
Inputs	**Outputs**	**Algorithm**
- data: gaitt.mat DataFile containing all gait data - identifier: signal identifier in DataFile format	- Cut signal	Reads the DataFile passed, picks the signal specified and cuts it according to the start and end indices specified within the DataFile.

createFig (utils.py)		
Inputs	**Outputs**	**Algorithm**
- (Opt.) width: % of width of the page - (Opt.) landscape: create the figure in landscape or vertical page width	- Matplotlib figure	Creates a matplotlib figure of width % relative to page width, either vertical or landscape

gaitEvalationPlot (*gait_evaluation.py*)		
Inputs	**Outputs**	**Algorithm**
- data: gaitt.mat DataFile containing the gait data to be plotted - title: title: title for gait evaluation plots - (Opt.) config: C++ file with plot configurations - (Opt.) perry: Plot perry phases on gait evaluation (T/F)	- Matplotlib figure	Creates and returns a single Matplotlib figure with several plots on it (velocity x stride length, percentage duration of gait phases, pitch and swing roll, and maximal lateral deviation and z position).

velLengthPlot (*gait_evaluation.py*)		
Inputs	**Outputs**	**Algorithm**
- res : gait evaluation data from one foot - side: foot to plot - config: C++ function containing gait evaluation plots' configuration	- Plots on the last Matplotlib figure created	Plots a boxplot of stride length against gait velocity.

stepTimePlot (*gait_evaluation.py*)		
Inputs	**Outputs**	**Algorithm**
- resL & resR: gait evaluation data from left & right foot - phase: phase to plot - config: C++ function containing gait evaluation plots' configuration - ticks: (T/F) plot default ticks on y-axis or differentiate between left and right	- Plots on the last Matplotlib figure created	Plots a boxplot of selected phase's duration of one foot against the other

anglePlot (*gait_evaluation.py*)		
Inputs	**Outputs**	**Algorithm**
- resL & resR: gait evaluation data from left & right foot - angle: angle (pitch or swing roll) to plot - config: C++ function containing gait evaluation plots' configuration	- Plots on the last Matplotlib figure created	Plots a boxplot of selected angle's min and max (one boxplot each) values of one foot against the other

posPlot (*gait_evaluation.py*)		
Inputs	**Outputs**	**Algorithm**
- resL & resR: gait evaluation data from left & right foot - name: plot will be maximal lateral deviation or maximal z-position - config: C++ function containing gait evaluation plots' configuration	- Plots on the last Matplotlib figure created	Plots a boxplot of selected name's max values of one foot against the other

boxplot2D (*gait_evaluation.py*)		
Inputs	**Outputs**	**Algorithm**
- xData: Values on the x-axis - yData: Values on the y-axis - col: colours to use - label	- Plots on the last Matplotlib figure created	Plots a boxplot of the data. Needs access to *gait_evaluation.cpp* to retrieve information about the boxplot's configuration.

A.2 C/C++ code

gpdFull (*gpd_offline.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *acc_sens* - *gyr_sens* - N - Rate - maxFind - jerkWindow - jerkThreshold - *w_acc* - *w_gyr* - *hist_acc_p* - *hist_gyr_p* - *th_rest_min* - *th_high_min*	- *gyr_rest* - *acc_rest* - *gpd_rest* - *gpd_two* - gpd - *gyr_nobias* - *gyrbiastest_rest* - *gyrnorm_sens* - *accnorm_sens* - tiltrate - jerkNorm - *th_accrest_used* - *th_gyrrest_used*	Performs whole GPD of 1 foot from raw IMU data. If Autotune is selected, it self-computes some of the necessary thresholds and hysteresis values needed.

removeGyrBias (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gyr - rate - gyrnorm - *min_rest_duration*	- *gyr_nobias* - *gyrbiastest_rest*	Estimates and removes the gyro's readings bias using rest phases In between rest phases, linearly interpolates said bias.

vecNorm(*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- vec - N	- ret	Returns the Euclidean norm of a Nx3 vector

movingAverageFiltFilt (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- signal - N - rate - t	- signal	Performs a zero-phase moving average calculation twice, once forward and once backwards, using movingAvg.

movingAvg (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- input - N - filterLen - reversedOutput	- output	Moving average calculator. Can return the output reversed if requested.

gpdOne (*gpd_offline.cpp*)		
Inputs	**Outputs**	**Algorithm**
- arr - N - mode - rate - th - h - *th_rest_min_miliseconds* - *th_high_min_miliseconds*	- ret	First phase of the GPD. Returns an array that distinguishes between movement (1) and rest phases (0). To do so, performs acausal (with hysteresis) thresholding of both acc and gyro signals (separately). Afterwards, removes short rest phases and motion phases, in that order.

Intersection (*gpd_offline.cpp*)		
Inputs	**Outputs**	**Algorithm**
- arr1 - arr2 - N - rate - *th_rest_min_miliseconds* - *th_high_min_miliseconds* - *th_combined_rest_min_p* - *th_combined_rest_high_p*	- ret	Returns the intersection of two binary signals. Concretely, it is used to return the intersection between the results of the GPD-One for the acc and the gyro, being the values of the returned array 1 only if both are, and 0 otherwise. Also, as before, removes short phases within the returned array.

Intersection (*gpd_offline.cpp*)		
Inputs	**Outputs**	**Algorithm**
- arr1 - arr2 - N - rate - *th_rest_min_miliseconds* - *th_high_min_miliseconds* - *th_combined_rest_min_p* - *th_combined_rest_high_p*	- ret	Returns the intersection of two binary signals. Concretely, it is used to return the intersection between the results of the GPD-One for the acc and the gyro, being the values of the returned array 1 only if both are, and 0 otherwise. Also, as before, removes short phases within the returned array.

tiltRate (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gyrsens - start - N	- tiltrate	Computes the tiltrate (integral times cross product) of a given gyro signal

findMax (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- array - N	- ret	Returns the index of the max element of the array.

findZero (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gyrsens - start - N	- tiltrate	Computes the tilt rate (integral times cross product) of a given gyro signal

findMax (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- array - N - th	- ret	Returns the index of zero-crossing element of the array, starting from those elements superior to a given threshold.

gpdThree (*gpd_offline.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *gpd_two* - *acc_sens* - N - Rate - percentageWindow - percentageThreshold	- *gpd_three* - jerknorm	Third phase of the GPD. Computes the jerk (finite difference derivative of the acceleration). Within a window defined between the toe-off and the beginning of the next foot-flat, looks for the first time that the norm of the jerk rises above a threshold of its maximum, and marks it as the start of the loading response (3)

finDiff (*gpd_offline_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- vec - N - rate	- ret	Returns the forward-finite difference derivative of a 3D array

perryGaitPhase (*gpd_offline.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gp - *gp_contra* - N	- *gp_perry*	Returns the Perry gait phases from the gpd of a foot and it's contralateral one.

combinedGPD (*gpd_offline.cpp*)		
Inputs	**Outputs**	**Algorithm**
- arr1 - arr2 - N	- ret	Combines two gpd signals into one, returned in ret, so the resulting phases are: 1. double full-contact 2. double support 3. single support 4. zero support 5. otherwise

gaitAnalysis (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- foot - gpd - *acc_sens* - *gyr_sens* - N - rate - fwdAxis - startP - stopP - tauAcc - zeta - timeout	- *gyr_footframe* - *vel_footframe* - xprime - x - y - z - *quat_sensor2eart* - *quat_sensor2foot* - *quat_sensor_pff2earth* - *quat_sensor2sensor_pff* - *eulerAngles_sensor2sensor_pff* - *quat_foot2earth*, - *eulerAngles_foot2earth* - *pos_footframe_pff* - *posnorm_footframe_pff* - *pos_footframe_fff* - *posnorm_footframe_fff* - *quat_foot2foot_pff* - *eulerAngles_foot2foot_pff*	Computes gyr and velocity in foot frame, the foot axes at foot-flats, several different orientations, Euler angles and positions. There are also more values computed not returned here.

stepCounter (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gpd - N	- ret	Returns the number of steps in a GPD array. To do so, it counts as a step a full 0-1-2-3 sequence of less than 5 seconds.

stepIndices (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gpd - rate - N - timeout - numSteps	- *step_indices*	Returns an array of 6 elements with the indices of the start of the step, each one of the 4 phases and the end of the step (in that order).

findStep (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gaitphase - N - startInd - timeoutSamples	- ret (bool) - stepStart - start1 - start2 - start3 - start0 - stepStop	Returns true if the indices have been found for the next step. If so, it returns also the indices of the start of the step, each one of the 4 phases and the end of the step.

globalQuaternion (*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- acc - gyr - tauAcc - zeta - N - rate	- outputQuaternion	Computes quaternion to translate sensor to global frame, using orientation estimation algorithm (either globalQuaternionFiltFilt for negative tauAcc values or CupdateOffline from CCsgOriEstIMU for the rest)

globalQuaternionFiltFilt (*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- acc - gyr - N - T - rate	- outputQuaternion	Computes quaternion to translate sensor to global frame, using orientation estimation algorithm (non-causal filtering of accelerations)

QuaternionFrom_pff (*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- oriestQuat - *step_indices* - numSteps	- *quats_pff*	Computes quaternion from the sensor at the previous foot flat to earth frame, using step indices and the quaternion from sensor to earth computed before.

accelerationToGlobal (*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *acc_sens* - *quat_sensor*2*earth* - N	- *acc_earth* - *accnorm_earth*	Transforms the acceleration to earth frame and removes the effect of gravity. Computes also the norm of said acceleration.

integrateAcceleration (*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *acc_earth* - *step_indices* - N - numSteps - rate	- *vel_earth* - *velnorm_earth*	Computes velocity in earth frame and its norm by integrating the acceleration. It also performs a linear drift correction by imposing that the velocity is 0 in the middle of the foot flat.

velocityTo_sensorframe (*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *vel_earth* - *quat_sensor2earth* - N	- *vel_sens*	Translates velocity back to sensor frame from the previously obtained velocity in earth frame

integrateToPosition(*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- vel - N - rate	- pos - posnorm	Integrates the velocity to get the position and its norm

defCoordXprime_footframe_from_pff (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *vel_earth* - *quat_sensor2earth_pff* - *step_indices* - N - numSteps - startP - stopP	- xprime	X' coordinate, found by calculating the normalized average sensor speed in earth frame during the middle 50% window of the swing phase.

defCoordZ_footframe (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *acc_sens* - N - *step_indices* - numSteps	- coordZ	Z coordinate, found by computing the average of the acceleration measurements during the whole duration of the rest phase.

defCoordXandY_footframe (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- xprime - z	- y - x	Computes the remaining Y and X coordinates via vector multiplication. Y is orthogonal to Z and X' , and X is orthogonal to Y and Z

defQuat_sensor2foot (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- x - y - z	- *quat_sensor2foot*	Computes quaternion that transforms from sensor to foot frame

footGyr_footframe (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *gyr_sens* - *quat_sensor2foot* - N	- *gyr_footframe*	Transforms gyro readings (angular rate) from sensor to foot frame

footVelocity_footframe (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *vel_sens* - *quat_sensor2foot* - N	- *vel_foot_footframe*	Transforms the velocity of the foot from sensor frame to foot frame

sensorRelativeTo_pff (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *quat_sensor2earth* - *quat_sensor_pff2earth* - *step_indices* - numSteps	- *quat_sensor2sensor_pff* - *eulerAngles_sensor2sensor_pff*	Computes the orientation quaternion and Euler Angles from any given sensor position to the sensor position at the previous foot flat.

footRelativeTo_pff (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- foot - *quat_sensor2sensor_pff* - *quat_sensor2foot* - N	- *quat_foot2foot_pff* - *eulerAngles_foot2foot_pff*	Computes the orientation quaternion and Euler Angles from any given foot position to the foot position at the previous foot flat.

footRelativeTo_earthframe (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- foot - *quat_sensor2earth* - N - *quat_sensor2foot*	- *quat_foot2earth* - *eulerAngles_foot2earth*	Computes the orientation quaternion and Euler Angles from foot frame to earth frame

footPosition_footframe_pff (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *vel_earth* - *step_indices* - *quat_sensor_pff2earth* - *quat_sensor2foot* - N - numSteps - rate	- *pos_footframe_pff* - *posnorm_footframe_pff*	Computes the position of the foot a the middle of the previous foot flat in foot frame.

footPosition_footframe_fff (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *vel_earth* - *quat_sensor2earth_fff* - *quat_sensor2foot* - N - numSteps - rate	- *pos_footframe_fff* - *posnorm_footframe_fff*	Computes the position of the foot a the middle of the first foot flat in foot frame

stepIndicesPerry (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- gpPerry - N	- *step_indices_perry*	Returns an array of 6 elements with the indices of the start of the step, each one of the 4 Perry phases and the end of the step (in that order).

stepTimesPerry (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *step_indices_perry* - rate - numSteps	- *step_times_perry*	Returns an array of 6 elements with time of the first initial contact, the percentage duration of the 4 Perry phases and the time of the next initial contact (in that order).

globalFrameKinematics (*gait_analysis.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *acc_sens* - *quat_sensor2earth* - N - rate - numSteps - *step_indices*	- *vel_earth* - *velnorm_earth* - *vel_earth_nodriftCorr* - *velnorm_earth_nodriftCorr* - *pos_earth* - *posnorm_earth* - *pos_earth_nodriftCorr* - *posnorm_earth_nodriftCorr*	Computes velocity and position (3 components, each) in global coordinates, both with and without drift correction.

integrateAcceleration_nodrift (*gait_analysis_utils.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *acc_earth* - *step_indices* - N - numSteps - rate	- *vel_earth* - *velnorm_earth*	Integrates acceleration to obtain velocity in global frame, and applies drift correction to it.

stepDurations (*gait_evaluation.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *step_times* - numSteps	- *step_duration* - *percentage_lr* - *percentage_ff* - *percentage_ps* - *percentage_sw*	Computes the total step duration of each step and the relative duration of each of its phases.

footAngles (*gait_evaluation.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *eulerAngles_foot2foot_pff* - N - *step_indices* - numSteps - rate	- *min_pitch* - *max_pitch* - *min_swing_roll* - *max_swing_roll*	Computes the min/max pitch angles during the whole step and the min/max roll angles during the swing phase. All angles returned are filtered using a moving average filter

footTrajectory (*gait_evaluation.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *pos_earth* - N - *step_indices* - numSteps - (rate)	- *stride_length* - *max_lat_deviation* - *max_z_pos* - *lat_deviation* - *z_pos*	Computes the stride length of each step as well as the (max) lateral deviation from the centre line and the (max) vertical position achieved.

gaitVelocity (*gait_evaluation.cpp*)		
Inputs	**Outputs**	**Algorithm**
- *step_duration* - *stride_length* - numSteps	- *gait_velocity*	Computes the gait velocity for each step

A.3 Matlab code

create_json (*/3d/matlab_preprocessing/*)		
Inputs	**Outputs**	**Algorithm**
- filename: name of a DataFile containing gait information. - options: • Parameters • Flags for corrections	- *gait_velocity*	The filename must contain gait data. This program loads it, applies several position, direction and orientation correction to the gait in it, creates a foot mesh, positions the foot and toes and makes the appropriate change of coordinates to suit a babylonjs file. Saves everything into it so it can be later displayed.

stepwise_ori_delta (*/3d/matlab_preprocessing/*)		
Inputs	**Outputs**	**Algorithm**
- delta	LEFT AND RIGHT - *foot_middle_orientation_left* - *stepind_left*	Computes the delta angle (deviation from straight path) necessary for he stepwise orientation correction

end_correction (*/3d/matlab_preprocessing/*)		
Inputs	**Outputs**	**Algorithm**
LEFT OR RIGHT - *sensor_position_left* - *foot_middle_orientation_left* - *perfect_start_left* - *perfect_end_left* - *num_samples*	LEFT OR RIGHT - *sensor_position_left* - *foot_middle_orientation_left*	Computes the vectors that transform each sample of the path followed (assuming perfect start) into another path where with perfect start and end positions.

z2ground (/3d/matlab_preprocessing/)		
Inputs	**Outputs**	**Algorithm**
LEFT OR RIGHT - *sensor_position_left* - *gait_phase_left*	LEFT OR RIGHT - *cost_l* - *stairs_num_left* - *stair_height_left* - *gp_0_start_l* - *gp_0_end_l*	Corrects the vertical deviation drift (z-axis) and classifies steps via k-means clustering into 3 different heights: level ground, 1 stair height and 2 stair height.

stair_num_correction (/3d/matlab_preprocessing/)		
Inputs	**Outputs**	**Algorithm**
LEFT AND RIGHT - *sensor_position_left* - *stairs_num_left* - *cost_l* - *gp_0_left* - method	LEFT AND RIGHT - *stairs_num_left*	Corrects the number of steps computed with z2ground using 2 different methods, and choosing the best outcome

lowpass_heading_correction (/3d/matlab_preprocessing/)		
Inputs	**Outputs**	**Algorithm**
LEFT OR RIGHT - *sensor_position_left* - *foot_middle_orientation_left* - *stepind_left* - rate	LEFT OR RIGHT - *sensor_position_left* - *foot_middle_orientation_left*	Computes the delta angle (deviation from straight path) necessary for he stepwise orientation correction and applies a low-pass filter to it. Returns the corrected positions.

lowpass_k_heading_correction (/3d/matlab_preprocessing/)		
Inputs	**Outputs**	**Algorithm**
LEFT OR RIGHT - *sensor_position_left* - *foot_middle_orientation_left* - *stepind_left* - rate - *k_ratio*	LEFT OR RIGHT - *sensor_position_left* - *foot_middle_orientation_left*	Computes the delta angle (deviation from straight path) necessary for he stepwise orientation correction and corrects it using a fixed gain *k_ratio*. Returns the corrected positions.

foot_V2 (/3d/matlab_preprocessing/)		
Inputs	**Outputs**	**Algorithm**
LEFT OR RIGHT - *foot_middle_orientation_left* - *sensor_position_left* - *perfect_sensor_position_l_mesh* - *heel_pos_mesh* - *midfoot_pos_mesh* - *toe_pos_mesh*	LEFT OR RIGHT - *toe_orientation_left* - *foot_earth_vec_left* - *correction_vec_left* - *middle_orientation_left*	Creates the foot mesh and positions it accordingly (appropriate orientation too). Moves midfoot and heel above the ground and rotates toes so they lie flat on the floor.

Bibliography

[1] Eva Kastenbauer. *Data-based development and validation of inertial foot motion tracking and 3D visualisation for clinical gait analysis*. 2019.

[2] TUB Control Systems group. "Gait Assessment by Foot-Worn IMUs – Extensive Validation of a Calibration-free and Magnetometer-free Method". 2020.

[3] Myeounggon Lee et al. "Validity of shoe-type inertial measurement units for Parkinson's disease patients during treadmill walking". In: *Journal of neuroengineering and rehabilitation* 15.1 (2018), p. 38.

[4] Tao Liu, Yoshio Inoue, and Kyoko Shibata. "Development of a wearable sensor system for quantitative gait analysis". In: *Measurement* 42.7 (2009), pp. 978–988.

[5] Edward P Washabaugh et al. "Validity and repeatability of inertial measurement units for measuring gait parameters". In: *Gait & posture* 55 (2017), pp. 87–93.

[6] Benoit Mariani et al. "Heel and toe clearance estimation for gait analysis using wireless inertial sensors". In: *IEEE Transactions on Biomedical Engineering* 59.11 (2012), pp. 3162–3168.

[7] Tri Nhut Do and Young Soo Suh. "Gait analysis using floor markers and inertial sensors". In: *Sensors* 12.2 (2012), pp. 1594–1611.

[8] H Martin Schepers. "Ambulatory assessment of human body kinematics and kinetics". In: (2009).

[9] Thomas Seel, David Graurock, and Thomas Schauer. "Realtime assessment of foot orientation by accelerometers and gyroscopes". In: *Current Directions in Biomedical Engineering* 1.1 (2015), pp. 446–469.

[10] Stacy J Morris Bamberg et al. "Gait analysis using a shoe-integrated wireless sensor system". In: *IEEE transactions on information technology in biomedicine* 12.4 (2008), pp. 413–423.

[11] Benoit Mariani et al. "On-shoe wearable sensors for gait and turning assessment of patients with Parkinson's disease". In: *IEEE transactions on biomedical engineering* 60.1 (2012), pp. 155–158.

[12] Angelo M Sabatini et al. "Assessment of walking features from foot inertial sensing". In: *IEEE Transactions on biomedical engineering* 52.3 (2005), pp. 486–494.

[13] Jacquelin Perry. *Ganganalyse: Norm und Pathologie des Gehens*. Elsevier, Urban&FischerVerlag, 2003.

[14] TW Ridler, S Calvard, et al. "Picture thresholding using an iterative selection method". In: *IEEE trans syst Man Cybern* 8.8 (1978), pp. 630–632.

[15] Thomas Seel and Stefan Ruppin. "Eliminating the effect of magnetic disturbances on the inclination estimates of inertial sensors". In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 8798–8803. ISSN: 2405-8963. URL: http://www.sciencedirect.com/science/article/pii/S2405896317321201.

[16] Patrick D. Wrede. *Ruhephasenerkennung bei "Inertial Sensor-Based Gait Analysis"*. 2019.