Use cases of inertial measurement units: Pedestrian dead reckoning navigation

Thomas Blommaert

Supervisors: Dr. ir. Frederick Bossuyt, Dr. ir. Bert Vankeirsbilck
Counsellors: Dr. ir. Ann Monté, Prof. dr. ir. Jan Vanfleteren

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Electronics and Information Systems
Chair: Prof. dr. ir. Koen De Bosschere

Department of Information Technology
Chair: Prof. dr. ir. Bart Dhoedt

Faculty of Engineering and Architecture
Academic year 2017-2018

GHENT
UNIVERSITY

Preface

I would like to thank our supervisors Frederick Bossuyt and Bert Vankeirsbilck for their support, help and feedback during the past year. I would also like to thank their colleagues Jacob Staley and Harald de Bondt who were present at some of our meetings and provided much useful feedback as well and our counsellors Ann Monté and Jan Vanfleteren to provide the opportunity to work on this thesis.

A special thanks also goes to Dries and Mats, with whom I worked together for the first part of this thesis and who made this past year very enjoyable.

Many thanks to Martijn Courteaux for his technical support and feedback during the past busy nights.

Thanks to my parents for supporting me throughout my whole education.

Last but not least, thanks to my girlfriend Feun Singin for her continued support and appreciation.

Thomas Blommaert, June 2018

Permission of use

"The author(s) gives (give) permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In the case of any other use, the copyright terms have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation."

Thomas Blommaert, June 2018

Use cases of intertial measurement units: Pedestrian dead reckoning navigation

by Thomas Blommaert

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering
Academic year 2017–2018

Supervisors: Dr. ir. Frederick Bossuyt, Dr. ir. Bert Vankeirsbilck
Counsellors: Dr. ir. Ann Monté, Prof. dr. ir. JanVanfleteren

Faculty of Engineering and Architecture
Ghent University

Department of Electronics and Information Systems
Chair: Prof. dr. ir. Koen De Bosschere

Department of Information Technology
Chair: Prof. dr. ir. Bart Dhoedt

Summary

In this work a low-cost inertial motion data capturing system is developed. Multiple inertial measurement units can be connected to this system and data can be transferred to an external device for further processing. The designed system is then used in a practical use case: pedestrian dead reckoning. Data captured with the inertial measurement units is used to estimate the position of a walking human without the use of an absolute measurement tool such as GPS.

Keywords

inertial measurement unit, pedestrian dead reckoning navigation

Use cases of inertial measurement units: Pedestrian dead reckoning navigation

Thomas Blommaert

Supervisor(s): Dr. ir. Frederick Bossuyt, Dr. ir. Bert Vankeirsbilck,
Dr. ir. Ann Monté, Prof df. ir. Jan Vanfleteren

Abstract— **Inertial motion data capturing is often performed with very accurate and expensive motion capture suits. These are typically used in small niche markets e.g. movie productions. However, low-cost solutions that can be used in a wider range of applications are desirable. A system capturing motion of low-cost inertial measurement units (IMUs) is developed and used in a specific use case: pedestrian dead reckoning navigation. This is a technique to estimate the position of a moving person by capturing data such as acceleration. The current position is calculated based on the previous position and the relative movement provided by the data of the IMUs. This will be done by using both classic mathematical transformations and more modern machine learning techniques.**

Keywords— **inertial measurement unit, pedestrian dead reckoning navigation, Lasso**

I. Introduction

THE principal theme of dead reckoning navigation is using relative data such as acceleration rather than using an absolute measurement system like GPS in order to determine positions. The goal is to find out how precise a relative estimation can be in different scenarios.

In order to collect data from which relative movements can be extracted, a low-cost prototype of a suit able to capture inertial motion data needs to be developed, which is covered in the first section of this abstract. The second section covers the process of going from raw inertial motion data to a prediction of the walked path. Finally, the last section summarises the results that were achieved.

II. Development of an inertial motion data capturing suit

The development of an inertial motion data capturing suit requires an interplay of both hardware and software components. The hardware component consists of all elements necessary to capture inertial motion data and transfer it to an external device. The software component is responsible for coordinating the communication between the different hardware elements and transform the raw data to a representation that can be used for further processing. In the remainder of this section, both of these components are described.

A. Hardware

To build a hardware system capable of recording inertial motion data, two components are essential: IMUs to capture inertial data and a central module to coordinate the communication between all the components. An IMU module is a printed circuit board (PCB) containing an inertial measurement unit along with all necessary components to allow data to be read from the module by a central component. The component of the central module is a microcontroller which has two important functions. On the one hand, it needs to coordinate the communication with the IMU modules and decide what data should be read from which module at each point in time. On the other hand, it needs to transfer this data to an external source for further processing. These components are then linked to each other through wires and attached to long sleeve pants or shirts to create a prototype motion data capture suit. The prototype we designed with 3 IMU modules attached to one pant sleeve is depicted in figure 1.

Fig. 1. Prototype IMU suit. The 3 circles indicate the positions of the IMU modules.

B. Software

The purpose of the software framework is to allow communication between the different hardware components. It can be divided into two major parts: the first one running on the microcontroller, the second one running on the device receiving the data.

The framework running on the microcontroller coordinates communication with the different IMU modules and is designed with flexibility in mind. It is able to automatically detect how many IMU modules are attached at startup, so modules can be

easily added or removed. Moreover, the kind of data that should be read from each sensor (e.g. acceleration or angular velocity data) can be configured as well. Finally, the external source to which the data should be written can be specified. For efficiency reasons, data is serialised before being sent to the external source.

The other major part of the framework runs on an external device i.e. a laptop or computer. Its function is to receive data produced by the central module, deserialise it, and transform it to a format that can be used for further processing.

III. Pedestrian Dead Reckoning Navigation

A. Experimental setup

In order to (1) measure the accuracy of the upcoming results, (2) use supervised machine learning techniques and (3) combine results with absolute positioning into a hybrid system, ground truth data is necessary. The goal is to collect this data together with the data from the IMUs and align the datasets based on time.

In this use case, a GPS module is used as ground truth since it provides absolute positioning which is not prone to increasing errors due to drift. The prediction using the IMU data could then either be combined with the absolute GPS positioning in order to augment the result or replace the GPS module in the case of walking indoor where ofter no GPS signal can be established.

B. Orientation estimation

IMUs typically provide absolute orientations via on-board computations. However, it is needed to evaluate the precision of such derived data, since for pedestrian navigation the error margin is very low.

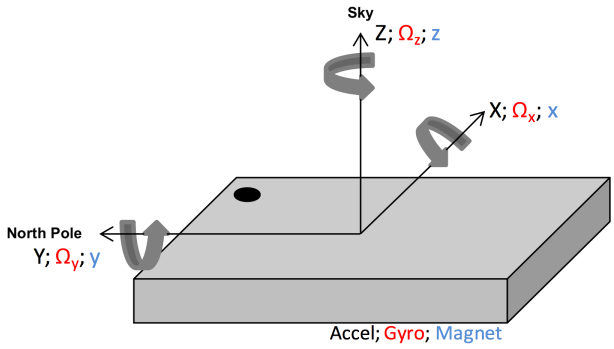


Fig. 2. The default coordinate system of the IMU.

Figure 2 shows the default coordinate system of the IMU. Regarding absolute orientations, the reference orientation appears to be the one where the Y-axis aligns with the direction of the North Pole and the Z-axis aligns with the inverse of gravity. It is now assumed the Z-axis will be the axis pointing forward when attaching the IMU onto the front of the body, e.g. when putting the back of the IMU on the leg. The orientation describing the walking direction can then be calculated as followed:

$$A = Q * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

with Q being the matrix representation of the quaternion. The projection A' in the horizontal plane (the ground) can then be determined by setting the Z-component to zero.

$$A' = \begin{bmatrix} A_x \\ A_y \\ 0 \end{bmatrix} \tag{2}$$

Finally, the angle θ can be calculated by inverse trigonometry and taking into account a result of 0 must align with the Y-axis (North Pole).

$$\theta = arctan2(A'_x, A'_y) \tag{3}$$

Here, $arctan2(y, x)$ is a variation on the more familiar $arctan(y/x)$ without a singularity at $x = 0$.

C. Displacement estimation

The walking speed of humans can vary in two possible ways. People can increase their speed by (1) taking larger steps in about the same time span or (2) taking about the same steps in a smaller time span. The real answer will be a combination of both which in fact is related to the definition of velocity: displacement (larger steps) divided by time (smaller time span).

Three features are investigated which are potentially correlated to the walking speed: acceleration, angular velocity and step duration using a step detection algorithm.

Thereafter, a learning model is selected. Based on simplicity and sparsity, Lasso is chosen as the learning model. Lasso is a machine learning method built on regularised linear regression. Its main difference with linear regression is that estimates sparse coefficients. It thereby tries to find a solution using as little as possible features which still result a high accuracy. It thus effectively reduces the number of variables upon which the given solution depends. This is done by minimising the loss function

$$L = \frac{1}{2n}\|y - w_0 - Xw\|_2^2 + \alpha\|w\|_1 \tag{4}$$

where n equals the amount of samples, X is the matrix with shape (n, p) consisting of n rows of samples having p features and α is a constant determining the penalty for the magnitude of the weights. This penalty $\alpha\|w\|_1$ is what differentiates Lasso from basic linear regression. Increasing α will decrease the magnitude of the weights, possibly resulting in a sparse solution with a few significant weights while the others approach zero.

IV. Results

The orientation and displacement estimation techniques are combined in order to predict the coordinates for the walked path. Figure 3 shows the predicted path for a walk in the Tech Lane Ghent Science Park, Ghent.

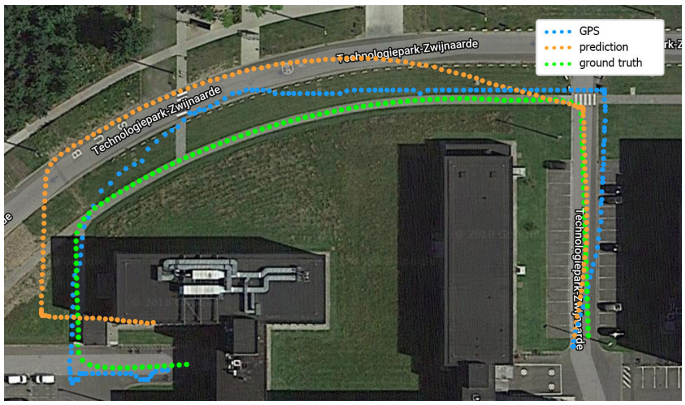


Fig. 3. Coordinates of the predicted path.

Apart from the top right region, the predicted path clearly resembles the turns and speed of the ground truth. The error in orientation at the top right is due to the IMU actually thinking this was the correct orientation.

A second test was done by walking in the city of Ghent. Figure 4 shows the results. The red lines indicate the real path at points where the prediction diverges. In the other parts, the real path is perfectly resembled by the prediction!

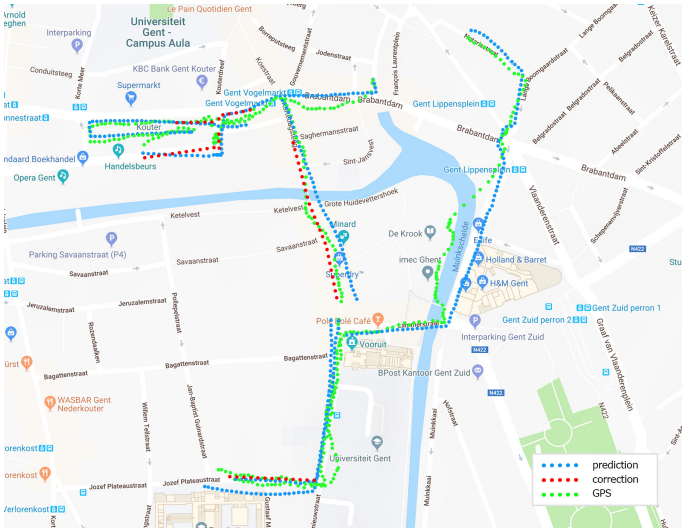


Fig. 4. GPS signal and prediction for the city walk. The red lines indicate the real path at points where the prediction diverges.

V. Conclusion

First, a system for capturing inertial motion data via low-cost inertial measurement units was built. Both hard- and software components for this system were developed. The results show that this system can be used in practical applications that require inertial motion data to be captured.

Thereafter, the use case of pedestrian dead reckoning navigation was discussed. The transformation to relative movements from the sensor data was done in two main parts: orientation estimation and displacement estimation. In the orientation part, the person's walking direction was estimated by using the IMU's fusioned data consisting of quaternions. In the displacement part, the data from the accelerometer and gyroscope were used to extract features from. A step detection algorithm was then used to extend this set of features by calculating the duration of every step cycle. Finally, these features were provided to Lasso, a linear machine learning model used for predicting the displacement.

The results of combining the orientation and displacement data prove that a low-cost IMU is capable of estimating movement of a walking human. Comparing the results to the data from a GPS module shows that the predicted position using an IMU can even outperform the result of the GPS module. In general, the prediction provides a much smoother and less noisy result which most of the time better describes the real situation.

Contents

Used abbreviations

MEMS	Microelectromechanical System
IMU	Inertial Measurement Unit
PCB	Printed Circuit Board
MCU	Microcontroller Unit
I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
SCL	Serial Clock Line
SDA	Serial Data Line
VCC	Voltage common collector
GND	Ground
SRAM	Static Random-Access Memory
MSE	Mean Squared Error
MAE	Mean Absolute Error

Introduction

Since their conception, the market for microelectromechanical systems (MEMS) has been steadily increasing, a trend that is believed to continue for the foreseeable future. The rapid development of MEMS has paved the way for many new applications, one of which is the production of very small and cheap inertial measurement units (IMU). An IMU is a device composed of accelerometers and gyroscopes that is designed to measure a body's acceleration and angular velocity. Some IMUs also contain magnetometers which measure the magnetic field surrounding a body which, when combined with acceleration and angular velocity, can be used to determine the absolute orientation of a body.
The low cost and small size of these devices has led them to be embedded in a myriad of devices e.g. smartphones, smartwatches or sports trackers. Moreover, the presence of these smart, wearable devices keeps growing. This omnipresence of inertial motion sensors has enabled many new and interesting use cases.

One of these use cases is pedestrian dead reckoning where a person's position is estimated without the use of an absolute measurement tool like GPS. Instead, the data from one or multiple IMUs is transformed in order to estimate the movement of the person.

In this thesis we will explore the applicability of small and cheap IMUs. This is covered in two major parts.
The first part describes the development of a hardware system, along with the software running on top of it. This system contains all components necessary to collect inertial data from IMUs and transfer this data to an external source.
In the second part, a technique to estimate the position of a moving person is explored. This will be done by collecting data with the IMUs and framework explained in the first part.

Part I

Inertial motion data capturing system

Systems for inertial motion data capturing are typically very expensive. Companies like XSens or Rokoko offer such systems for prizes ranging from several thousands to several ten thousands of euros. These systems are highly accurate and are often used in professional environments e.g. motion capture in movies or evaluation of the technique of professional athletes. Although these systems are very good at what they do, the target market is rather small.

On the other hand the presence of low-cost inertial measurement units (IMUs) in many consumer devices has steadily been increasing. It would therefore be very interesting to investigate whether such cheap units could enable serious competition for motion tracking.

The development of such a system requires an interplay of both hardware and software components. The hardware component consists of all elements necessary to capture inertial motion data and transfer it to an external device. The software component is responsible for coordinating the communication between the different hardware elements and transform the raw data to a representation that can be used for further processing.

In this part, we describe the development of such a system. First, the final design of the hardware system and the different design decisions that were made along the way are described. After that, an overview is provided of all the software that is developed to allow efficient communication between IMUs and external devices. The third section provides a brief overview of a tool that is developed to track and visualise the position of the sensors in 3 dimensions. In the final section, some results regarding the performance of the system are presented.

1 Hardware

To build a hardware system capable of recording inertial motion data, two components are essential: inertial measurement units (IMUs) to capture inertial data and a central module to coordinate the communication between all the components. In order to make the reader more familiar with inertial measurement units and the differences between different types, section 1.2 provides some background about IMUs. In section 1.3 these IMUs are integrated in what we call an IMU module, containing an IMU and all components necessary to allow communication with the central module. Section 1.4 then covers the development of a prototype of the central module, followed by the design of a dedicated printed circuit board (PCB) to replace the prototype. Before exploring the development of the hardware system however, first the requirements that should be met are described.

1.1 Hardware system requirements

An inertial data capturing system should, by definition, be able to capture inertial motion data. For this purpose, 1 or more IMUs need be incorporated into the system. The amount of IMUs that are necessary depends on the particular application. In order to support as many applications as possible, the amount and location should be flexible. Another important characteristic related to this is the sampling rate per sensor which is dependent on both hardware and software (described in more detail in section 2). This is again very use case dependent. In order to support the applications we are interested in, a sampling rate of at least 50 Hz per sensor is required.

Once data has been collected, it needs to be transferred to an external source for further processing. The system should therefore contain at least 1 way to achieve this. In order to improve portability, one of these methods should ideally be without a direct connection to the external device.

1.2 Inertial measurement units

An inertial measurement unit is a device that contains a combination of one or more accelerometers, gyroscopes and (sometimes) magnetometers in order to measure the linear and angular velocities and the magnetic field respectively. In a typical configuration, three of each type of sensor are present in order to cover the 3 spatial axes. Like any sensor, accelerometers and gyroscopes are prone to measurement errors. The data measured by an accelerometer is often obscured by noise, while that of a gyroscope has to cope with drift over time. However, since both of these sensors capture data about the movement of an object, their measurements can be combined in order to compensate for these errors. This way the relative heading and speed of an object can be estimated in a process called sensor fusion, which is explained in the next section. Some IMUs are equipped with a magnetometer, which on the one hand can be used to compensate for the drift in the accelerometer and on the other hand is able to provide a constant reference (magnetic north) allowing the relative heading to be transformed into an absolute one.

1.2.1 Sensor fusion

In some applications, only raw inertial measurements are needed and thus this data can readily be extracted from the IMUs. In other applications however, we are interested in measuring the relative or absolute orientation of the object an IMU is attached to.
This could be achieved by directly integrating the angular velocity, resulting in the 3-dimensional orientation of the object. Gyroscopes suffer from sensor bias however, even when properly calibrated (see section 1.2.3). Sensor bias is the small offset that can be seen in the average signal output, even when no movement is present. The presence of this small bias will cause errors to accumulate when integrating the angular velocity, leading to deteriorating accuracy in 3-dimensional orientation over time [2].

Sensor fusion is used to provide accurate estimates of 3-dimensional orientation over time, even in the presence of sensor bias. The most popular approach to achieve this is the use of Kalman filters, which use the gyroscope, accelerometer and magnetometer signals to calculate a statistically optimal, drift-free 3-dimensional orientation. It uses measurement of gravity, extracted from the acceleration signal, and Earth magnetic north to compensate for the accumulation of orientation errors over time. More specifically, when considering the Euler angles as depicted in figure 1.1, knowledge about the direction of gravity can be used to compensate for drift in the roll and pitch angles. Since the yaw angles describes

rotation around the z-axis, gravity cannot help in compensating drift errors. However, in this case the magnetic north can be used as an absolute reference to compensate for these errors. Here we have assumed gravity is pointed downward to make the explanation more clear, but direction of gravity and magnetic north can be used in general to compensate for drift errors [1].

It should be clear that sensor fusion requires quite a bit of processing of the raw inertial signals. This can be done by first reading the raw measurements from the IMU and then applying the sensor fusion algorithms on a more powerful machine (typically a computer). However, some IMUs come with an onboard microcontroller responsible for performing sensor fusion on chip. The benefit of these kind of IMUs is that they can directly output an (absolute) orientation which makes them especially user-friendly.

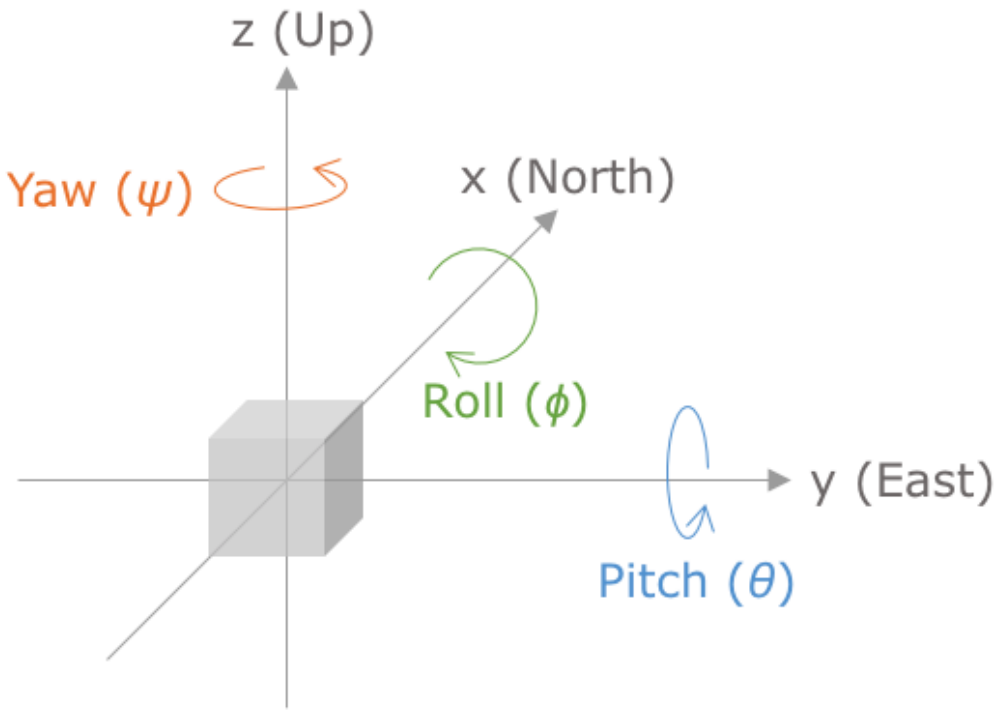


Figure 1.1: Definition and naming of Euler angles.

1.2.2 Comparison of different IMUs

Many different types of IMUs are commercially available. Depending on their grade, prices can range from a few to several thousands of euros. IMU grades are categories indicating the quality of an IMU. A very important characteristic in this regard is the bias stability of the gyroscope. Bias stability indicates how much the bias fluctuates, due to flicker noise in electronics and other effects, around the average constant bias [3]. In general, the bias stability (in degrees per hour/second) is a measure of the 'goodness' of a gyroscope [4]. Table 1.1 gives an overview of the different gyroscope grades and the possible technologies that are most often used. Since prices of even the cheapest industrial IMUs start at roughly €100 and we would like to investigate the possible uses of low cost IMUs, like the ones present in many consumer devices, we will be limited to consumer grade IMUs only.

Grade	Bias stability	Gyro technology
Consumer	30-1000°/h	MEMS
Industrial	1-30°/h	MEMS
Tactical	0.1-1°/h	FOG/RLG/MEMS
Navigation	0.01-0.1°/h	RLG/FOG
Strategic	0.0001-0.01°/h	RLG/FOG

Table 1.1: Bias stability of different gyroscope grades.
(FOG: Fiber Optic Gyroscope, RLG: Ring Laser Gyroscope)

Within the consumer grade class of MEMS IMUs, the main differences between units are the presence or absence of magnetometers and the possibility for onboard sensor fusion. At this stage of development we want to place as few restrictions as possible on the type of data that can be gathered from the IMUs. We want the system to be as flexible as possible so it can be used in as wide a range of use cases as possible. Therefore, we will only consider IMUs with magnetometers.
Finally, an important factor for the comparison between IMUs is the ability for onboard sensor fusion. Not only would this improve user-friendliness, it would also make it possible to obtain (absolute) orientations without the need for an external computer. However, since almost all use cases will need such an external device anyway for processing of inertial data, onboard sensor fusion is not vital.

Based on these criteria we have selected two possible IMUs: the Bosch BNO055 and the Invensense MPU9250. Table 1.2 briefly summarises the most important characteristics of

both sensors. At the start of this thesis, an IMU module, which will be described in more detail in section 1.3, based on the BNO055 was already available. We have developed a similar module based on the Invensense in order to be able to compare both IMUs and finally decide on which one will be used in the hardware system.

	BNO055	**MPU9250**
Communication protocols	HID-I2C, I2C, UART	I2C, SPI
Fusion on Chip	Yes	No

Table 1.2: Comparison of BNO055 and Invensense MPU9250.

1.2.3 Calibration

In order to achieve accurate sensor data, calibration of the IMU is key. It is considered one of the main challenges as it often requires interaction by rotating the device in different angles.

Sensor offsets

Figure 1.2 shows an example of sensor data from an accelerometer when there is no movement. Neither the x, y or z angular velocities have averages around zero. By estimating the offsets for each axis, these values can be repositioned around zero.

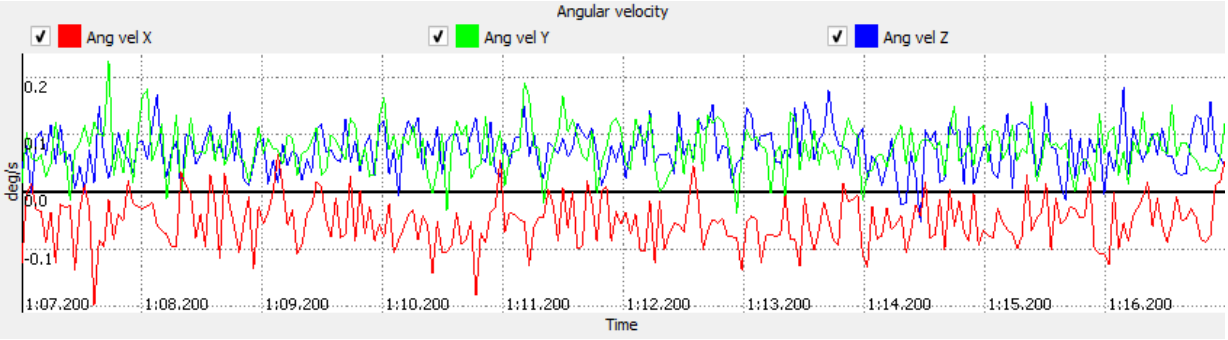


Figure 1.2: Angular velocities (x, y, z) showing gyroscope offsets. [2]

Calibration is the calculation of the sensor offsets (biases). Every sensor has a specific offset based on its physical properties and manufacturing. These properties vary over time which results in different characteristics at different moments. Depending on sensor usage

and time, the internal sensor offsets will increase, introducing drift which causes more inaccurate data over time.

The BNO055 features a built-in calibration algorithm to calibrate all three sensors (accelerometer, gyroscope and magnetometer) in the background to remove the offsets. Although this means calibration takes place out of the box, offsets are lost every time the device (and thus the IMUs) powers off due to the absence of persistent storage on the BNO055. To prevent that the calibration process needs to be started from scratch every time the IMUs are powered on, sensor offsets could be read and stored on the Arduino. When power is supplied, the offsets can be reloaded in the IMU's registers. This can speed up the initial calibration time from minutes to seconds. The BNO055 also provides information about the current calibration status. There are four statuses which can be read from the IMU's registers: SYS (system), GYR (gyroscope), ACC (accelerometer), MAG (magnetometer). These statuses hold a value from 0 (uncalibrated) to 3 (fully calibrated). The SYS status depends on the status of all three sensors.

Absolute orientation

While the calibration of the accelerometer and gyroscope are necessary to prevent drift, the main goal of the magnetometer calibration is finding the north pole. In most cases, the absence of a correct value of the magnetometer transcends the incorrectness of an uncalibrated accelerometer and gyroscope. That is, as long as the north pole is not found, the IMU in fact provides only relative measures: rotations can be measured but no conclusion can be made about the current orientation of the IMU.

Where the calibration of the accelerometer and gyroscope can reduce an error from around 1 degree to an insignificant error value, the calibration of the magnetometer can cause the angles to adopt a completely different value due to the switch from relative to absolute orientations from the moment the calibration algorithm assumes the North Pole is found.

This absolute orientation is what makes the IMU very suitable for combining multiple units. When all units (IMUs) are calibrated, their physical orientation compared to each other is described completely by the measured values. If two IMUs lie on a plane in the same manner, their measured values should be the same too. Technically, this is achieved by the fact that each IMU is calibrated w.r.t. the "same" North Pole as well as the gravity affecting the accelerometers in the same way.

1.2.4 Accuracy

The accuracy of a sensor is defined by how precise data resembles the real situation and how noise influences the results. This accuracy is the key parameter to consider when evaluating whether a given sensor performs adequately for a targeted use case. A use case as step detection can be accomplished with moderate accuracy as long as a periodic trend can be determined (each step should be a period where the correlation with other steps is still visible regardless of noise). On the other hand, visualisation of human gestures is a use case where exact orientations are important to resemble the real situation.

In order to get a view on the accuracy, the tool displayed in figure 1.3 was designed. The main part consists of a metal bar attached to a step motor.

Figure 1.3: Measurement tool for IMU accuracy.

Motor

The motor is able to perform 12800 equal steps per rotation, which resolves to an angle of 0.028125 degrees per step. The step offset should be small enough to achieve a smooth rotation when analysing the captured data of the IMUs, otherwise the steps could be visible in the results. As long as the sample rate is significantly larger than the step rate of the motor, the result should not be influenced by the steps. Section 4.3 explains the rates at which data can be captured from the IMUs. Following those results, the sample rate will never be larger than 400 Hz or thus, 2.5 ms per sample. Furthermore, slow motor rotations are negligible if a decent accuracy is measured at moderate or fast rotation speed. Lowering the speed will then only increase the noise created by the motor.

Eventually, a representative motor speed seemed to be around 40 rotations per minute or 1.5 seconds per rotation. This value is estimated in comparison to the speed of a human arm gesture with the visualisation use case kept in mind. This results in around 117 µs per step, an order of magnitude smaller than the sample rate and thus fulfilling our requirement for smooth rotations.

Acceleration

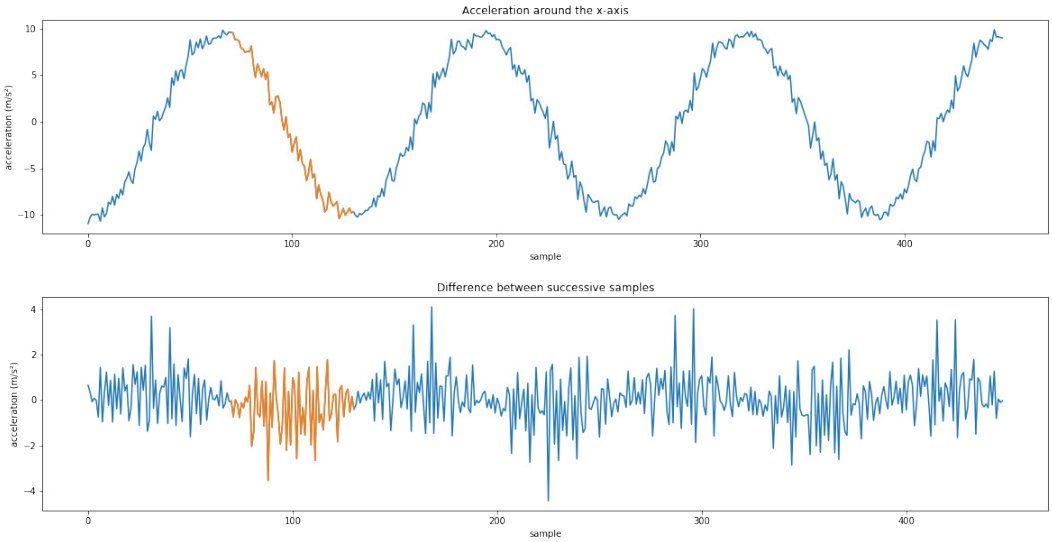


Figure 1.4: (a) Acceleration around the x-axis. (b) Difference in acceleration between successive samples. The orange part is further investigated in figure 1.5.

The upper graph in figure 1.4 shows the resulting acceleration around the x-axis (the axis capturing the rotation in this case). The graph captures three complete rotations as can clearly be seen by the sine waves. It is worth to note that this concerns linear acceleration as the motor rotates with a constant speed and thus without radial acceleration. Thanks to the (unfiltered) gravity, sine waves can be observed. Next to this, it seems noise is present continuously too. The lower graph investigates this further by showing the difference between a sample and its predecessor. It is clear the mean difference lies around zero with the ascending fragments slightly above and the descending slightly below this mean.

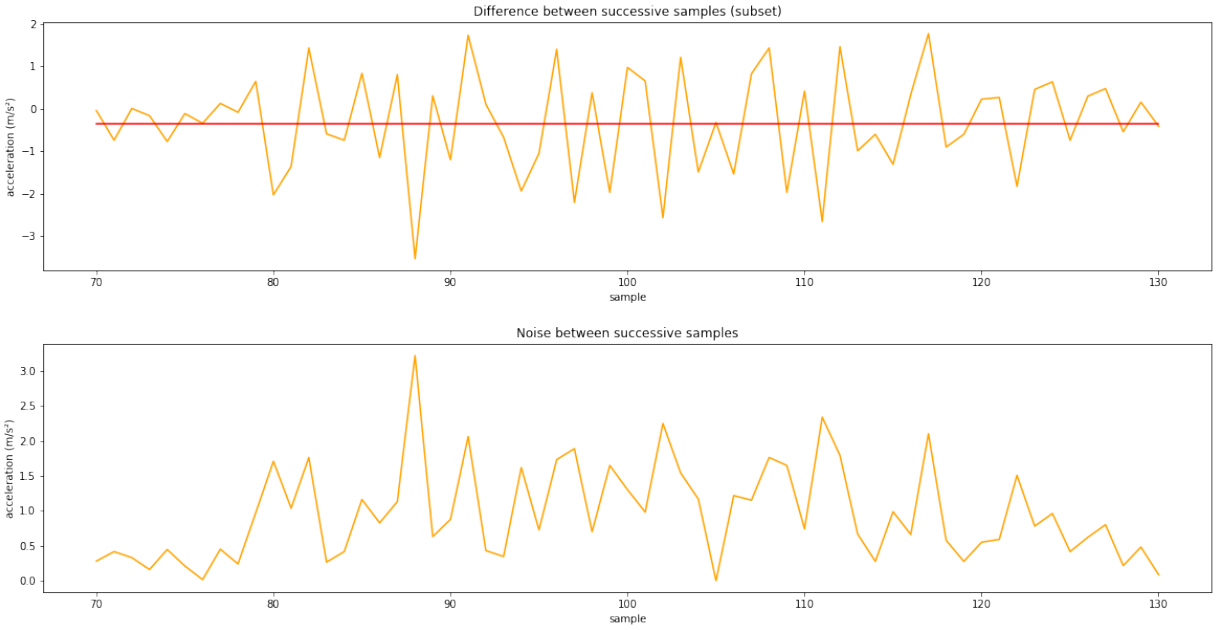


Figure 1.5: (a) The same difference in acceleration between successive samples for the region coloured in figure 1.4. The red line displays the mean value. (b) Noise between the successive samples of (a).

Figure 1.5 zooms in on a subset of the data, the orange coloured part of figure 1.4. The mean difference in acceleration of this subset can be found in table 1.3. The lower graph of figure 1.5 investigates the noise of the data in the upper graph.

	mean	min	max
accel. (m/s^2), upper graph	-0.32	-3.54	1.78
accel. (m/s^2), lower graph	0.95	0.00	3.22

Table 1.3: Statistics of the data in figure 1.5.

Table 1.3 also provides the mean noise with magnitude almost three times the magnitude of the mean difference in acceleration between samples. As a conclusion, the acceleration could be used to determine a trend (cfr. step detection) but seems to contain too much noise for calculating movement (cfr. human visualisation).

Quaternions

Next to the acceleration, quaternion data is observed. As explained in section 1.2.1, the quaternion output combines the best of three worlds: acceleration, gyroscope data and magnetic field. Two remarkable differences with the acceleration data above are (1) gravity is filtered and (2) orientations are absolute based on the north pole and gravitational force. Figure 1.6 shows the quaternion components for the same sample range as the acceleration data from figure 1.4, making three complete cycles. Unlike the acceleration data, almost no noise is observed. After converting the quaternions to Euler angles and transforming the base axes, a more interpretable result is obtained.

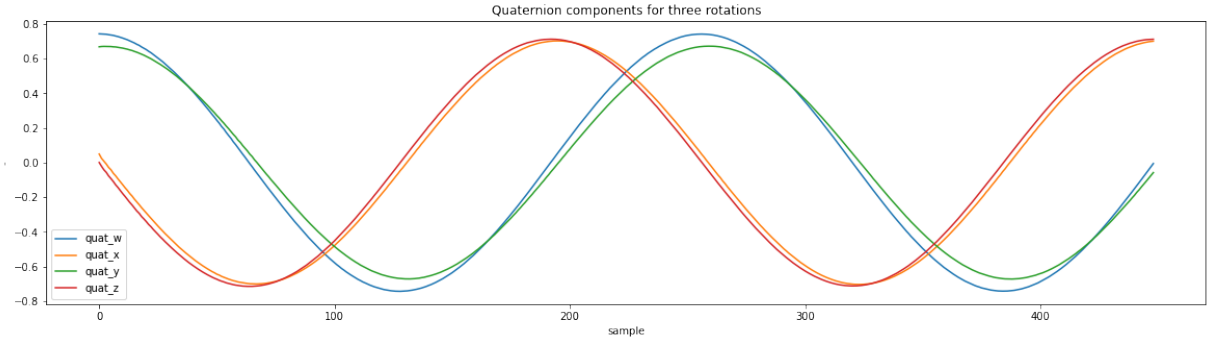


Figure 1.6: Quaternion components for a sample range of three rotations.

The upper graph in figure 1.7 describes the orientation of the IMU around the motor axis throughout time. Values lie between]-90,90] as the result is always the smallest (absolute) difference to a fixed orientation, in this case when the metal bar is vertically aligned and the IMU is on top. As opposed to the quaternion graph, the three cycles are clearly visible. The lower graph shows the difference in degrees between successive samples. Ignoring the jumps due to the]-90,90] range, noise can be observed on the horizontal pieces.

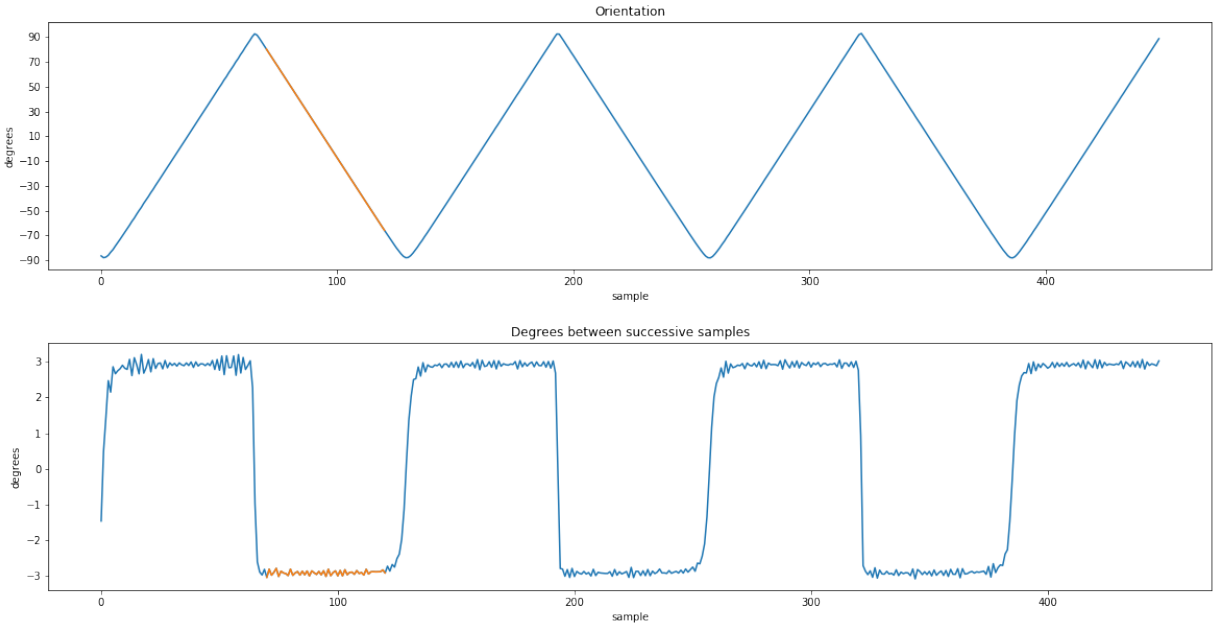


Figure 1.7: (a) Orientation of the IMU around the motor axis. (b) Difference in degrees between successive samples.

Figure 1.8 zooms in on the same subset as in the acceleration showcase. This time, the mean noise is almost sixty times smaller than the mean difference in degrees between successive samples. Table 1.4 provides more detailed statistics. As a result, the benefit of the fusion data is clearly demonstrated. It seems feasible to use this data in a use case as step detection as well as a more complex use case as calculating movement of human gestures.

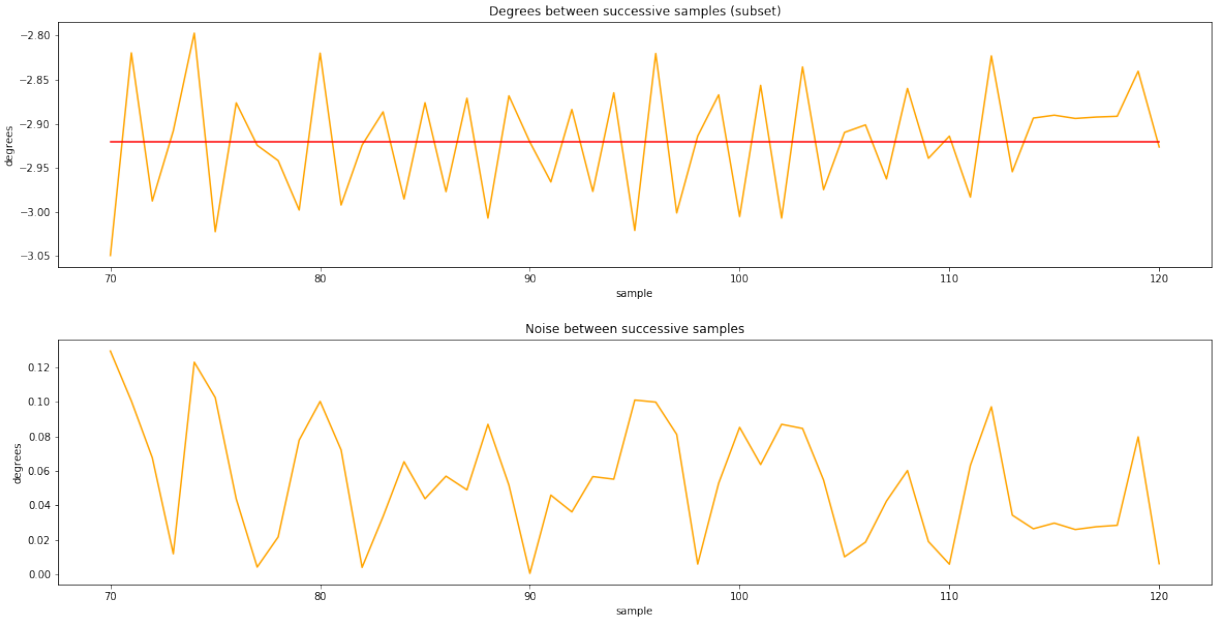


Figure 1.8: (a) Difference in degrees between successive samples. (b) Noise between successive samples.

	mean	min	max
angle (in °), upper graph	-2.92	-3.05	-2.80
angle (in °), lower graph	0.05	0.00	0.12

Table 1.4: Statistics of the data in figure 1.8.

1.3 IMU module

Armed with the knowledge about IMUs provided in the previous section, we can now start with the development of a PCB containing all necessary components to allow data to be read from the IMUs: the *IMU module*. This module should not be confused with a bare IMU sensor, which is the central component of an IMU module. Remember that two different IMUs were selected in the previous section, the BNO055 and the MPU9250. The PCB design for the BNO055 had already been made and based on this we designed a similar module for the MPU9250. The remainder of this section describes the most important design decisions that were made to end up with the final design of the IMU module depicted in figure 1.9.

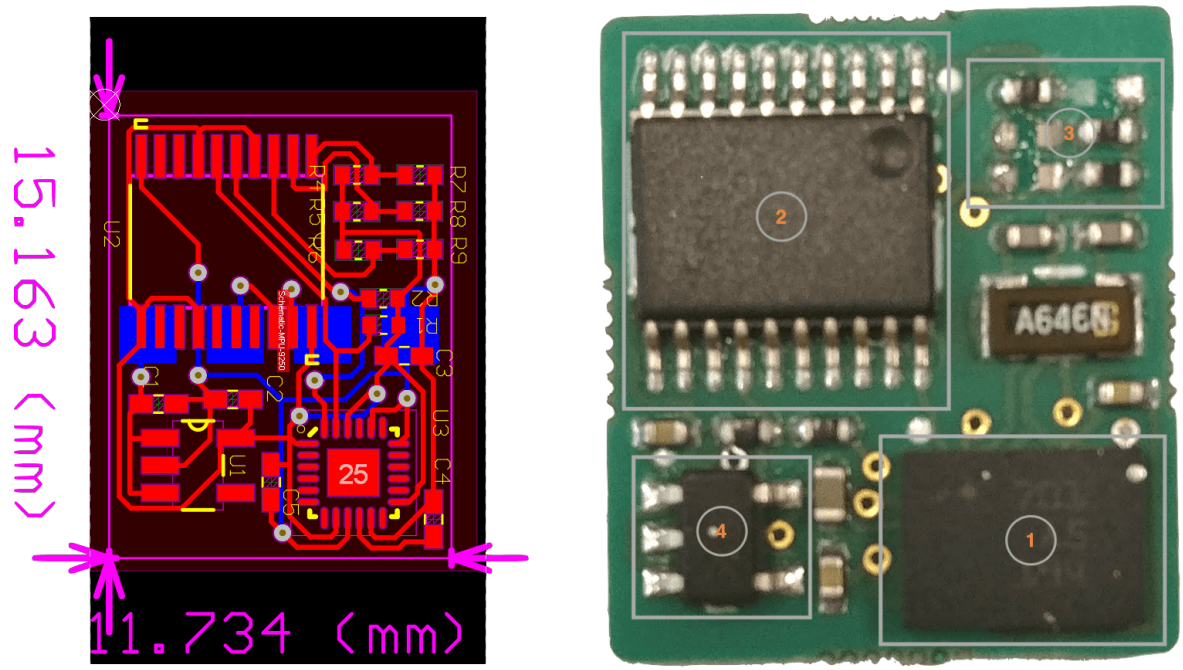


Figure 1.9: Illustration of BNO055 based IMU module.
Altium design on the left, completed PCB on the right.

1.3.1 Communication protocols

One key difference between the BNO055 and the MPU9250 is the kind of communication protocols they support. Both are able to communicate with a controller via Inter-Integrated Circuit (I2C). Moreover, the BNO055 is also able to communicate via Universal Asynchronous Receiver-Transmitter (UART) while the MPU9250 can also use Serial Peripheral Interface (SPI).

Since UART is inherently a one to one connection between two devices and the hardware system should be able to contain multiple sensors all communicating with the same central controller, this way of communication is not an option. This leaves only one option, I2C, for the BNO055 modules. The MPU9250 modules also allow communication via SPI, which in general allows for higher transfer speeds. However, as opposed to I2C, which only needs 4 wires regardless of the number of modules connected to the bus, SPI needs 1 extra

wire per connected module. Because of these extra wires, combined with the fact that the BNO055 modules already use I2C, we opt to use this protocol for the MPU9250 modules as well.

I2C is a communication protocol that allows the connection of multiple integrated circuits to the same communication bus. It requires 2 wires, one carrying a clock signal (SCL) and another one for the data signal (SDA). In order to allow multiple devices to be connected to the same bus, an addressing scheme is used which allows up to 112 different addresses. A master device initiates communication by sending the address of the slave device over the SDA line. Every device knows its own address and thus knows when it is being addressed by another (master) device. Because both addressing and data transfer is done over the same wire, no extra chip select wires are necessary as is the case in SPI. The I2C address of a device is generally fixed, possibly to a certain range, and chosen by the manufacturer. In case of the BNO055, the least significant bit of the I2C address can be configured depending on a high or low signal being present on input pin COM3 of the IMU, resulting in 2 possible addresses. The same applies for the MPU9250, where the least significant bit can be configured by setting the AD0 input pin high or low.

1.3.2 I2C Multiplexer

Recall that an important requirement of the hardware system is that it should be possible to easily connect multiple sensors. However, the IMUs only allow 2 different I2C addresses to be configured, which is insufficient for many practical use cases. We therefore need a second component that would allow us to assign different I2C addresses to the IMUs. Ideally, this would be a component of which the I2C address can be configured and that would simply forward signals from its input to its output whenever it is being addressed. Such a component could not be found however, so we opted to use an I2C multiplexer, the TCA9548A, which is able to achieve the same end result.

The multiplexer, of which a simplified diagram is shown in figure 1.10, makes it possible to connect up to 8 devices with the same I2C address to the same bus. Like the IMUs, the multiplexer also has an I2C address that can be configured within certain limits. Unlike the IMUs however, the multiplexer has 3 address pins instead of 1, allowing 8 different addresses. The way these addresses are configured is again by connecting the 3 address pins to either VCC or GND (each possible combination resulting in a different address). Since the connection of these pins is done in the hardware design, it would be necessary to design 8 different modules, all having a different address configuration. In order to be able

to choose the address even after the IMU module PCBs are printed, each address pin is connected to both VCC and GND through a resistor. When assembling the boards, we can then use three 0Ω resistors to connect each pin to either VCC or GND and thus configure the I2C address.

Because the IMUs are now connected to the bus through the multiplexer, an extra step is required before communication can take place. In order to achieve the forwarding functionality we are looking for, all IMUs are connected the same gate (gate number 2) of the multiplexer. On startup, each multiplexer is set to gate number 3, effectively disconnecting all IMUs from the bus. Whenever an IMU needs to be read out, the corresponding multiplexer gate is set to 2, connecting the IMU to the bus. Once read out has finished, the gate is set to 3 again, disconnecting the sensor from the bus again. The framework running on the central module, described in section 2.2, handles the switching of the multiplexer gates so this becomes transparent for the end user.

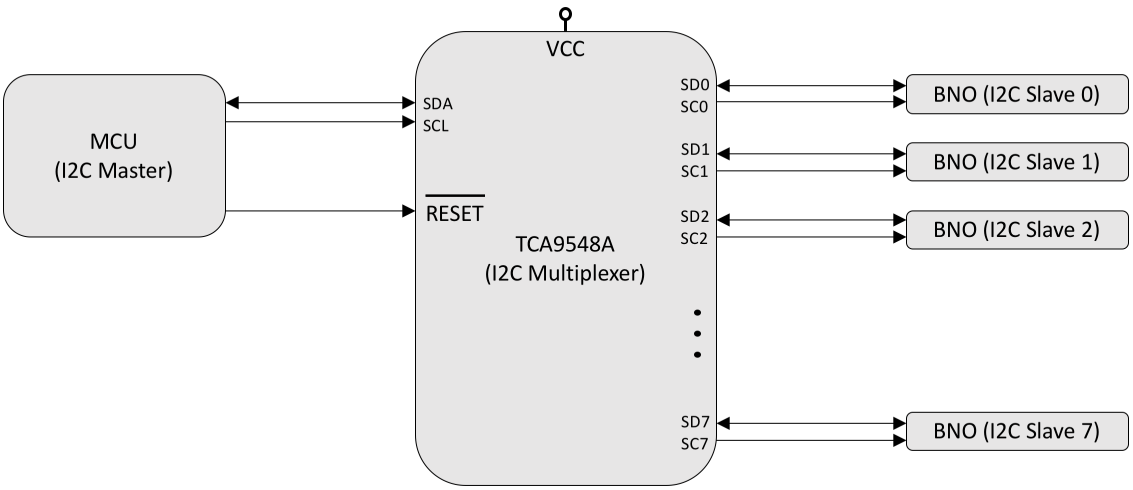


Figure 1.10: Simplified diagram of an I2C multiplexer.

1.3.3 Voltage regulator

As described in section 1.4, the central module uses a 5 V power source, while both IMUs operate between 2.4 V and 3.6 V. Because an extra power supply would decrease the portability of the suit, we add a voltage regulator to the IMU modules to lower this 5 V input voltage to 3.6 V, which can be used as input to the IMUs.

1.4 Central module

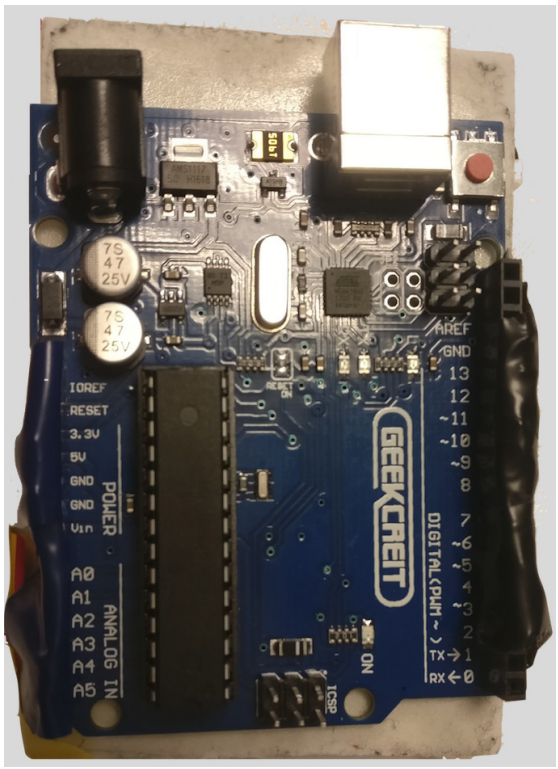
The central module has several functions: it needs to coordinate the communication with the different IMUs, transfer the data from the hardware system to an external device and provide power to all connected components. In order to be able to assess the impact of different design decisions, an initial central module, based on an Arduino Uno is made. Using an Arduino makes prototyping much cheaper, faster and easier. Most importantly, it allows us to try out different ways of transferring data to external devices and since programming an Arduino is very user-friendly, it makes development of the software that will eventually run on the hardware system much easier.
Once a functional prototype is developed, it is converted to a dedicated PCB. After all, an Arduino is very useful for prototyping, but the robustness and portability leave something to be desired.
The following two sections describe the development process of the prototype up to the final version and the design of the dedicated PCB, respectively.

1.4.1 Prototype

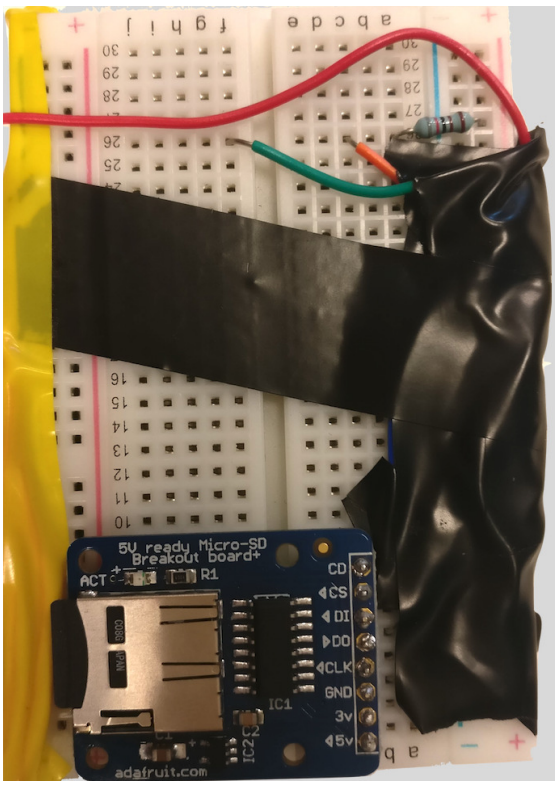
The development of the prototype of the central module happens in parallel with the development of the software framework (described in section 2) that will run on it. Initially, a single IMU module is connected to the Arduino via I2C and data is read out via a serial connection with the laptop. With the inclusion of the sensor scanning functionality in the software framework, a breadboard is attached to the back of the Arduino to allow multiple sensors to be attached at once. The addition of this breadboard also makes it possible to include an SD card module (see figure 1.11), which provides a more portable alternative to transfer data to an external source. The original prototype also included a Bluetooth module as a second alternative to transfer data. However, because the Bluetooth functionality was never used in practice and would significantly increase the complexity of the receiving end of the software framework (described in section 2.3, it is not included in the final design. It should also be noted that this design is based on the BNO055 IMUs because they were already available at the start and the development of the MPU9250 modules took some time.

The final prototype allows up to 8 IMU modules (see section 1.3.2) to be connected to the central module. Data from these modules can be read out via a serial connection (USB) or can be written to an SD card. In the latter case, a button is added to the breadboard

in order to be able to start and stop data collection. However, this setup proves to be unstable at times. Depending on the conditions and the way the modules are fixed to the body, data sometimes fails to be written correctly to the SD card. The serial connection on the other hand proves to be very reliable, especially with the addition of a small piece of software to start data collection with a press of a button on the laptop to which the central module is connected.



(a) Front of the prototype central module consisting of an Arduino with wires to IMU modules and SD card unit. Wires taped tightly together to ensure reliable connection.



(b) Back of the prototype central module where the SD card module can be clearly seen.

Figure 1.11: Prototype of the central module.

1.4.2 Printed circuit board

The final prototype as described in the previous section works as expected but leaves something to be desired with regard to reliability, robustness and portability. A printed circuit board (PCB) bringing together all components present in the prototype is developed

to deal with these weaknesses. In the first part of this section, the design of this PCB is presented. The second part describes the extra steps required to run the developed software framework on the microcontroller unit (MCU) contained in the PCB.

PCB design

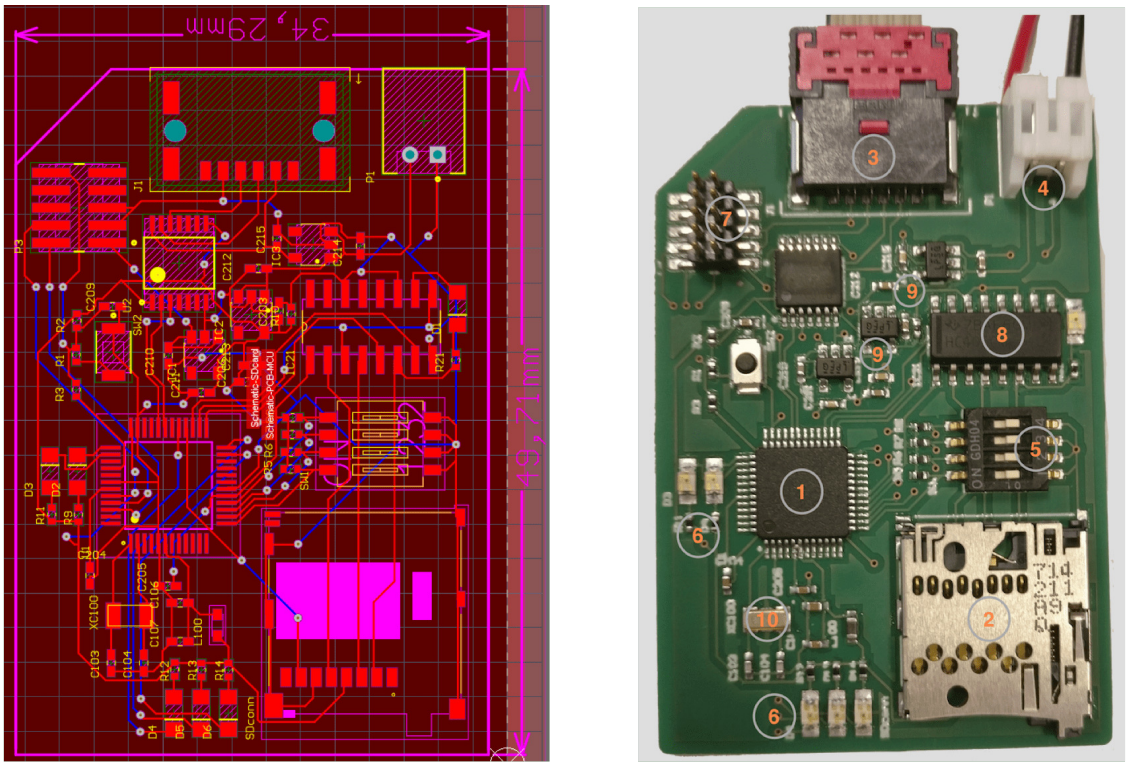
The final PCB is depicted in figure 1.12. The components that are present on the final prototype are also present on the PCB. In addition, a DIP-switch has been added to allow for some degree of configurability. This might for example be useful to choose between different specifier setups (for more information about specifiers see section 2). Five indicator LEDs have also been added to visualise the status of the device. Two connectors are present, one for the battery and one for the I2C bus, to which all the IMU modules are connected as well.

One major change compared to the prototype is the kind of MCU that is used. The prototype uses an Arduino Uno and thus an ATmega328P, which only possesses 2 kB of static random-access memory (SRAM) memory. As explained in section 4.2, this limits either the maximum number of sensors that can be read out, or the amount of data that can be read out from each IMU module. For this reason, a switch is made to the Arduino Zero, which uses an ATSAMD21G18 MCU. This MCU has 32 kB of SRAM, which is better suited for our application.

Running the software framework on the PCB

The software framework running on the central module is designed with the Arduino development environment in mind. It makes use of many Arduino libraries and implements the `setup` and `loop` like any other Arduino sketch [6]. The `setup` function is called once at the start of the program after which the `loop` function is called over and over again. However, because many components of the Arduino were stripped, including the USB port which is used to program the prototype, the standard Arduino environment cannot be used anymore to program the MCU.

In order to get a working PCB, the first step that needs to be taken is burning the Arduino bootloader on the MCU [7]. To this end, Atmel studio, a software package used to develop code for Atmel MCUs, is used. The next step is cross-compiling the Arduino code to machine code that can be run on the MCU, which is again accomplished using Atmel studio. Finally the obtained machine code can then be uploaded to the Arduino.



1: ATSAMD21G18 MCU
2: SD card connector
3: ERNI connector connecting central module to I2C bus
4: Battery connector
5: Switches to configure different modes
6: Indicator LEDs
7: Connector for MCU programmer
8: Voltage level translator
9: Voltage regulators
10: External clock

Figure 1.12: Illustration of the final PCB of the central module. Altium design on the left, completed PCB on the right.

1.5 Bringing it all together: suit prototype

Having developed the IMU modules, central module and both software frameworks for the sending and receiving side, the final task is to build some sort of 'suit' to which these modules can be attached. Here a prototype with improvised connections is developed. In parallel with our efforts however, work has been done to develop more robust connections which can in the future be combined with our modules and frameworks to come to a robust and easy to use inertial motion capture suit.

1.5.1 Sensor locations

The locations of the sensors are roughly based on other popular inertial motion capture suits [9]. Two prototype garments are developed, a long sleeve t-shirt and long sleeve pants to which IMU modules are attached with tape. The I2C bus is formed by 4 standard electrical wires and IMU modules are connected to this bus van screw terminals. Because long-term attachment of IMU modules to textile using tape proves to be challenging, only one sleeve of the t-shirt and pants is provided with IMU modules. However, if some particular application demands more sensors, these can easily be added.

Figure 1.13: Prototype of the IMU suit with sensors attached to an arm and leg.

2 Software

In this section we describe the software frameworks which control the data collection and processing. The software contains 2 frameworks, one for the sender and one for the receiver. The sender side framework is deployed on the microcontroller and handles all sensor and output management. The receiver's software is a Python framework to handle incoming bytes generated by the suit.

We will refer to both as *the software*. We call the software deployed on the microcontroller the *Arduino framework* and the processing software used by the computer the *Python framework*. First we discuss the design objectives and the resulting requirements of the software. We then specify the internals of the software.

2.1 Software requirements

Our software is created with multiple design objectives in mind. It needs to be flexible in reading out different sensors, different numbers of sensors, specifying which data is needed and choosing where to write the output to. This allows the system to be used for a variety of use cases. In contrast to the flexibility, the data collection needs to be fast enough for real use cases. Following these objectives we pose the following requirements for the Arduino framework:

- Program size and memory usage

  The Arduino framework should fit on the used microcontroller. For development, we will deploy this software on the Arduino Uno which uses the ATmega328P microcontroller. The available flash memory for program storage is 32 kB. Next to the flash memory, the ATmega328P has 2 kB of SRAM for storage of variables during execution. Of this capacity, 0.5 kB is already used by the bootloader. This very limited memory requires our software framework to be very careful with memory allocation.

- Readout speed

  Our program will take control over which sensors to read and when to read what information. We define the readout speed P as the frequency at which the software is able to generate a data point for some number of sensors n and some setup s. Every datapoint contains a sensor identification, a timestamp and data specified by the setup s. These datapoints can later be analysed in different use cases. Examples include *Limb monitoring*, *Activity recognition*, *Position detection from movement data*, *Gait authentication* and many others. All of these use cases analyse human motion, we require our suit to accurately capture this movement. Karantonis et al. show that 99% of human movement energy is contained below 15 Hz [10]. To make sure we capture enough relevant data for all use cases, we require to read out quaternion and calibration data for 8 sensors at 50 Hz.

- Data generation

  We should be able to gather all data required for a movement analysis. It should be able to read out accelerometer, gyroscope, magnetometer, quaternion, euler angle and calibration data. All of these are generated by the inertial measurement unit. Next to these the software framework will have to add the sensor id and a timestamp to form a data point.

  These data points need to be exported either to some form of external storage or via a USB/Bluetooth connection. For practical reasons, we require to have a microSD and USB compatible system.

- Flexibility

  We require the Arduino framework to be flexible in specifying what data to read out. Not only should it be easy to change what data is being read out by the program, completely new data sources should also be easy to integrate. Extending this concept, we want to be able to integrate new types of sensors. Similarly, the output location should be configurable and new types of output should be easy to integrate. Finally, we want the framework to be able to change configuration dynamically, during execution.

- Plug 'n play

As we will use our suit in experimental setups, it needs to be ready as soon as possible, with as little configuration as possible. Therefore we require the Arduino framework to be able to scan the presence of sensors on startup and collect data for all of them.

2.2 Arduino framework

In this section we discuss the Arduino framework construction. We clarify the design choices and their relation with the objectives. In Figure 2.1, an overview of the entire software is plotted. Three parts can be distinguished: a configuration, read out and processing phase. First the user specifies what the Arduino framework needs to do, next the microcontroller reads out the sensors and afterwards the generated bytes are parsed in the Python framework. Here, we will take a closer look at how the sensors are read out and a bytestream is generated.

2.2.1 Sync & scan

Before reading out the sensors, the managing component of the framework, the `sensorManager`, initialises the system. As a first step, it needs to link the microcontroller with the sensors. The I2C protocol is used for the communication. Arduino's *Wire* library conveniently lets us read and write bytes over the communication bus. First the MCU joins the I2C bus, then the clock speed is initialised with a value ranging from 100 000 Hz to 400 000 Hz.

Next, the framework scans the I2C bus for sensors. As stated in section 1.3.1, all IMU's have the same I2C address (`0x28`) and a multiplexer enables us to differentiate between the sensors. To scan the system, the framework first traverses all 128 I2C addresses. It initiates a transmission for every address and keeps track of the addresses which respond with an acknowledgement. After the I2C addresses with devices are found, the systems checks the different multiplexer gates for sensors. If a sensor is found it is added to the `sensorManager sensors` list. Here, each `sensor` is an abstraction of the real IMU breakouts.

Once the sensor locations have been found, the manager initialises the sensors. First it is checked that the found location actually contains a sensor, by reading out the id register and checking the correctness. Next, the manager turns on the use of an external crystal oscillator to get the best performance from the IMU. Finally the system checks if it has stored calibration information for the sensor. If so, it initialises its offsets with the saved

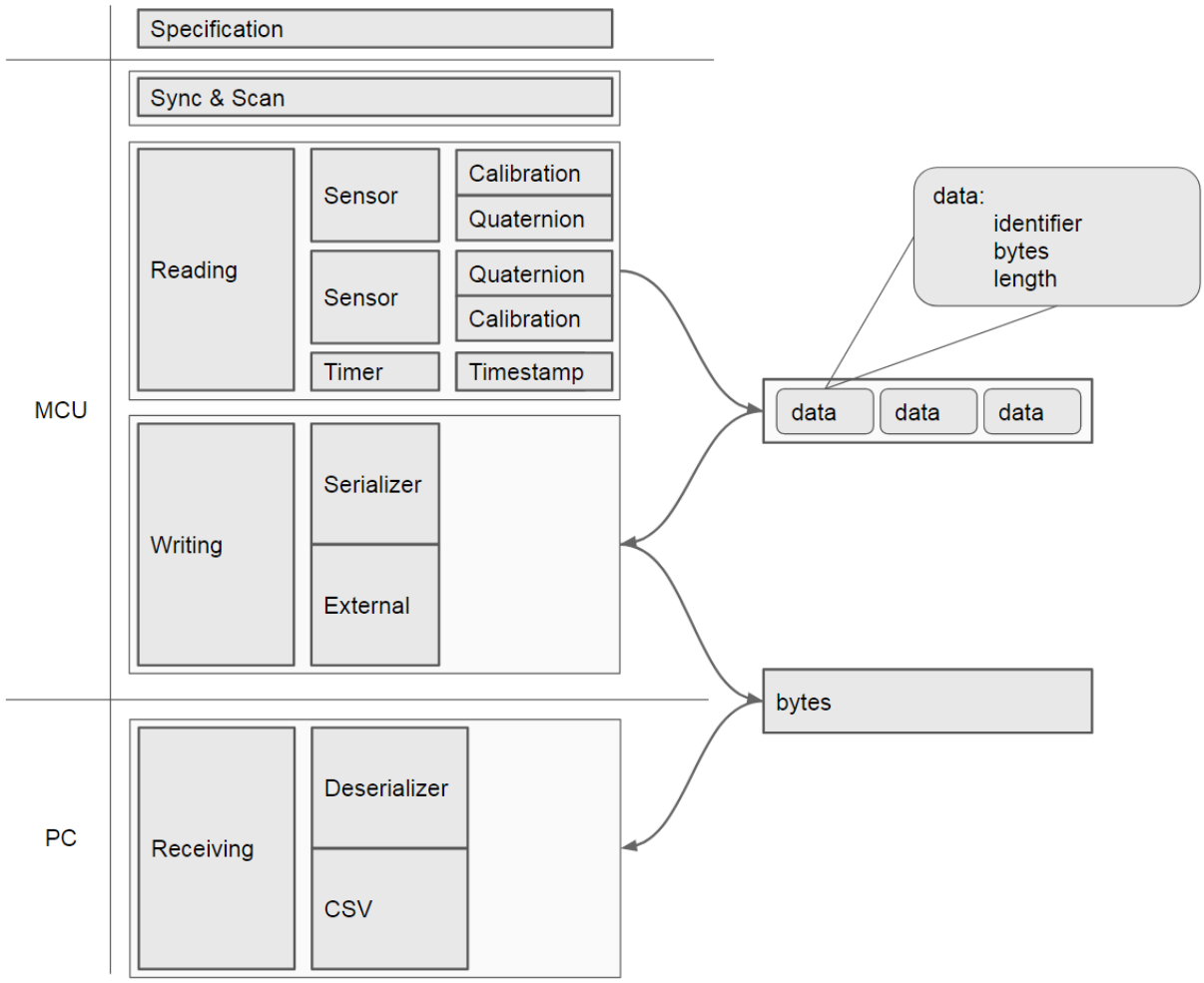


Figure 2.1: On overview of the software framework.

values. If not, we can instruct the IMU to calibrate and the framework will automatically store the information.

As well as setting up a connection with the sensors, a connection with the output needs to be put in place. In our framework we refer to the output location as the `external`. Each `external` is an abstraction of a physical output location. Two possible output locations are implemented: USB and SD. A USB connection uses serial communication and is started on construction of the USB object. The SD card uses SPI communication and is started using the Arduino *SD* library.

Finally the system is started by writing a marker to the `external`. This marker is a 25 byte sequence that will be used in processing to identify the start of the data blocks.

2.2.2 Data readout

Once all connections are set up, the manager can start reading out the sensors. In this section we discuss how we can specify what as well as when data is requested. We also describe the actual read out routine.

Specification

A variety of data can be requested from the sensors and the microcontroller. To form an interface for easily reading out different information sources, the system uses the abstract class `Specifier`. Each `specifier` has an `identifier`, a `period` and a counter `progress`. Each class implementing this interface provides a `getData()` function.

The `getData()` function will retrieve the actual data. The `period` and `progress` are used to accurately specify when the data is needed. That is, if the specifiers are placed in an array and the `getData()` function is called iteratively, we can choose to read out the data every *period* times the function is called. When this function is called, it wraps the data in a `Data` object. This object contains the actual data as bytes, keeps track of the length (in bytes) and has an id to identify the type of data that is contained in the block.

Each `sensor` has an array of specifiers, each of which can be changed dynamically. The `sensorManager` itself can have specifiers to provide other (meta-)information. An overview of the different specifiers is given in Table 2.1.

Name	Location	Description	Size (bytes)
Accelero	IMU	Accelerometer data for every axis	6
Gyro	IMU	Angurlar velocity for every axis	6
Quat	IMU	Four quaternions represent sensor orientation	8
Calibration	IMU	IMU calibration status	1
Id	MCU	Multiplexer address of a sensor	1
Timer	MCU	Average timer measurement	4
Timestamp	MCU	Current timestamp	4

Table 2.1: Data sources are described in specifiers. These abstract classes are combined in a list to define all requested data.

We note that the implementation of a `specifier` is no concern for the framework, as long as the `getData()` function is implemented. This architecture allows for example a `TimerSpecifier` to include a timer which can be started and stopped. When requesting its data, the result will be packed correctly, ready to be used in the bytestream.

Another example of the strength of this architecture is the re-usability of specifiers. For example, we have created a `DataSpecifier` for the BNO055 IMU. This specifier is initialised with a `sensor`, a register and number of bytes to read. When called, this `specifier` will read out the number of bytes, starting from the register for this given sensor. Using this component as a building block for other specifiers, we can easily request all sorts of data, without worrying about the underlying protocols.

Read out routine

Once all initialisation and specification is ready, the `sensorManager` can start collecting data. On every `read` call the manager receives from the main loop, he will generate data objects according to its own specifiers and those of the sensors.

When cycling through `sensors`, the manager will call the `sensor`'s `read` function. Here, the `sensor` object will first `select` himself on the I2C bus. That is, requesting the attached multiplexer to open the gate to the sensor. Doing so enables the microcontroller to contact the IMU on address `0x28`. Next the data is read out according to the `sensor`'s `specifiers`. Once all data collection is done, the `sensor` will `deselect` itself by requesting its multiplexer to switch gates, effectively opening up the `0x28` address for other sensors to be read out. This process continues until all `sensors` have had their turn.

Data write out

Finally, the data is transformed into bytes and written to the chosen output. The way the data is transformed is specified by the `serialiser`. The serialisation function implemented by this component turns an array of `Data` objects into a byte stream. This function is important, because the inverse deserialisation function will later be used to retrieve the data.

The most basic serialisation function takes a `data`'s bytes and pastes its `id` in front. However, we leave the possibility for more complex schemes. For example, if a fixed read out scheme is used, we could not write the `id` every time, but only send it once at the start of the session. If data size were an issue, quantisation or real time data compression could be an option.

The `External` class is used as an abstract base class for all outputs. All outputs implementing this interface provide a `write` and `close` method, enabling the manager to write out bytes and correctly close the stream.

2.3 Receiver

As stated above, the processed output can be written to an SD card or passed over a serial connection (USB). Since the data is packed as compact as possible, a cross-platform tool is created to further process the data on a computer. The tool does a similar job as the part of the framework on the Arduino: first the data is read, then it is processed and finally the processed form is written to either a file or the display.

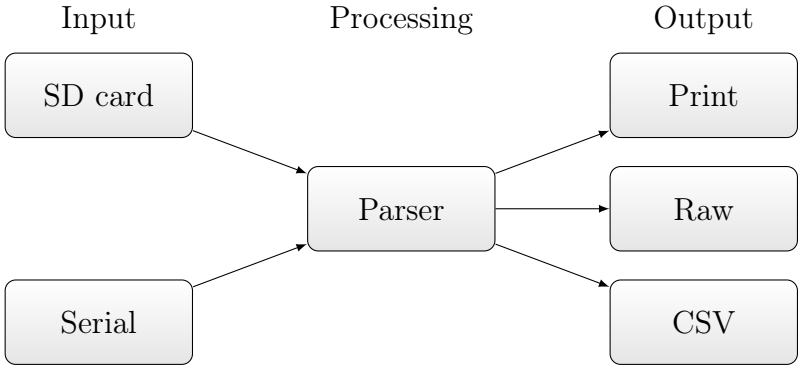


Figure 2.2: Structure of the receiver tool from input to output.

Figure 2.2 shows the structure of the program. The data is read from the file on the SD card or through the serial connection. This last option enables live analysis of the data, e.g. to see which movements have impact on the calibration status. Once raw bytes are received, they are processed by the parser which converts the bytes to interpretable data structs. The parser is able to handle real-time data and stores incomplete data into a bytes buffer. At last, an output is chosen to which the parser will pass the data structs. Using the $PRINT$ output immediately displays the incoming data. The RAW and CSV outputs create a new file and store the processed data in an efficient or formatted way, respectively.

3 Visualisation

Section 1.2.4 shed some light on the data quality of the IMU. Next to graph plotting, a 3D visualisation tool was made for multiple purposes.

3.1 Virtual IMUs

On the one hand, the absolute orientation of multiple IMUs can easily be investigated and compared by recreating them in a virtual space.

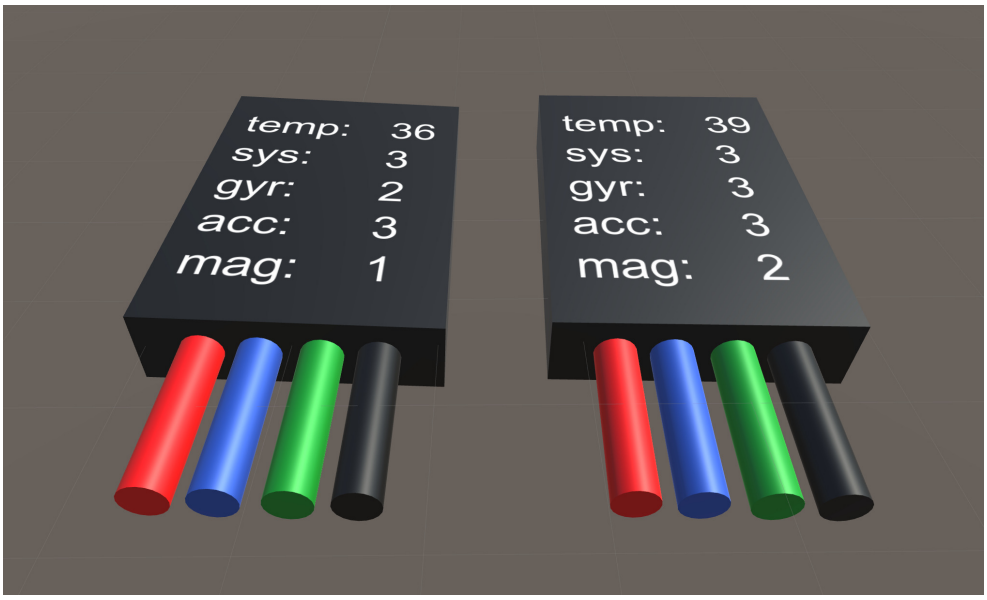


Figure 3.1: Snapshot of two IMUs next to each other. The temperature and calibration values are displayed on top.

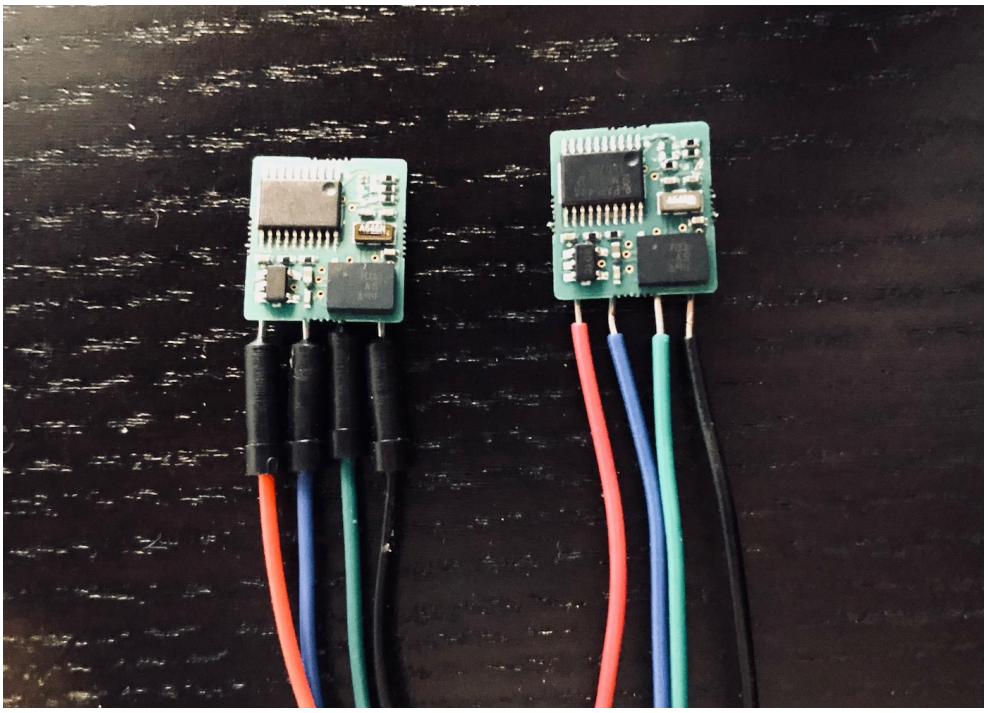


Figure 3.2: The real situation related to figure 3.1.

Figure 3.1 shows a snapshot of two IMUs lying next to each other. The red, blue, green and black cylinders represent the wiring of the IMUs. The temperature and calibration statuses are updated and displayed on top of the virtual IMUs. The real situation is shown in figure 3.2.

Connecting multiple sensors at the same time results in some useful observations. Firstly, orientations don't seem to represent the real situation at all as long as the SYS status remains zero. As described in section 1.2.3, this is due to the magnetometer still searching the north pole. The moment SYS becomes greater than zero, the virtual IMU suddenly jumps to the expected orientation. Secondly, it seems high accelerations are captured well and movements look smooth continuously.

3.2 Humanoid

On the other hand, a particularly interesting use case of a motion tracking suit is the real-time visualisation of human movements.

Figure 3.3: A humanoid with six sensors attached.

Figure 3.3 shows what a so called "humanoid" looks like. The goal is to resemble as accurately as possible the movements of the human wearing the suit. This technique is mainly used in the gaming and film industry to animate characters without the need of a studio with special cameras. Typically, highly accurate and expensive sensors are used for such a use case. Although the premised industries mostly focus on accuracy instead of pricing, it is worth investigating the performance of a low-cost IMU which can drastically decrease the cost of a motion tracking suit.

The main idea is to attach a sensor between each joint of the body. E.g. placing a sensor at both the upper and lower arm makes it possible to completely determine the

movement of the elbow joint. Since the lower arm can be simplified as being a straight connection between the elbow joint and the wrist joint, the movement is simply determined by gathering the absolute orientations of both sensors. The most important part is the position and orientation of the virtual sensors which should be as close as possible to the real situation in order to correctly calculate the orientations of each joint. Figure 3.4 zooms in on the position and orientation of the sensors.

Figure 3.4: A more detailed look on the IMU positions.

As a conclusion, angles vary smoothly and even fast movements can be made. The accuracy is mostly determined by two aspects: the accurate alignment of the virtual sensors and the calibration status of the real sensors. All sensors must have a SYS calibration status larger than zero before the angles are compared and movements are calculated. This because sensors can only be compared when they output angles are relative to the north pole. This means that a high performance motion tracking suit can be made by using low-cost IMUs as long as a good calibration algorithm is present together with software mapping the position and orientation of the virtual sensors onto the real ones.

4 Experimental setup & results

In this section, we will have a look at some performance results of the final prototype of the system. First, the capabilities of the hardware system are compared with the requirements that were imposed. Next, some experiments to evaluate the performance of the software framework regarding memory usage and readout speed are defined and the results are presented. Finally, the flexibility and plug 'n play requirements of the software framework are briefly investigated.

4.1 Hardware capabilities

Here we will have a look at the requirements stated earlier and see whether they are met by the hardware system. As already described in section 1.3.2, the maximum number of IMUs that can be attached to the system is 8. Although this is lower than commercially available solutions like the one offered by XSens, 8 sensors is sufficient for many practical use cases. Results regarding the maximum sampling rate or readout speed for each sensor are covered in section 4.3.

The capabilities for external communications were described earlier as well. Data can be transferred to an external device over a serial connection (USB) or it can be stored on an SD card. The last option makes it possible to use the suit anywhere, without the need for an external device present.

4.2 Program size and memory usage

As our framework has different setups, the program size and memory usage varies according to the setup. We immediately notice that program size is no issue, in its largest state the framework, bootloader and text files takes up 22660 bytes or around 70% of the total available memory.

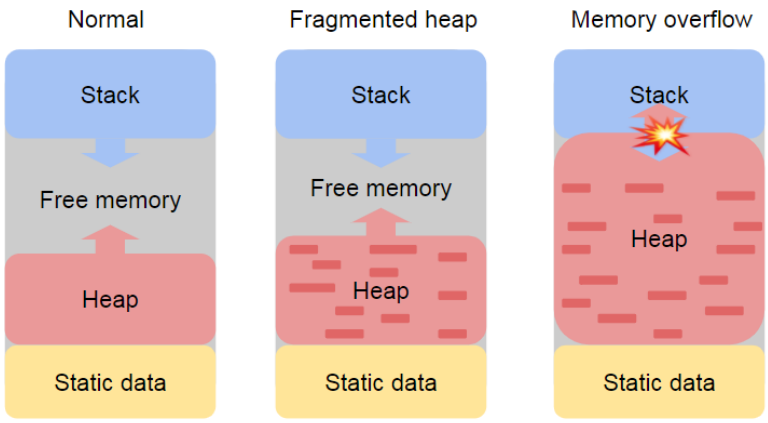


Figure 4.1: Heap usage causing memory overflow.

The SRAM usage is more problematic. As only 2048 bytes are available on the ATmega328P, all memory usage is crucial. This has some important consequences. Most notable is the unfeasibility of using the heap. In our experiments, we instantly noticed memory overflows when dynamically allocating memory. To eliminate memory fragmentation, as shown in Figure 4.1, we keep all allocations on the stack. The drawback however is that sizes need to be known at compilation time. Therefore, buffers need to be allocated upfront. These buffers then have the maximum size the program will be able to process.

External	Switchable	Bytes
USB	No	844
USB	Yes	1454
SD	Yes	1549

Table 4.1: Output influence on SRAM usage. If switching between USB and SD is required, more bytes are used.

Three choices can be made by the end user that largely influence the memory usage: the output location, the maximum number of sensors to accommodate and the specifier setup per sensor. If the SD card is used as output, the memory usage jumps enormously due to its data buffer. The entire SD library and buffer require an extra 705 bytes in comparison to using the USB connection. An overview is given in Table 4.1. The influence of the number of sensors and specifiers on memory usage is shown in Figure 4.2. Here, the SD module is included. As our multiplexers are limited to 8 addresses, no more sensors should be considered. The importance of memory footprint reduction is clear, the framework

runs at the limits of memory availability. We note that the possible setups are more than sufficient for practical usage.

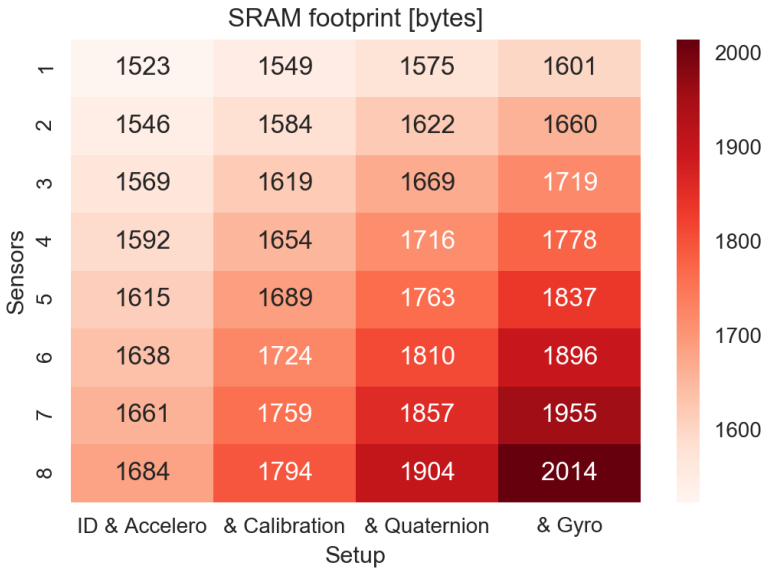


Figure 4.2: Memory usage for different setups using SD.

Finally, it is worth noting that in earlier iterations of the software framework, we did run into memory limitations. This was due to the fact that all string literals were stored in RAM. However, the Arduino environment provides a function called the $F()$ macro, which makes it very easy for application developers to store these string literals in flash memory [8]. Using this macro for all string literals freed up much memory and resolved the issues we had.

4.3 Readout speed

To test the read out speeds, we started trying different setups and I2C speeds for 1 sensor attached directly to the Arduino. An overview can be found in Table 4.2. For a setup which includes quaternion and calibration data, we conclude that speeds between 349 Hz and 675 Hz are obtainable. These speeds are inversely proportional to the number of sensors attached to the system. Therefore frequencies between 43.6 Hz and 84.4 Hz would be achievable for 8 sensors attached directly to the Arduino.

We can segment the consumed time in 4 parts: selecting (multiplexing), reading, writing

and software overhead. We clearly see that reading out the sensors consumes most of the time. We can further break down the read out time as the product of the number of requests and the bytes needed per request divided by the I2C speed. Indeed if we request more data or data which is lengthier, the speed drops. If we increase the I2C speed the overall speed increases too. Therefore the readout speed is a function of the chosen setup and I2C speed. How can we optimise the achievable speed for a realistic setting?

Operation	**Without** write		**With** write	
	100 kHz	**400 kHz**	**100 kHz**	**400 kHz**
Select & deselect	2096 Hz	4864 Hz	-	-
Sensor ID	2092 Hz	4798 Hz	1991 Hz	4305 Hz
Calibration	882 Hz	1764 Hz	861 Hz	1667 Hz
Quaternion	467 Hz	982 Hz	451 Hz	854 Hz
ID, Cali., Quat. & Timestamp	364 Hz	808 Hz	349 Hz	675 Hz

Table 4.2: Readout speeds for 1 sensor

When taking a closer look at the maximum I2C speed and driving up the frequency to find the fastest reliable transmission speed. We notice that the maximum achievable speed is a function of the wire length. For a 0.7m wire, the maximum is 160 kHz. In Figure 4.3, this situation is plotted. The SCL and SDA lines are shown in yellow and blue.

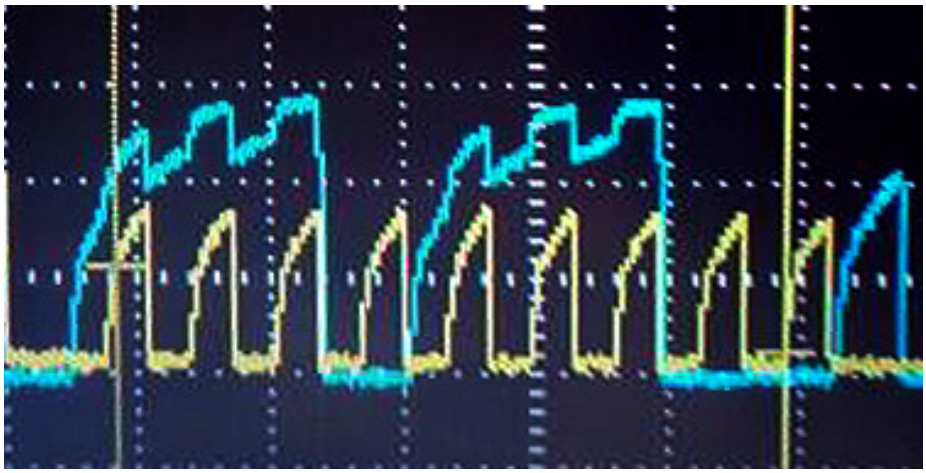


Figure 4.3: Maximum I2C speed on the wire.

Overall, we find that the time used by the program is dominated by the reading of the sensors. Multiplexing too, takes up a lot of time. An overview is given in Figure 4.4.

We note that opting for SPI instead of I2C eliminates the multiplexing factor at the cost of an extra wire per sensor. This would reduce the time per readout by 16.2% and thus increase the frequency by 19.3%. If hardware changes could be made to allow higher I2C speeds, a 10% decrease in I2C readout times would result in a 7.9% drop in total time per measurement, which would lead to a frequency increase of 11.1%.

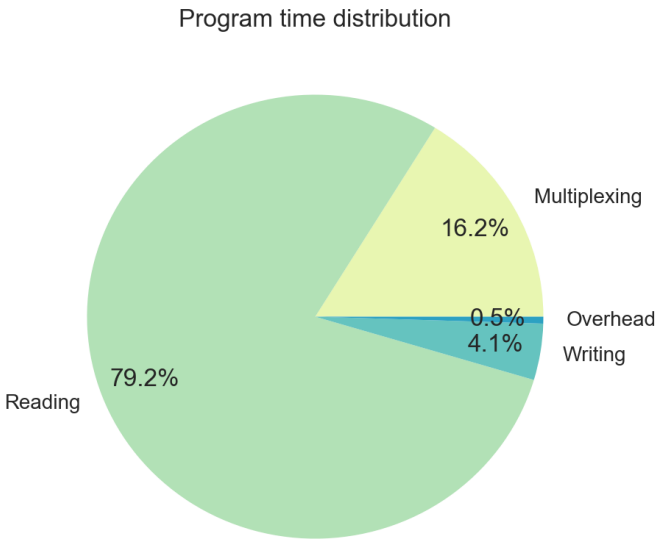


Figure 4.4: The time used by the program is dominated by reading out the sensors. Multiplexing also comes at a large overhead.

4.4 Flexibility and Plug 'n play

Using the concept of specifiers described in section 2.2.2, different types of data can easily be turned on and off. Moreover, new types of data can be added in a modular way by creating a new type of specifier. By using the sync & scan principle explained in section 2.2.1, IMUs can be added or removed without the need of reprogramming. When connecting multiple IMUs, the ID specifier identifies to which IMU the output belongs.

Finally, it should be noted that the current suit has been developed with the BNO055 in mind. Since the framework is written with modularity in mind however, it suffices to

only rewrite the `Sensor` class using MPU9250 specific functions. If this class is written to provide the same interface as the original BNO055 `Sensor` class, the framework will work with this new type of sensor as well.

Part II

Pedestrian dead reckoning navigation

In this part, a technique to estimate the position of a moving person is explored. Rather than using an absolute measurement system like GPS, relative data such as acceleration is captured to determine positions. This is the principal theme of dead reckoning navigation. The data will be captured by using an inertial measurement unit as extensively described in the previous part. The goal is to find out how precise a relative estimation can be in different scenarios.

Sections 1 and 2 provide an introduction to dead reckoning navigation and related work using both traditional approaches as well as more modern techniques. Section 3 investigates how data can be collected to use as ground truth. Section 4 describes the complete process of orientation estimation while section 5 examines a machine learning technique to predict displacement. Thereafter, the results of the orientation and displacement estimation are combined and discussed in section 6. Finally, a conclusion is made and possible future work is enumerated.

1 Introduction

In navigation, dead reckoning is the process of calculating one's current position by using a previously determined position and advancing that position based upon known or estimated speeds over elapsed time and course.

In theory, a position can be calculated by using a previous position in combination with acceleration data. This is done by integrating the acceleration to velocity and integrating the resulting velocity to displacement. In practice, the quadratic growth of errors caused by sensor drift during double integration makes results unfeasible to use.

Figure 1.1 shows the error throughout time over an interval of one second. A mean acceleration error of 0.001m/s^2 causes an increasing error in velocity up to 0.02m/s due to integration. Results even get worse after the second integration. The error in velocity now just seems flat compared to the error in displacement as one second of acceleration data causes a displacement error of over 8m.

Fortunately, most types of human movement as walking and running include repeated recognisable periods. These periodic trends can be incorporated in order to drastically decrease the error rate, something which is extensively exploited throughout the upcoming sections.

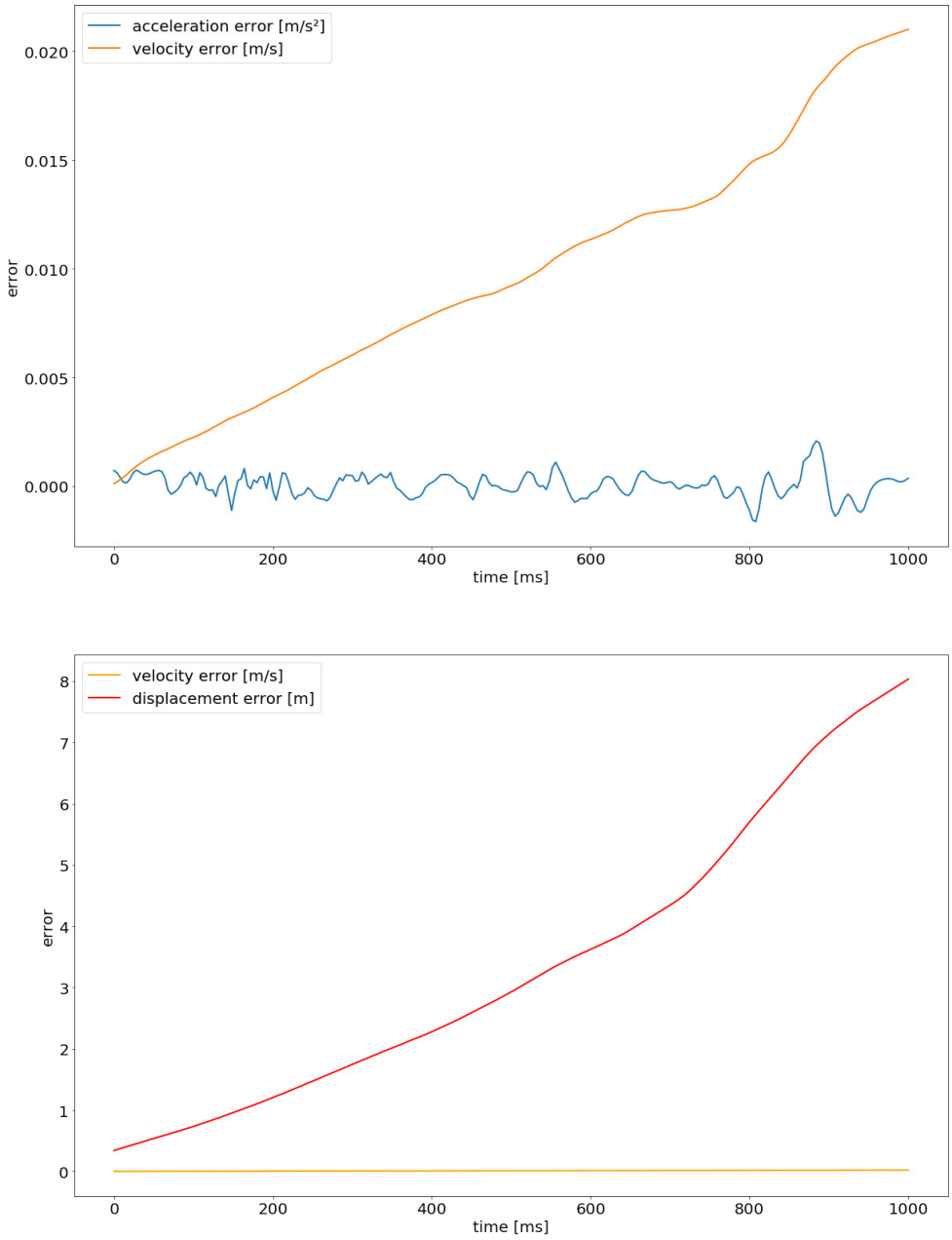


Figure 1.1: (a) Small acceleration errors due to noise (blue) cause an increasing velocity error after integration (orange). (b) The same velocity error (orange) compared to the displacement error after integration (red).

2 Related work

A lot of research has been done concerning pedestrian dead reckoning applications based on acceleration data. As described above, a first challenge is avoiding excessive error accumulation (drift) due to the integration of acceleration. This can not only be done by taking advantage of the periodic nature of walking but also by preprocessing the acceleration and double integrating a drift corrected version of the acceleration. Zhou et al. describe such a preprocessing which successfully improves the results to estimate inclinations of the elbow joint with respect to the wrist [11].

In most of the cases a step detection algorithm is implemented. Pan & Lin use a smartphone to collect acceleration data and detect steps [21]. These steps can further be distinguished in order to adapt the length of each step based on the acceleration input. Shin et al. investigate this method by using a low-cost IMU [22]. Displacements are thus rather based on the detected steps than the double integrated acceleration, although some older research tries to combine the two principles of double integration and step detection [23, 16].

Typically, a Kalman filter is applied to the acceleration data in order to remove noise and improve step detection [13, 15], although it is also used in other dead reckoning use cases such as a driving robot [14]. Nabil et al. prove that a higher accuracy regarding calculated distance can be achieved by a novel Kalman filter algorithm which also uses gyroscope and magnetometer data next to acceleration [12].

Next to the traditional approaches which determine a specific algorithm to transform acceleration and step data into displacement directly, some more modern and varying techniques using machine learning have been proposed. Diamant & Jin propose a clustering technique after transforming the data by principal component analysis (PCA) [17]. Kim et al. focus on improving the accuracy of step lengths by using a neural network [20, 19].

Besides the way of calculating results, another important specification is the position of the IMU on the body. Beauregard proposes a wearable solution using a helmet [18] whereas

most of the time an IMU is placed on the foot in order to better detect steps [16, 27, 29].

Lastly, Brajdic et al. come up with another perspective. They use the frequency domain as an alternative to exploit the repeating patterns in the components of acceleration and angular velocity by detecting peaks in the frequency components [24], [25].

3 Experiment setup

In order to (1) measure the accuracy of the upcoming results, (2) use supervised machine learning techniques and (3) combine results with absolute positioning into a hybrid system, ground truth data is necessary. The goal is to collect this data together with the data from the IMUs and align the datasets based on time.

As a start, outdoor pedestrian navigation is investigated. GPS is here still the most popular solution since it provides absolute positioning which is not prone to increasing errors due to drift. The goal is therefore not to replace this standard but rather to combine the relative IMU data with the absolute GPS positioning in order to augment the result and to reduce negative sides of a GPS module, mainly being power consumption.

3.1 Smartphone GPS vs dedicated module

The most difficult part concerning the ground truth is the accuracy of the ground truth itself. GPS accuracy typically lies around a few meters, although more accurate equipment exist. A few meters can be sufficient when driving a car but will mostly be insufficient for pedestrian navigation. Therefore, a dedicated GPS module (chip) is examined next to the more simple collection of data through a smartphone.

Figure 3.1 shows the dedicated GPS module connected to an Arduino. Data is collected by uploading a program to the Arduino which will poll the GPS module for location updates and save the output. Table 3.1 shows the output format.

time	**fix**	**quality**	**lat**	**lon**	**satellites**
2018-5-9 14:38:10	1	1	5100.736400N	342.631200E	7

Table 3.1: Data format of the dedicated GPS module.

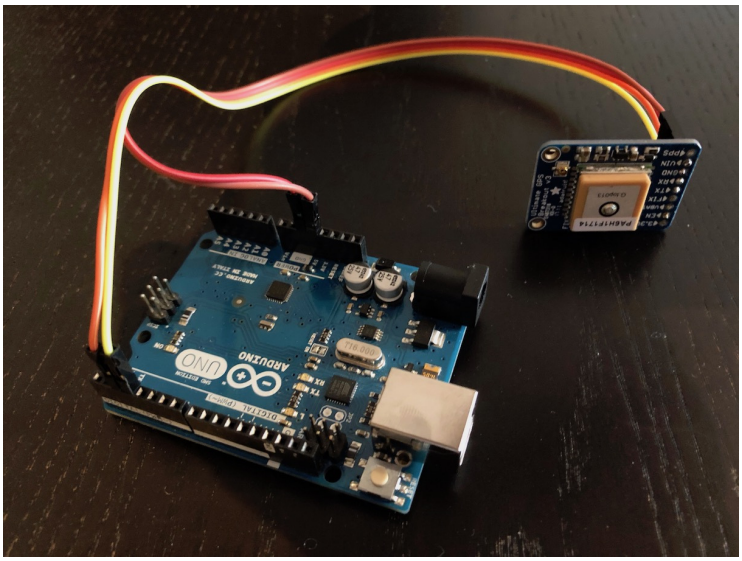


Figure 3.1: The Adafruit Ultimate GPS v3 connected to an Arduino.

As can be seen, the current time is received through the module too. Next to this, an advantage over the smartphone module is the ability to check the quality of the location based on the fix and *quality* columns.

Figure 3.2 compares the dedicated GPS module with the smartphone GPS module while walking in the Tech Lane Ghent Science Park. The data collected by the dedicated module seems more stable as there are more parts with straight sections and less wobbly ones.

Figure 3.3 shows the same path but now including a manual drawing of the real walked path. The start is located at the bottom of the parking lot on the right. The street going up (to the north) was followed in a straight line until the crosswalk was reached. Thereafter, the long bending sidewalk was taken, going straight to the parking lot on the left and finally walking to the door of the building. When taking the mean distance between samples and the real walked path as a metric, the smartphone data would be said to perform slightly better, mostly because of the bad region at the top left marked in red where the dedicated module lost its quality. Nevertheless, the input labels used for a learning algorithm will rather be relative displacements than absolute location samples, simply because the absolute region of the walked path should not influence the result. After taking the difference between subsequent location samples, the start position doesn't matter anymore. This is why the dedicated GPS module is still preferred above the smartphone module for the use case.

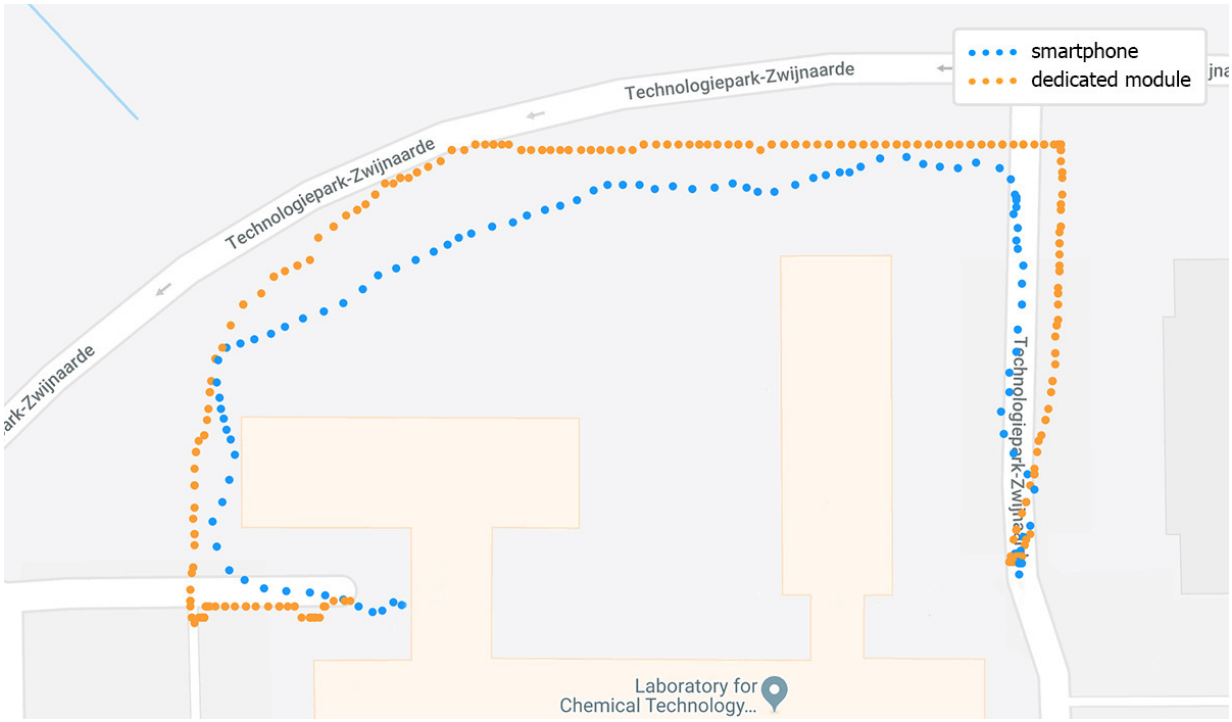


Figure 3.2: Comparison of the Adafruit Ultimate GPS v3 (orange) and the iPhone X GPS module (blue).

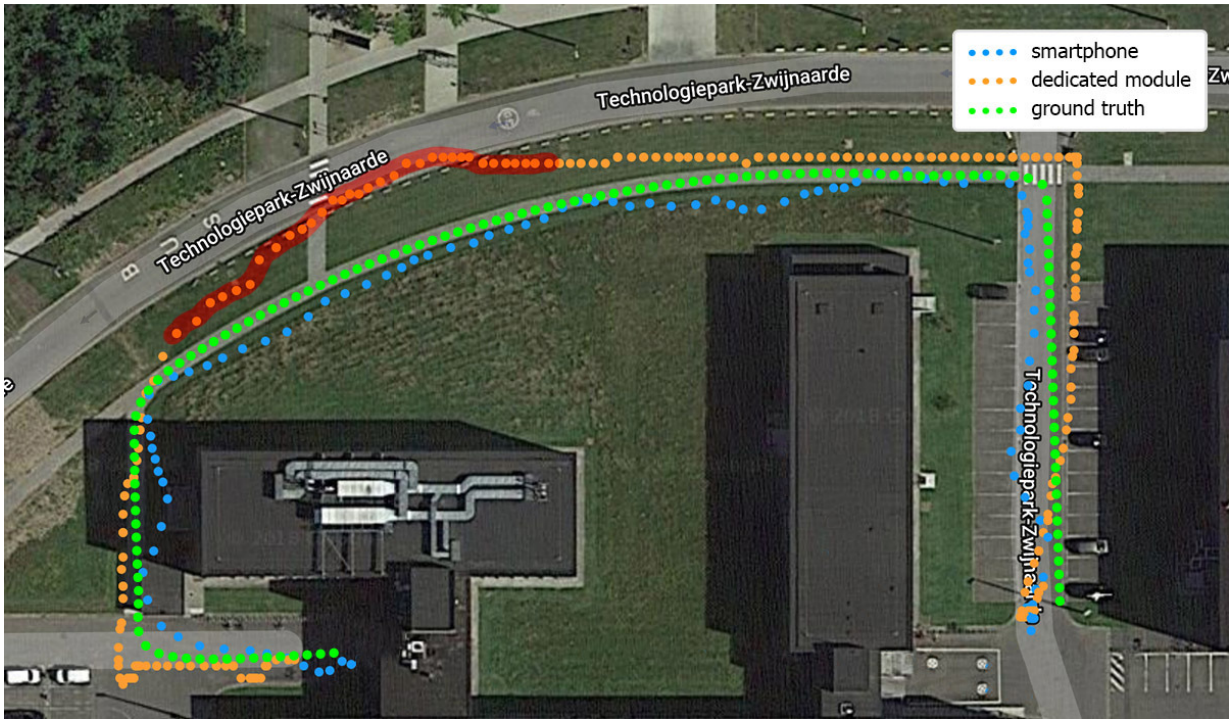


Figure 3.3: Comparison of the Adafruit Ultimate GPS v3 (orange), the iPhone X GPS module (blue) and the real walked path (green).

3.2 Resampling

The GPS and IMU data are collected at the same time but separately. This means both the GPS module and the IMU are connected to a dedicated Arduino running the framework discussed earlier in section 2.2 of part I. Therefore, data needs to be synchronised in order to get meaningful results. The Arduino connected to the IMU can provide data at a rate of 200Hz while the GPS module itself has a 10Hz update rate. For this reason, the GPS data is resampled to match the sample count of the IMU data. This is done by equally spreading the GPS data over the IMU sample range and interpolating the result to fill the gaps.

4 Orientation estimation

IMUs typically provide absolute orientations via on-board computations. However, it is needed to evaluate the precision of such derived data, since for pedestrian navigation the error margin is very low.

A good estimation for pedestrian navigation can be split up in two parts: (1) finding the correct angle describing the walking direction and (2) estimating the displacement which can differ based on the walking speed. In this section, the calculation of the angles is investigated.

4.1 Coordinate system

The BNO055 provides absolute orientations by returning quaternions. Quaternions can both be seen as rotations and orientations. In the latter case, quaternions should be interpreted as rotations from a reference orientation.

Figure 4.1 shows the default coordinate system of the BNO055. Regarding absolute orientations, the reference orientation appears to be the one where the Y-axis aligns with the direction of the North Pole and the Z-axis aligns with the inverse of gravity. This means that when aligning the IMU as demonstrated in the figure, the quaternion output will correspond to a 0° rotation, meaning the orientation equals the reference orientation. This coordinate system will further be referred to as the global coordinate system.

The coordinate system is important because of the way orientations will be used in the use case. First of all, changes in height will be neglected and everything will be projected onto the plane representing the ground. Looking top-down at a map, rotations around the X and Y axis can be ignored, leaving interest in solely the Z-axis practically (cfr. figure 4.1).

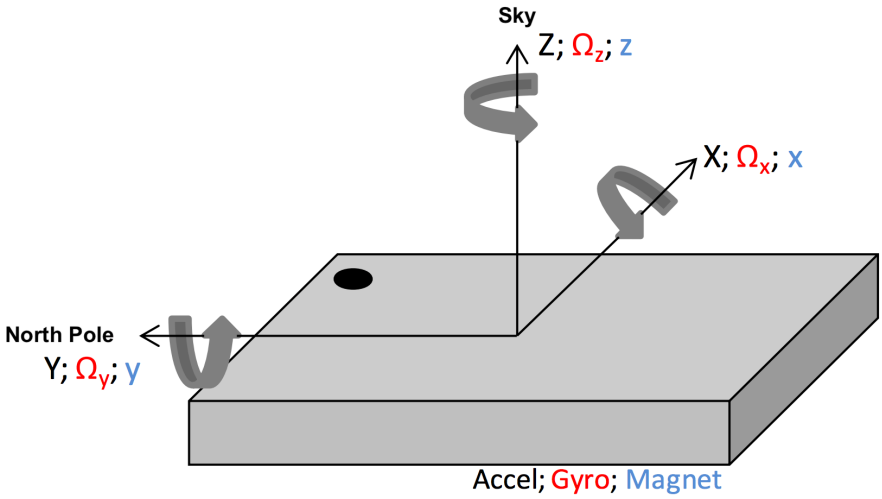


Figure 4.1: The default coordinate system of the BNO055.

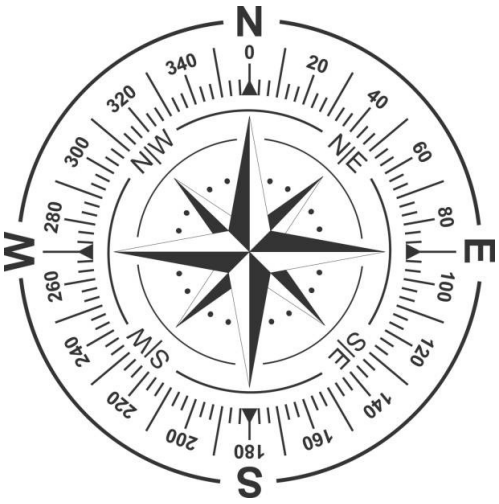


Figure 4.2: Corresponding degrees for wind directions.

To simplify reasoning, the "orientation" of a human will be defined as the angle in the horizontal plane, the ground, with respect to the North Pole and increasing in clockwise direction when viewed top-down. Looking north thus results in a value of 0°, looking east a value of 90° and so on. Figure 4.2 clarifies the situation visually.

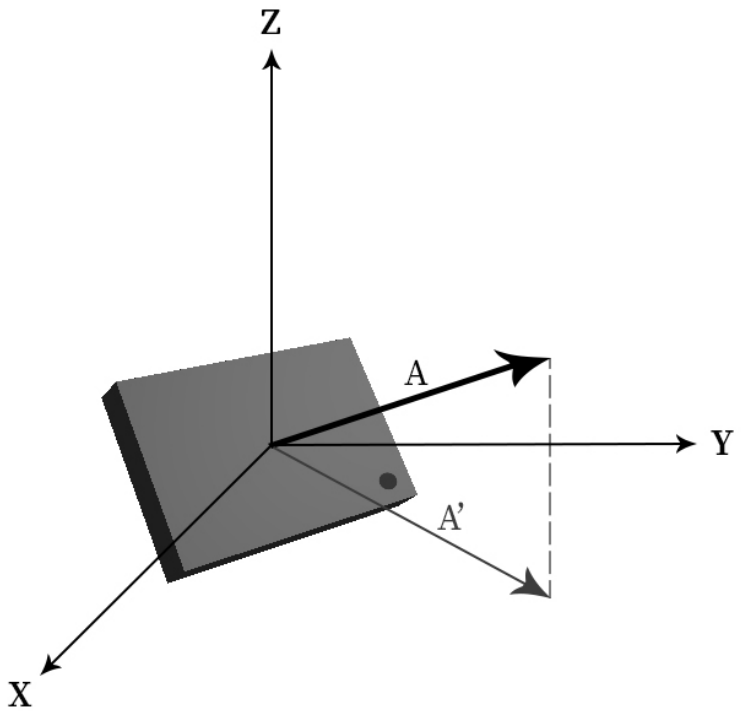


Figure 4.3: 3D visualisation of the IMU within the global coordinate system.

4.2 Calculation of angles

In physical terms, it depends on how the IMU is attached to the human body to correctly calculate the current angle. Figure 4.3 shows an example of how the IMU could be oriented in the world coordinate system. Following the earlier described properties of the coordinate system, the IMU would be pointing at the north-east in the figure. The black dot in the corner shows that the large visible face describes the top of the IMU (cfr. figure 4.1). This will be a typical orientation when the IMU is attached to a body part like the hip or the front of a leg and can be seen by imagining the body part behind the IMU in figure 4.3. When the IMU is attached well to the front of the body, vector A points out of the body and approximates the walking direction. In the following calculations, this condition of attaching the IMU to the front of the body is assumed to be satisfied.

Vector A' is the projection of A in the XY-plane, removing the height component. Next to

simplifying calculations, the projection also removes unwanted movements due to stepping. Section 5.2.3 will go deeper into step detection which obviously works best when attaching the IMU somewhere on the leg because step movements are better captured there. Step movements are mainly rotations around an axis in the XY-plane, e.g. bending a knee lets vector A point more upwarts to the sky but will not change the projection A' much. By projecting A into the XY-plane, the influence of taking steps is thus minimalised.

As long as the IMU is attached like described in the previous paragraph, vector A is represented by the Z-axis if the orientation equals the base orientation as shown in figure 4.1. When the sensor is rotated, the resulting vector A is calculated by

$$A = Q * \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.1}$$

with Q being the matrix representation of the quaternion. The projection A' can then be determined by setting the Z-component to zero.

$$A' = \begin{bmatrix} A_x \\ A_y \\ 0 \end{bmatrix} \tag{4.2}$$

Finally, the angle θ can be calculated by inverse trigonometry and taking into account a result of 0° must align with the Y-axis (North Pole).

$$\theta = arctan2(A'_x, A'_y) \tag{4.3}$$

Here, $arctan2(y, x)$ is a variation on the more familiar $arctan(y/x)$ without a singularity at $x = 0$.

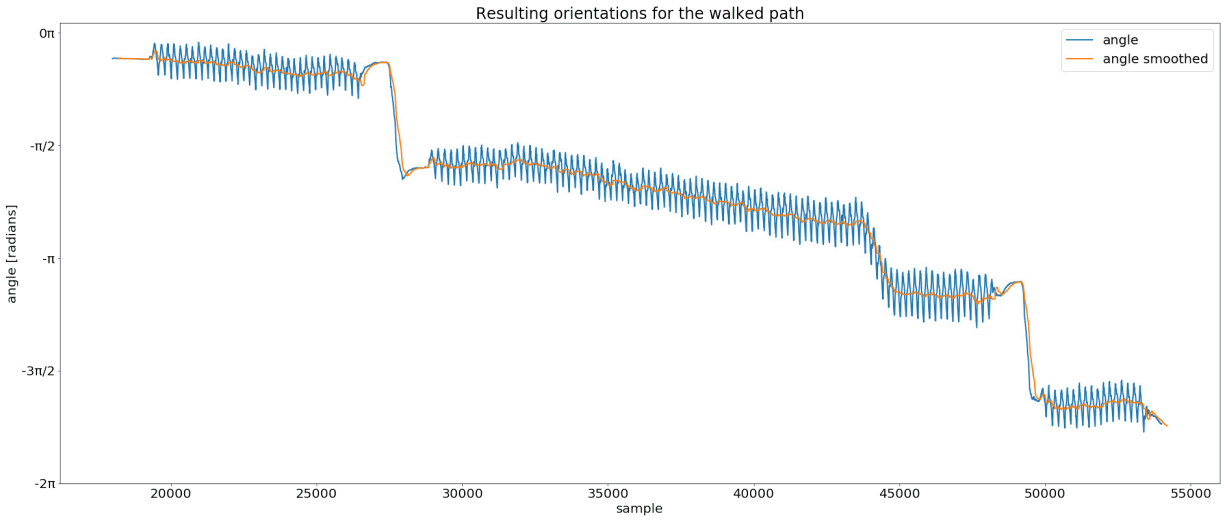


Figure 4.4: Calculated angle for the path shown in figure 3.3.

Figure 4.4 shows the calculated angle for the path walked as indicated in figure 3.3. Since steps are still visible and increasing the variance, a simple smoothing technique is applied. This is done by using a sliding window and taking the mean of the minimum and maximum value in this window.

4.3 Angle offset

As mentioned above, it is assumed the IMU is attached to the front of the body as this results in the vector A approximating the walking direction. Nevertheless, the IMU will rarely be attached in such a way that the vector A exactly aligns with the walking direction. This is proven in the bottom graph of figure 4.5. The top graph shows the coordinate offsets for every sample of the GPS data. They are calculated by subtracting the coordinates of subsequent samples. The result is used for the calculation of the angles in the bottom graph. The middle graph is a smoothed version of the top graph which takes the difference between samples lying 200 samples away from each other instead of subsequent ones. This however is only used to enhance the interpretation visually since the result in the top graph doesn't involve complications for predicting the offset despite being less clear visually.

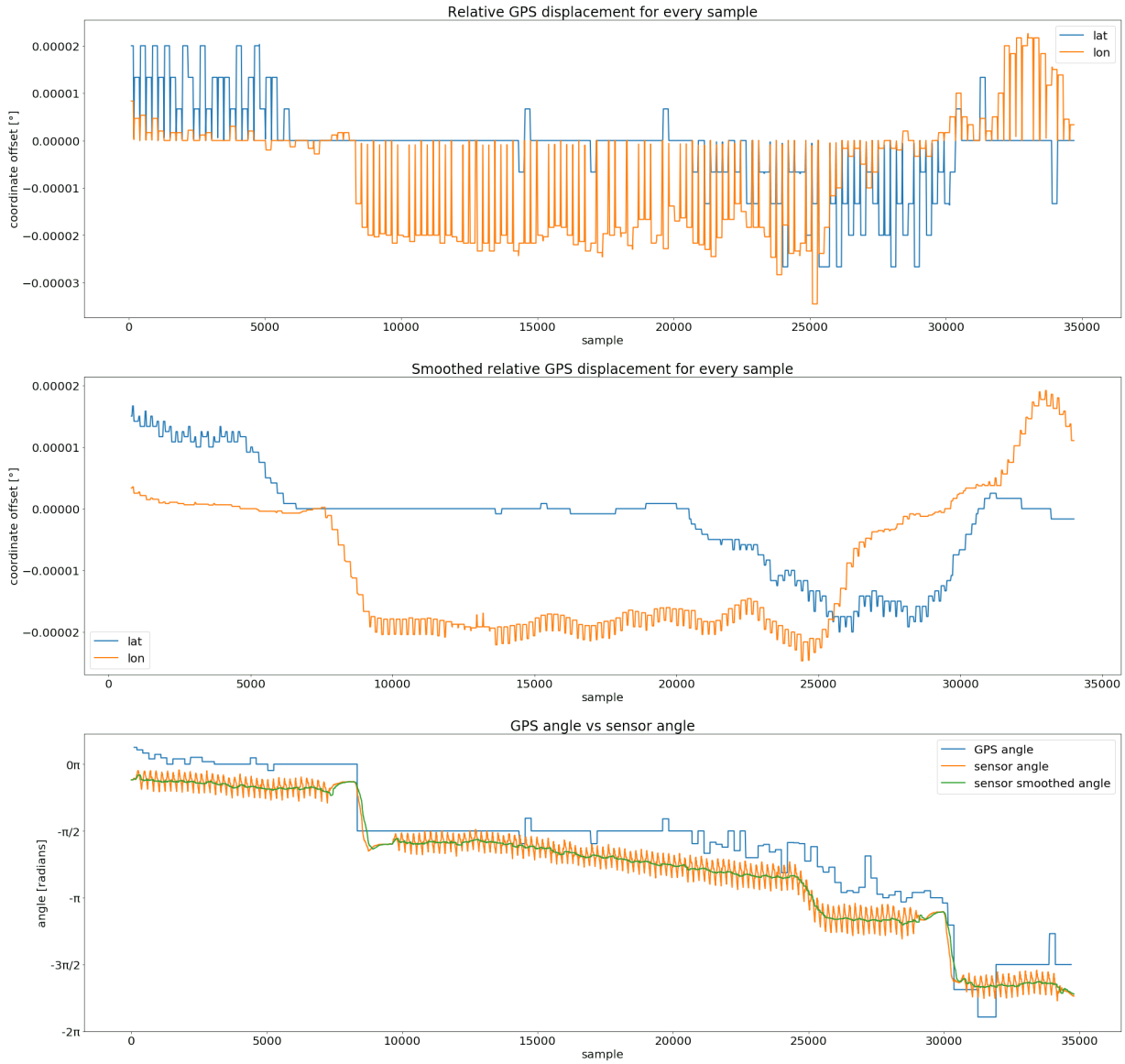


Figure 4.5: Comparison of the angles from the GPS data and the IMU data.

- **Top**: displacement of the GPS coordinates for every sample.
- **Middle**: a smoothed version reducing the variance, only used as visualisation to improve interpretation of the top graph.
- **Bottom**: GPS angle calculated from the coordinate offsets of the top graph compared to the previously calculated sensor angle of figure 4.4.

There is a clear difference between the angle from the GPS and the calculated angle from the quaternions of the previous section. The difference however seems to be constant. As long as the IMU is attached firmly without any changes in position and orientation with respect to the body part during the walk, a simple offset can be added to resolve this issue.

4.4 Prediction

When it comes to predicting results, a model must be chosen which fits the true function f best. This is the function providing the wanted result for every input and is most of the time unknown, resulting in different models being examined and compared. However in this case, f should be of the form $f(x) = x + c$ where the input x corresponds to the angle θ of equation 4.3. Since the orientation of the IMU on the leg is believed to be the only reason of the offset, the angle should be the only feature as input of the model. The model can then calculate the offset by taking the difference between the angle θ and the angle of the ground truth for every sample and taking the mean of all these differences.

The above described calculation could easily be performed by the linear regression model where the result would be of the form $\hat{f}(x) = a * x + b$ and factor a would lie close to 1 to justify the assumption of the offset. Anyhow, taking differences of angles in order to get an average should consider the nature of a radial basis. Averaging two angles with values 350° and 10° should give a result of 0° instead of 180°. Therefore, the calculation is defined in a simple but dedicated *radial baseline* model which calculates the offset as described in the previous paragraph but using modular arithmetic to overcome the above mentioned complication. Figure 4.6 demonstrates the effective result. In this case, the offset is 28° and the mean absolute error decreases from 31° to 12°. It has to be noticed that the remaining difference is not only due to inaccuracy of the IMU. The GPS data used as ground truth will not be exact too as explained in section 3. The marked region in red displays this inaccuracy of the GPS data. The calculated sensor angle however isn't affected by this inaccuracy due to the simplicity of the radial baseline model. These first observations show there is a clear linear relation between the angles of the GPS data used as ground truth and the IMU data. Figure 4.6 hereby concludes this section. The results will be used further on when combining them with the position prediction of the following section.

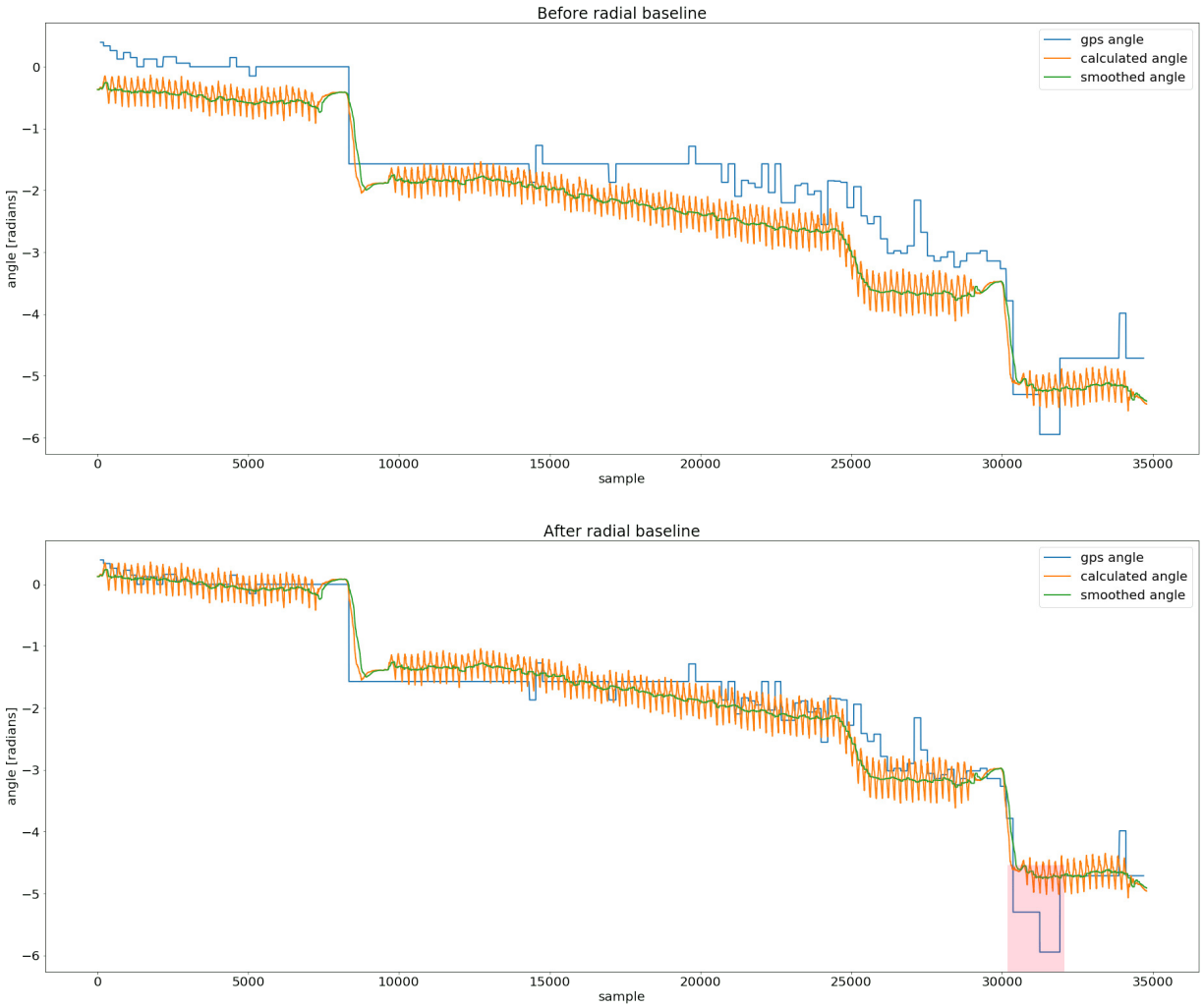


Figure 4.6: The angle calculated from the gps data (blue) compared to the angle from the IMU calculated via the method of section 4.2 (orange, green). Before applying the radial baseline function, a clear offset between the two is visible (top). After applying the function, the offset is minimised (bottom). The red region displays a less accurate part of the GPS data.

5 Displacement estimation

As already mentioned, a good estimation for pedestrian navigation can be split up in two parts. The first part, finding the correct angle describing the walking direction, was treated in the previous section. Here, the second part, estimating the displacement which can differ based on the walking speed, is investigated.

The walking speed of humans can vary in two possible ways. People can increase their speed by (1) taking larger steps in about the same time span or (2) taking about the same steps in a smaller time span. The real answer will be a combination of both which in fact is related to the definition of velocity: displacement (larger steps) divided by time (smaller time span).

5.1 Ground truth

New data was collected in order to discover differences in speed more easily. A distance of 51 meters was traversed four times. Firstly, the path was walked at a fast and steady speed, followed by walking back the same path using a regular speed. Thereafter, this was repeated starting with a significant lower speed and returning again using a regular speed. Table 5.1 provides the duration and speed details of the four walks.

Figure 5.1 plots the relative displacements based on the GPS data. The displacements are calculated by taking the difference between every sample and the one 100 samples further. Since the data is resampled as explained in section 3.2, a range of 100 samples equals about half a second which corresponds to the time of an average step. The result doesn't seem to lose important details compared to the difference between subsequent samples and in fact reduces noise and variance effectively. The four walks are marked as the four regions with a blue background in the figure while the orange and red lines indicate the mean displacement over each region. The orange lines are based on the 51 meters real displacement while the red ones use the estimated displacement of the GPS. On the one

hand, the similarity of the two regular regions is clearly visible while on the other hand, the difference with the other regions can easily be seen too.

region	duration (s)	speed (m/s)	ratio	GPS (m)
fast	21.5	2.37	1.83	46.32
regular 1	38.5	1.32	1.02	51.65
slow	57.5	0.89	0.69	47.73
regular 2	40.5	1.26	0.98	47.58

Table 5.1: Statistics for the four walked regions in figure 5.1. Ratio column: the ratio of the duration compared to the average duration of the two regular regions. GPS column: the total meters walked according to the GPS data.

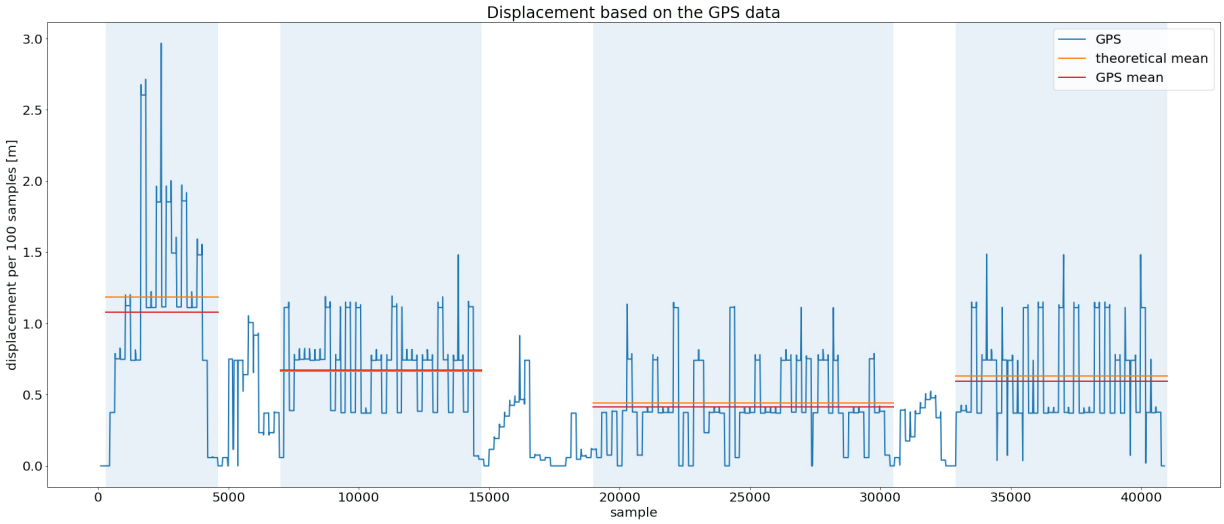


Figure 5.1: The displacement based on the GPS data between each sample and the one 200 samples further. Four regions are highlighted with turns in between.

The displacement calculated from the GPS signal seems very unstable in every region knowing a constant pace was kept while walking. This is because the GPS isn't providing the accuracy to estimate positions within a few meters. Therefore, the theoretical mean for each region will be used as the ground truth instead of the GPS signal since the total

distance (51m) and the duration of each region is known and it is assumed the walking speed was kept steady within each region.

5.2 Features

The following sections will inspect three features which are potentially correlated to the walking speed. Firstly, the acceleration. This is mathematically related to the speed of stepping but can also introduce drops in accuracy due to noise. Secondly, the angular velocity provided by the gyroscope is investigated in the same way as the acceleration. Lastly, a step detection algorithm will be used to detect individual steps. The goal here is to extract the duration of each step.

5.2.1 Acceleration

The acceleration components (x, y, z) for the four regions earlier discussed are displayed in figure 5.2. Comparing the regions, a clear distinction can be made at first sight.

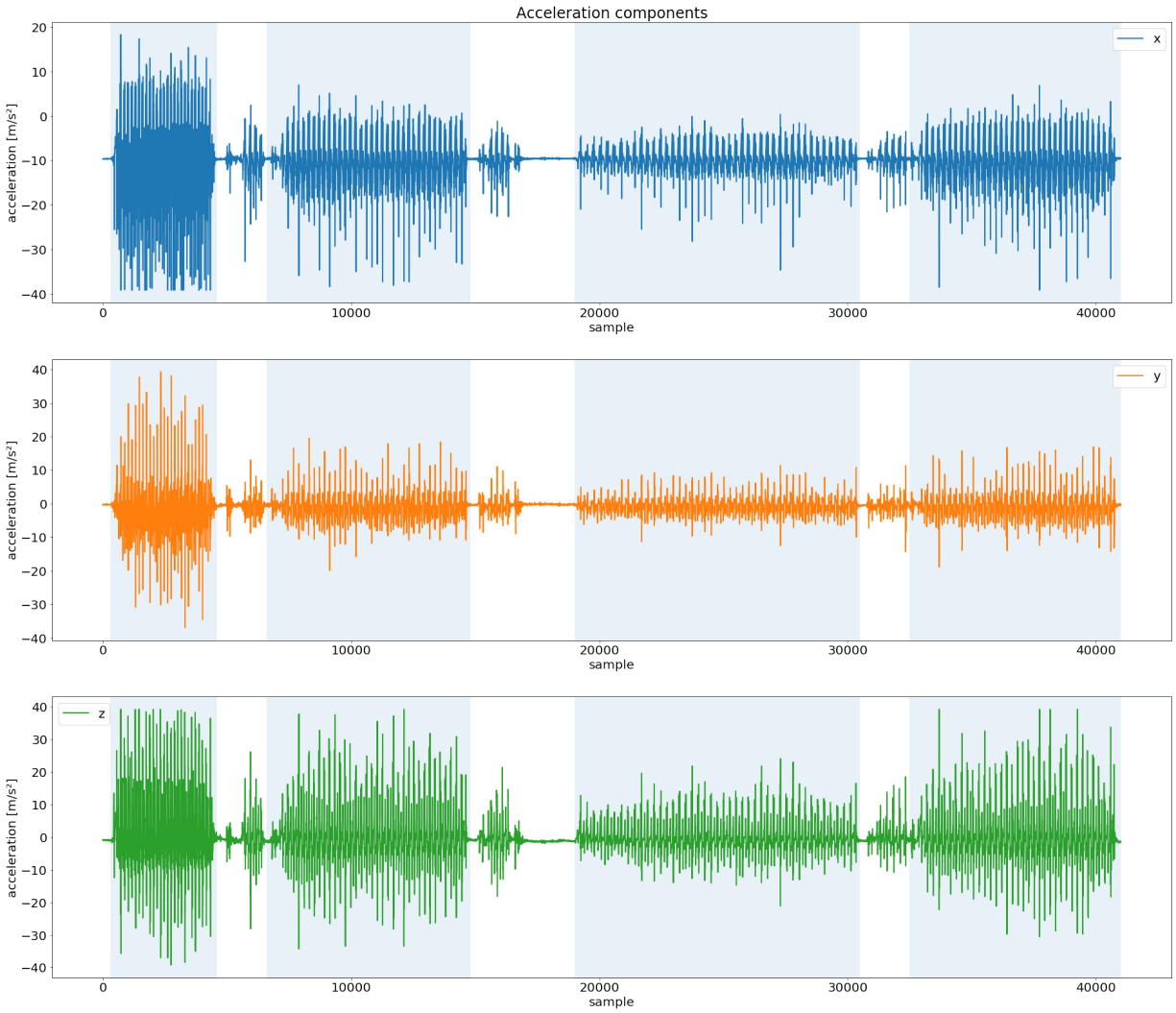


Figure 5.2: Acceleration components for the four regions defined in section 5.1.

In an attempt to generalise and increase the difference between the regions even more, the absolute value was taken after subtracting the mean of the complete sample range. This mean value can differ from zero due to gravity. Based on the result, shown in figure 5.4, three operations are performed which could lead to potential features for a learning algorithm. All operations are done by using a rolling window in order to reduce noise and lower the variance within a region. The first operation is a rolling mean, defined over a window size of two steps or thus one step cycle (left + right leg). The second operation uses the standard deviation with the same window size as the rolling mean. The last operation is a rolling maximum, defined over 1.2 times the window size of the other operations. This

bigger window size reduces the variance once again because the maxima are defined by the peaks in the acceleration, typically one peak per cycle caused by a step of the leg where the IMU is attached to. Setting the window slightly bigger than one cycle prevents the maximum from dropping right before the next peak is reached. This is shown in figure 5.3. This in fact approaches the extraction of data from individual steps without the need of a step detection algorithm. Every step can clearly be seen by the varying stairs from the rolling maximum.

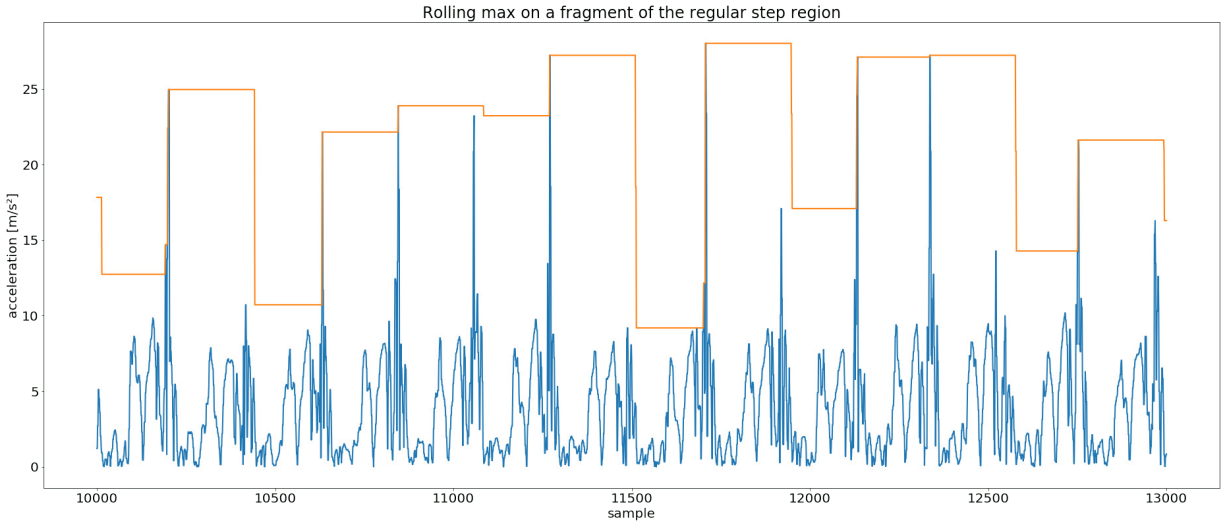


Figure 5.3: A fragment of the rolling maximum on the regular step region.

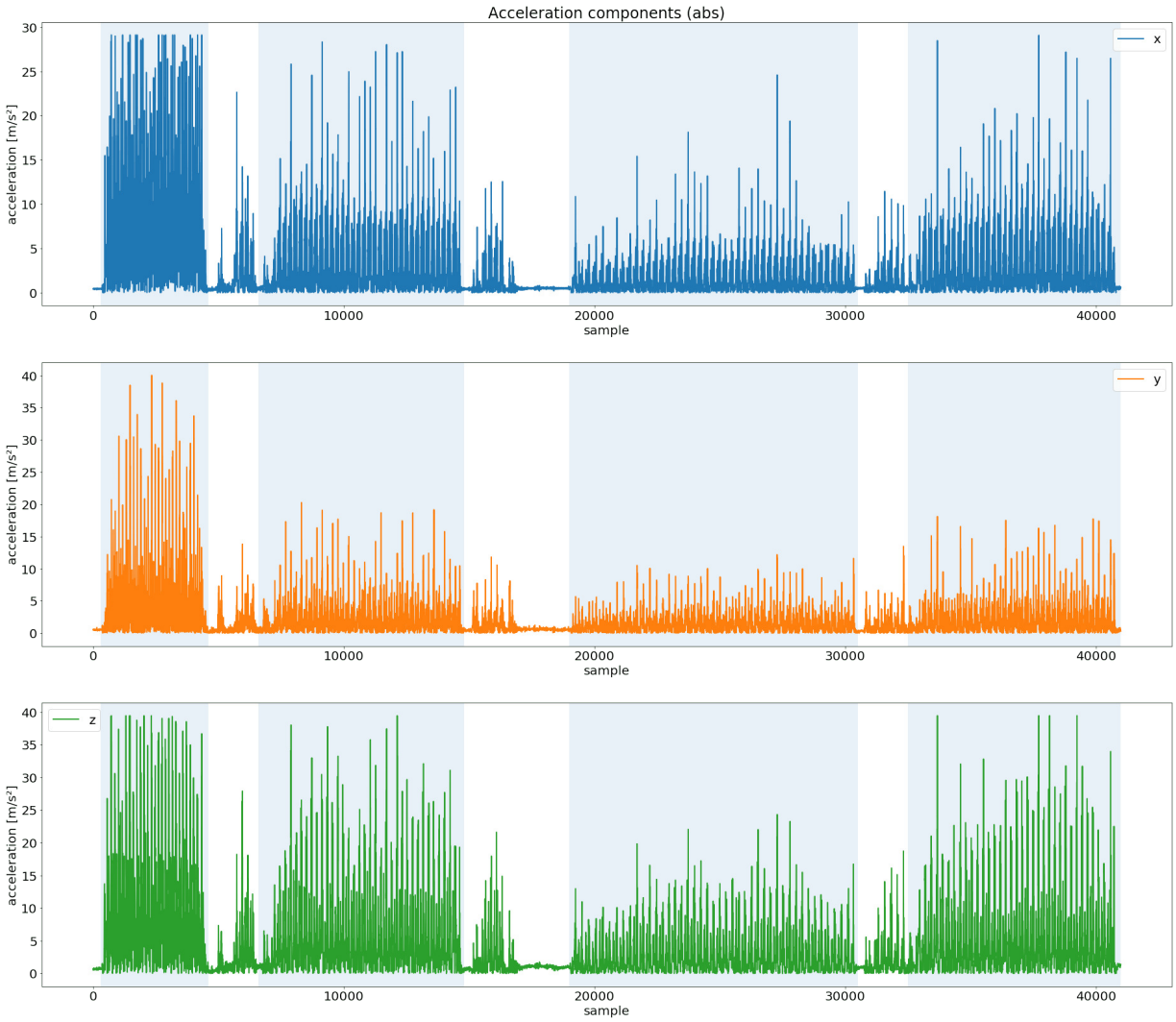


Figure 5.4: Acceleration components of figure 5.2 after subtracting the mean and taking the absolute value.

Figures 5.5, 5.6 and 5.7 show the operations for each component and every step region while tables 5.2, 5.3 and 5.4 provide the mean values corresponding to the red lines in the figures. Every mean value is also compared to the mean value of the regular regions to get a better insight in the ratio between regions. These results will further be used as input features of a learning algorithm.

bottom=0.1cm

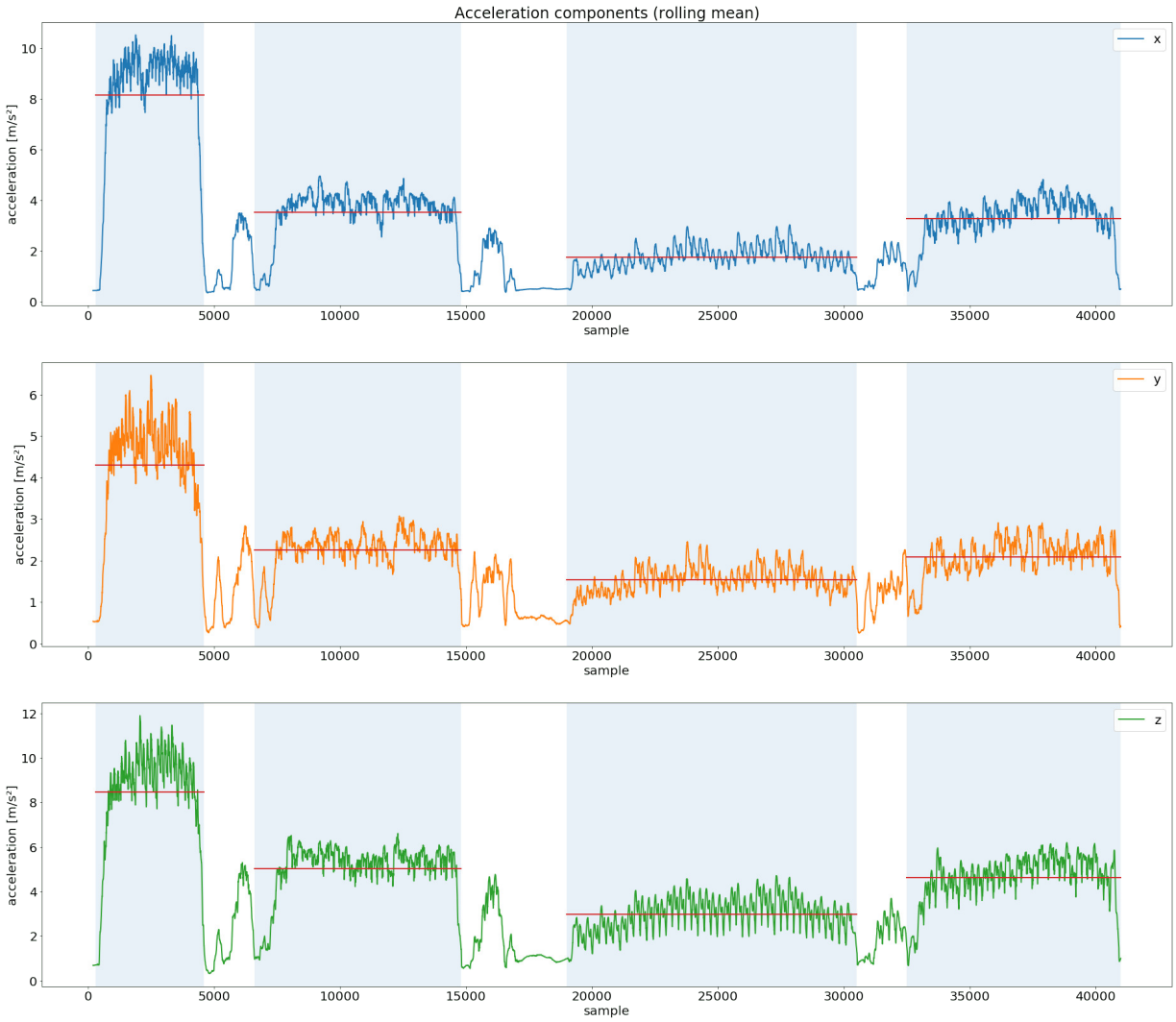


Figure 5.5: Rolling mean of the transformed acceleration components of figure 5.4.

region	mean (m/s^2)			ratio		
	x	y	z	x	y	z
fast	8.17	4.30	8.47	2.39	1.98	1.76
regular 1	3.54	2.25	5.01	1.04	1.04	1.04
slow	1.77	1.54	2.97	0.52	0.71	0.62
regular 2	3.29	2.09	4.62	0.96	0.96	0.96

Table 5.2: Statistics of figure 5.5. The ratio is the mean value divided by the mean of both regular regions.

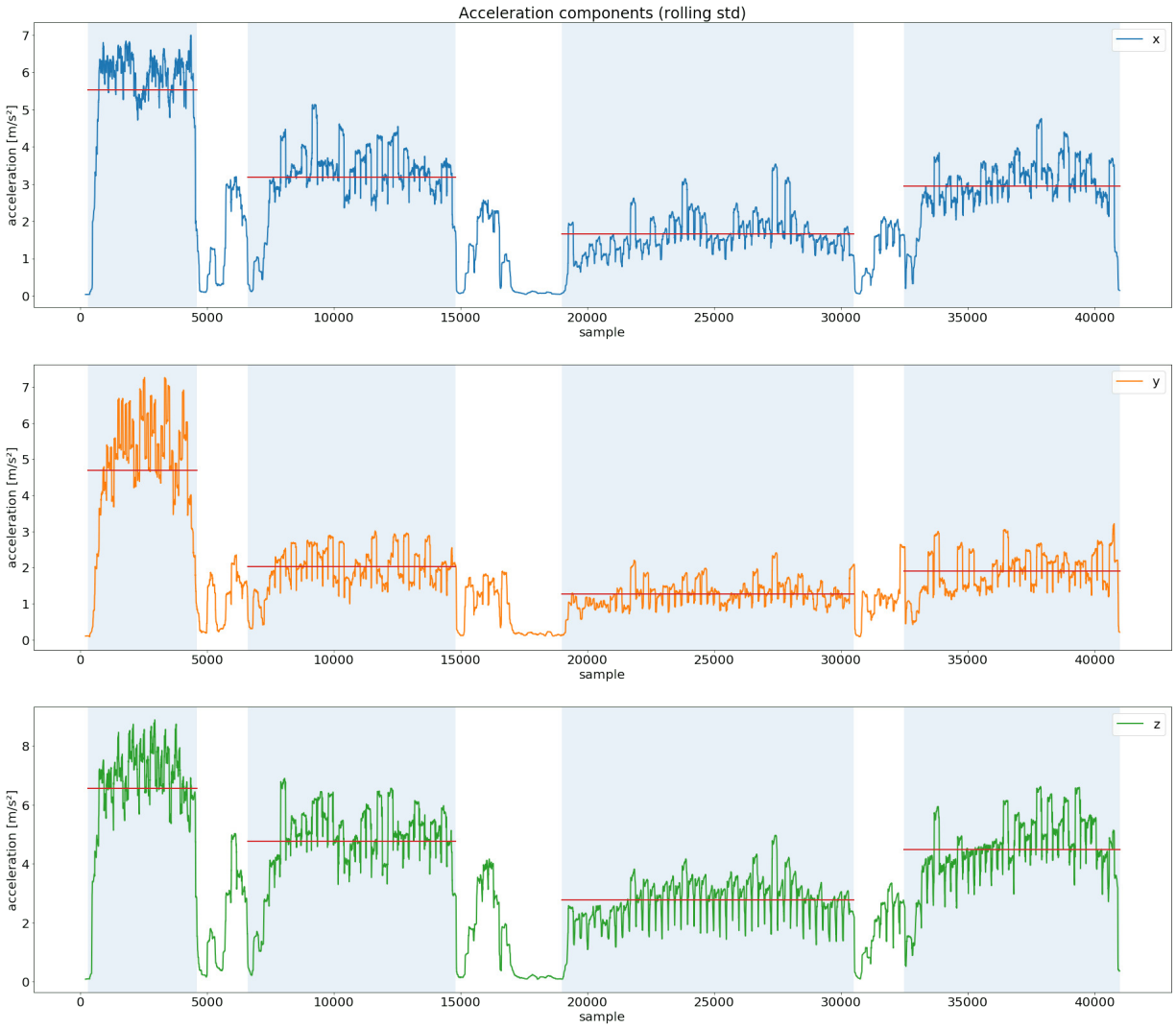


Figure 5.6: Rolling standard deviation of the transformed acceleration components of figure 5.4.

region	mean (m/s^2)			ratio		
	x	y	z	x	y	z
fast	5.53	4.68	6.56	1.81	2.39	1.42
regular 1	3.18	2.02	4.76	1.04	1.03	1.03
slow	1.66	1.27	2.78	0.54	0.65	0.60
regular 2	2.94	1.90	4.48	0.96	0.97	0.97

Table 5.3: Statistics of figure 5.6. The ratio is the mean value divided by the mean of both regular regions.

Figure 5.7: Rolling maximum of the transformed acceleration components of figure 5.4.

region	mean (m/s^2)			ratio		
	x	y	z	x	y	z
fast	26.47	27.22	35.05	1.57	2.27	1.38
regular 1	17.74	12.53	25.44	1.06	1.05	1.00
slow	9.02	7.22	13.35	0.54	0.60	0.53
regular 2	15.88	11.45	25.41	0.94	0.95	1.00

Table 5.4: Statistics of figure 5.7. The ratio is the mean value divided by the mean of both regular regions.

5.2.2 Angular velocity

The exact same operations as defined in the previous section about acceleration are repeated for the angular velocity. The results look very similar to the ones of the acceleration, albeit less prone to noise and therefore potentially more robust as input features. Figures 5.8 to 5.12 show the results, tables 5.5 to 5.7 provide the same statistics as earlier explained. Looking at these statistics, the ratios of the Y-component in the table of the rolling mean (table 5.5) resembles the ratios of the ground truth in table 5.1 almost exactly. This means that a simple scaling of this feature could perhaps result in a good prediction further on. This could be explained by the fact that the Y-axis of the IMU is closely aligned to the rotation axis of the upper leg and therefore best captures an increase or decrease in angular velocity which could approximately be linearly related to the displacement.

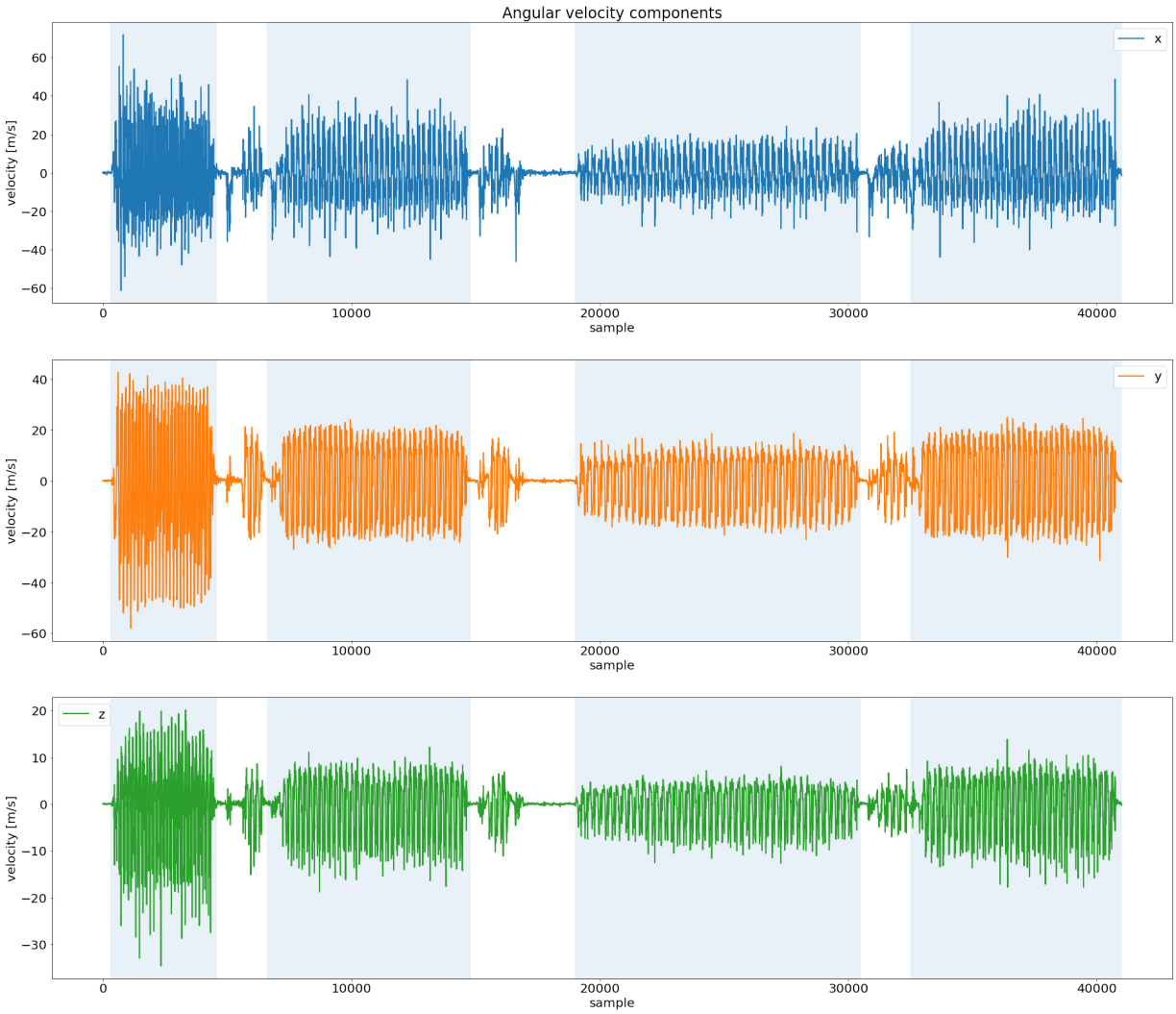


Figure 5.8: Angular velocity components for the four regions defined in section 5.1.

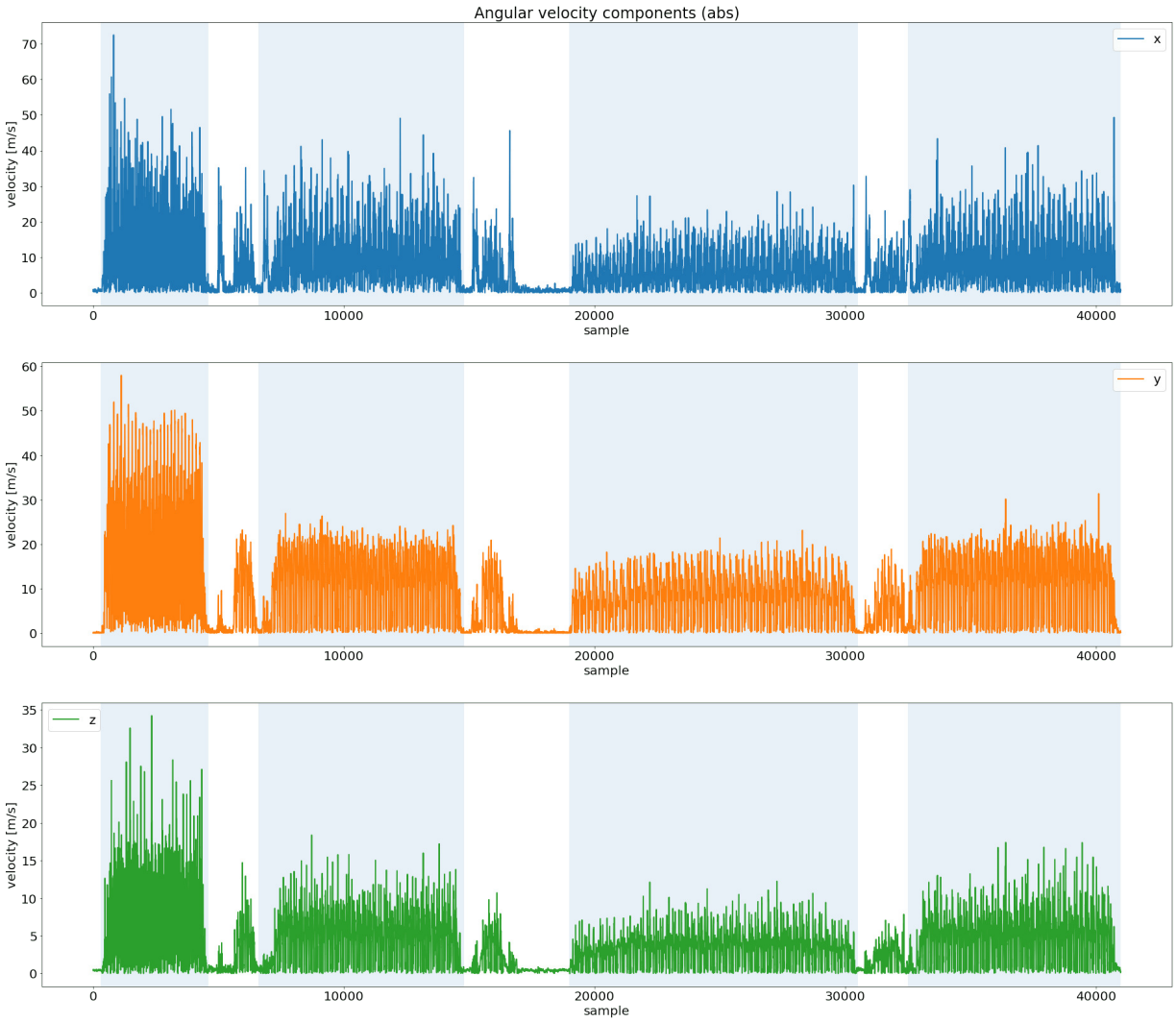


Figure 5.9: Angular velocity components of figure 5.8 after subtracting the mean and taking the absolute value.

bottom=0.1cm

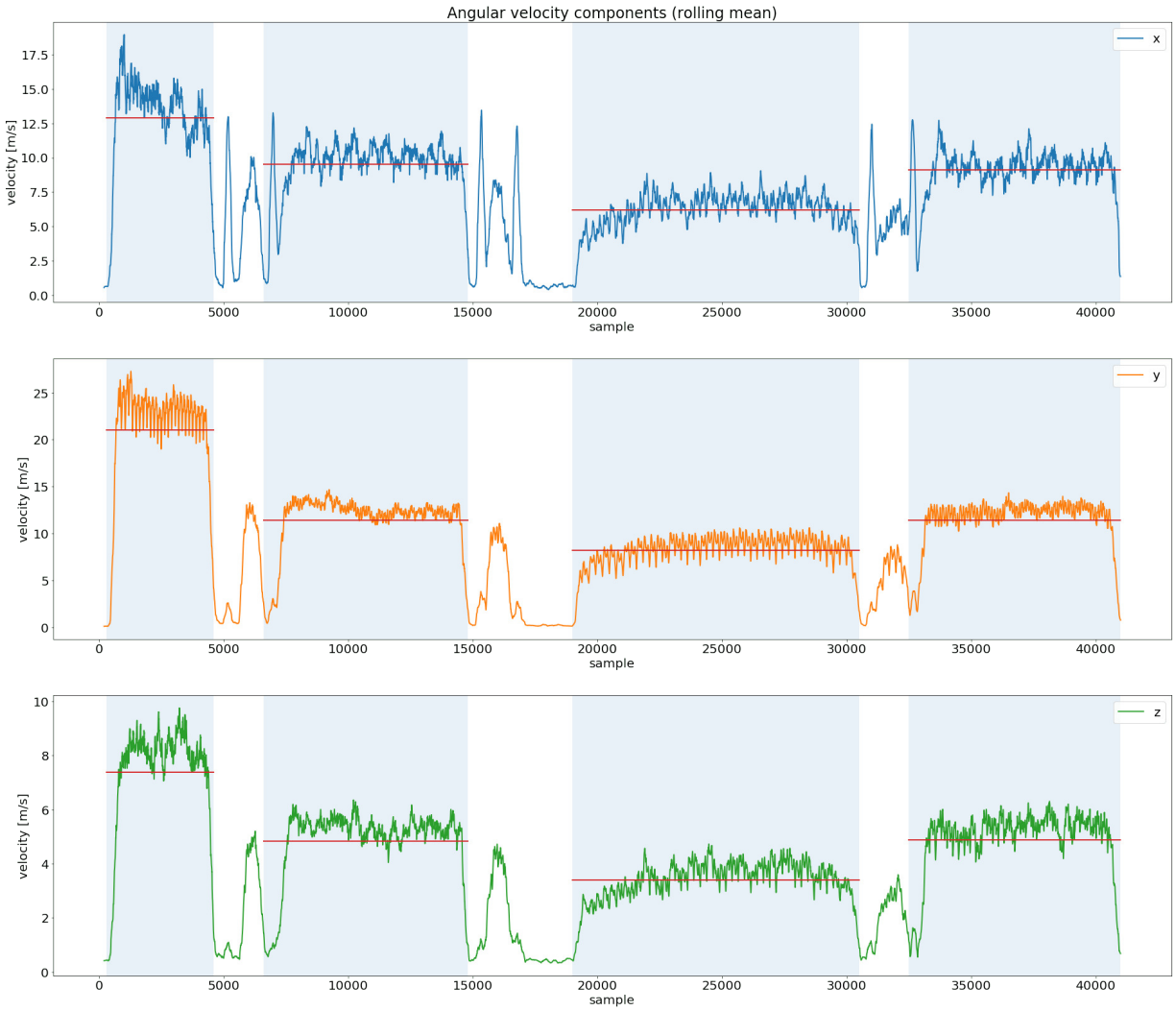


Figure 5.10: Rolling mean of the transformed angular velocity components of figure 5.9.

region	mean (m/s)			ratio		
	x	**y**	**z**	**x**	**y**	**z**
fast	12.89	21.0	7.38	1.38	1.84	1.52
regular 1	9.54	11.4	4.84	1.02	1.00	0.99
slow	6.22	8.23	3.39	0.67	0.72	0.70
regular 2	9.09	11.38	4.89	0.98	1.00	1.01

Table 5.5: Statistics of figure 5.10. The ratio is the mean value divided by the mean of both regular regions.

Figure 5.11: Rolling standard deviation of the transformed angular velocity components of figure 5.9.

region	mean (m/s)			ratio		
	x	**y**	**z**	**x**	**y**	**z**
fast	9.40	11.67	4.60	1.51	2.02	1.69
regular 1	6.38	5.78	2.66	1.02	1.00	0.98
slow	4.16	4.37	1.89	0.67	0.76	0.69
regular 2	6.10	5.79	2.78	0.98	1.00	1.02

Table 5.6: Statistics of figure 5.11. The ratio is the mean value divided by the mean of both regular regions.

Figure 5.12: Rolling maximum of the transformed angular velocity components of figure 5.9.

region	mean (m/s)			ratio		
	x	y	z	x	y	z
fast	44.26	45.98	23.34	1.42	2.11	1.87
regular 1	31.79	21.59	12.48	1.02	0.99	1.00
slow	19.15	17.39	8.44	0.62	0.80	0.68
regular 2	30.44	22.04	12.48	0.98	1.01	1.00

Table 5.7: Statistics of figure 5.12. The ratio is the mean value divided by the mean of both regular regions.

5.2.3 Step duration

For detecting steps and their duration, the IMU is usually placed on the foot or leg since lower placement is more sensitive to phases of the walking cycle [30]. One of the most popular methods to detect steps is the zero velocity update (ZUPT) algorithm based on the acceleration components [31]. Although the acceleration is by far the most used method to detect steps and typically outperforms the data from the gyroscope and magnetometer regarding step detection, it is worth investigating the quaternion components in this case. Recall that these quaternions are the result of the fusion algorithm on chip combining the best of three worlds (acceleration, angular velocity and magnetic field).

As can be seen in figure 5.13, the raw quaternion components look like already processed signals ready to use for detecting steps in contrast to the acceleration components which contain a lot more noise, albeit still having the possibility to detect steps. Therefore, a leap in the dark is taken trying to transform the quaternion data into individual steps instead of using the widely known ZUPT algorithm on the acceleration data.

The first step is the combination of the components into one signal. When working with acceleration, the signal is typically transformed by calculating the magnitude. This method cannot be used for the quaternion data since these are normalised, resulting in a constant magnitude of 1. Therefore, the sum of the quaternion components will be used. A part of both the acceleration and quaternion signal is shown in figure 5.14 which definitely encourages the choice of using quaternions over acceleration.

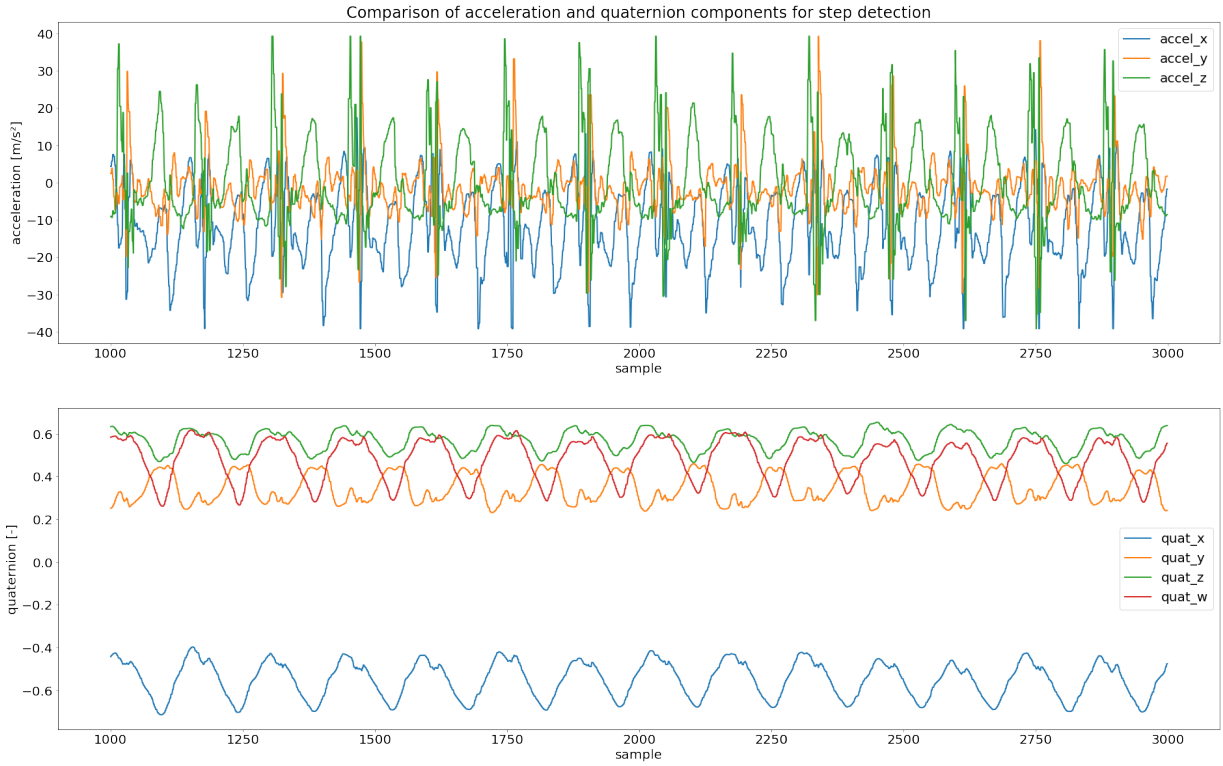


Figure 5.13: Comparison of the acceleration and quaternion components for step detection.

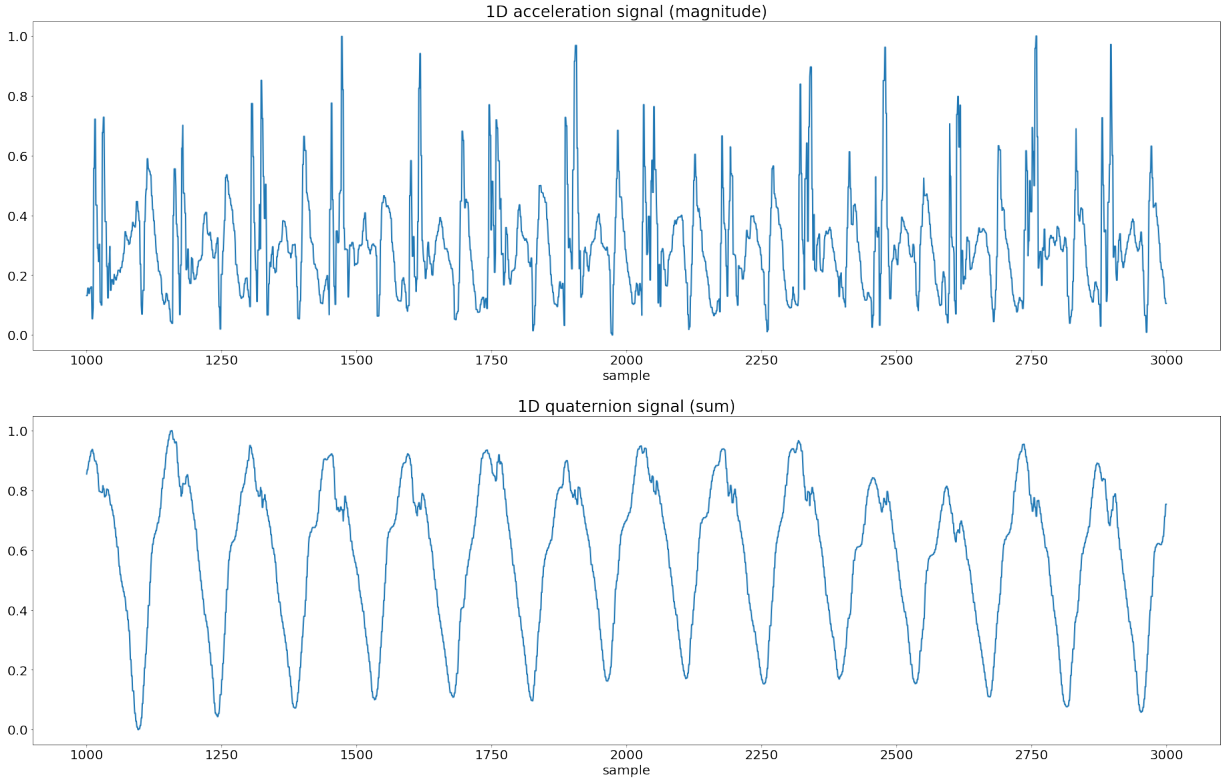


Figure 5.14: 1D acceleration and quaternion signals. The acceleration components are combined by taking the magnitude while the quaternion components are summed.

Next, the signal is centered by using a rolling window with a window size equal to a mean cycle length of 200 samples (based on the regular speed regions). The window calculates the mean value and subtracts it from the signal in order to center the signal around zero. Thereafter, a Savitzky-Golay filter is applied which smooths the signal, thereby reducing noise and bumps. Looking at figure 5.15, the filter removes the roughness in the peaks of the signal, which will be important for the next step.

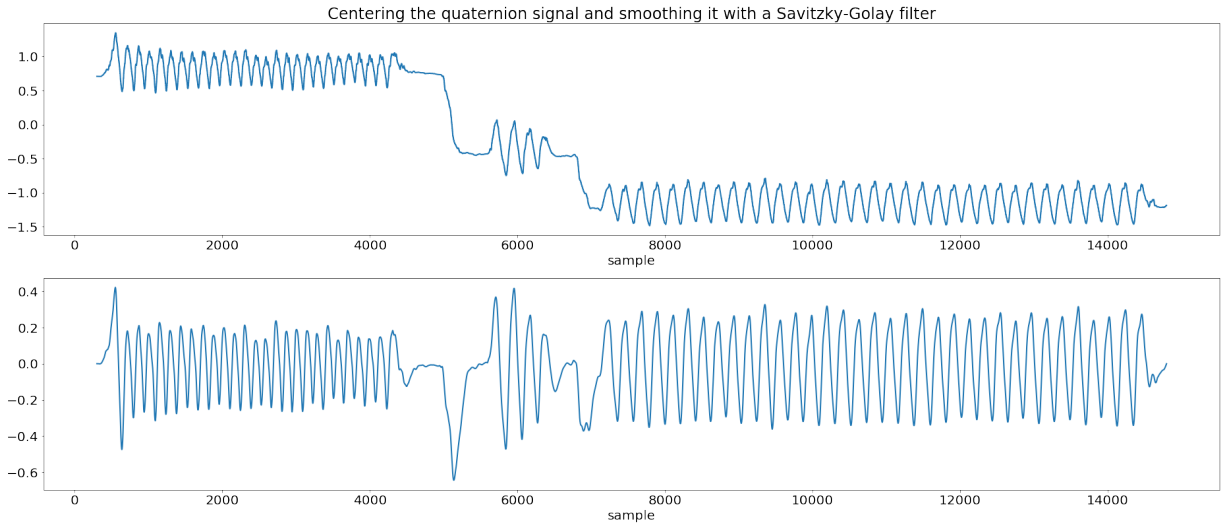


Figure 5.15: Centering the quaternion signal and applying a Savitzky-Golay smoothing filter.

A last step before extracting steps from the filtered signal is dividing it into step regions. A disadvantage of the quaternion signal compared to the acceleration in this case is the change of the signal even if people make a turn by just pivoting around their axis. To solve this, a new rolling window with the same size as before (200 samples) is applied, this time to the angular velocity. First, the minimum value of the components (x, y, z) for every sample is calculated. The philosophy here is that each component should have a significant value while taking steps. Then, the output is provided to the rolling window which calculates the standard deviation. This standard deviation then decides whether a value is significant based on the mean standard deviation of the signal. In most research, this method is applied to the acceleration instead of the angular velocity. However, the angular velocity once again proved to be more stable, resulting in less unwanted switches from non-stepping to stepping and vice versa. This was already demonstrated by figures 5.6 (acceleration) and 5.11 (angular velocity). Figure 5.16 shows the detected step regions. The sample range is the same as used before, containing the four walked regions (fast, regular, slow, regular). These regions are nicely detected together with some extra parts in between. These parts indeed contain some steps made by walking from the left side of the street to the right or the other way around before the start of the next region. Also, notice how the down spike in figure 5.16 around sample 5000, introduced by pivoting without stepping, is filtered out by using the detected step regions.

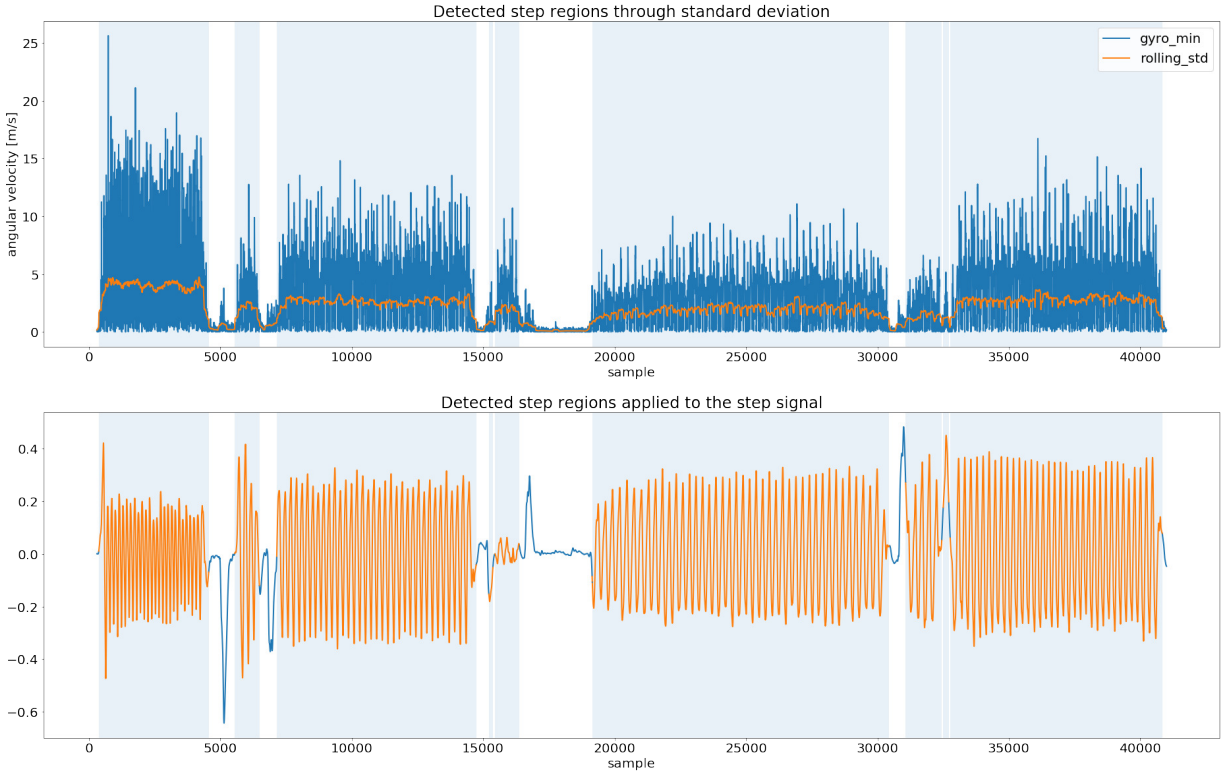


Figure 5.16: The detected step regions by using the standard deviation, having a blue background. The minimum value of the angular velocity components is calculated and the standard deviation is applied by the rolling window (top). The same regions are used on the filtered quaternion signal (bottom).

The last part of the step detection consists of peak detection. Taking the first order difference together with a threshold and minimum distance between peaks results in figure 5.17, detecting every wanted peak. Having the sample index for every peak, the duration in amount of samples between peaks can easily be calculated. Assumed here is that every peak belongs to an individual step cycle, such that the amount of samples between two consecutive peaks equals the duration of a step cycle. Figure 5.18 displays the peaks for all four regions together with the amount of samples between each peak. The amount of samples for both the regular regions beautifully lie around 200 which was already stated to be the mean cycle length of these regions. The duration of the fast and slow regions also confirms assumptions as it is shorter and longer than the regular ones, respectively.

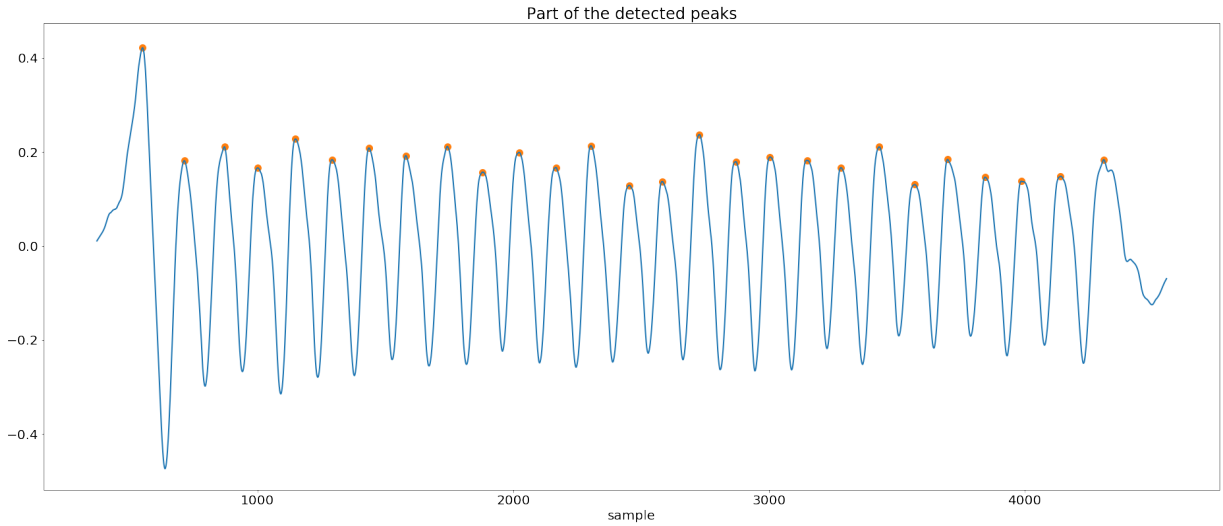


Figure 5.17: Part of the detected peaks in the filtered signal. Each detected peak is marked by an orange dot.

Figure 5.18: The detected steps for the four regions defined in section 5.1 (top). The amount of samples between peaks (bottom).

The amount of samples between peaks represents the duration of each step cycle and will be used as a feature in the learning model.

5.2.4 Additional features

A total of 19 features is collected by the previous sections: 9 based on acceleration (3 operations on 3 components), 9 based on angular velocity and 1 being the step duration. Based on these features, some additional features can be extracted.

Firstly, the norm of the acceleration and angular velocity components could be calculated in an attempt to generalise. The advantage of having separate components is that some movements are better represented in one axis than in the others based on how the IMU is attached. The disadvantage is that this restricts the IMU to be attached in the same way every time. If the IMU is turned 90° around its Z-axis, movements which were previously only visible in the X component would now rather be visible in the Y component. The norm is resistant to the exact orientation of the IMU and therefore is added as a new feature. Nevertheless, the separate component features will still be used since the IMU was attached in the same way for both datasets used in the previous sections, although the norm could help since the attachment to the leg will not have been exactly the same. This brings the total of features from 19 to 25 since a norm for each of the three operations (mean, standard deviation, maximum) and for both types of data (acceleration, angular velocity) can be calculated.

Secondly, the inverse of the step duration is added as a last feature. This can be helpful since the outcomes of the prediction are displacements. Looking at the regions in figure 5.18, the step duration becomes smaller when walking faster and displacements becomes bigger. Therefore, the inverse of the step duration might be proportional to the displacement.

5.3 Learning model

The choice of the learning model will be based on two criteria: simplicity and sparsity.

5.3.1 Simplicity

First of all, there are multiple sample regions in the data where the IMU provides better accuracy than the GPS data used as ground truth. The model should therefore be simple enough in order to not learn the inaccurate parts of the GPS data. Secondly, it should also be robust to overfitting too. A more complex model could overfit more easily resulting in learning noise which is clearly present in the data, mostly in the acceleration data. Finally, Occam's razor suggests that the answer making the fewest assumptions is the most likely.

As the figures in the features section (5.2) already show possible correlations visually, it is most likely that these correlations can be found by a simple model too.

5.3.2 Sparsity

The model should be able to work with the limited amount of features provided. Moreover, it is very likely that the result can be based on an even smaller subset of these features since some of the features are very similar and perhaps redundant while other ones may be provide only irrelevant details.

5.3.3 Lasso

Based on the criteria above, Lasso is chosen as the learning model. Lasso is a machine learning method built on regularised linear regression. Its main difference with linear regression is that estimates sparse coefficients. It thereby tries to find a solution using as little as possible features which still result a high accuracy. It thus effectively reduces the number of variables upon which the given solution depends. The solution for each sample will be of the form

$$\hat{y}(x) = w_0 + w_1 x_1 + ... + w_p x_p \tag{5.1}$$

where $w = (w_1, ..., w_p)$ is designated as the coefficients vector and w_0 as the intercept, a constant. $x = (x_1, ..., x_p)$ contains the p features provided as input to the model. The prediction for a new sample with features $\tilde{x}$ will then be represented by $\hat{y}(\tilde{x})$. Training the model means the coefficients w and the intercept w_0 are calculated. This is done by minimising the loss function

$$L = \frac{1}{2n}\|y - w_0 - Xw\|_2^2 + \alpha\|w\|_1 \tag{5.2}$$

where n equals the amount of samples, X is the matrix with shape (n, p) consisting of n rows of samples having p features and α is a constant determining the penalty for the magnitude of the weights. This penalty $\alpha\|w\|_1$ is what differentiates Lasso from basic linear regression. Increasing α will decrease the magnitude of the weights, possibly resulting in a sparse solution with a few significant weights while the others approach zero.

5.3.4 Cross validation

As described above, the factor α in equation 5.2 punishes large weights in an attempt to regularise. α is called a hyperparameter of the model. A hyperparameter is a parameter

whose value is set before the learning process begins. This is in contrast to the parameters w_i which are set by the learning process itself. Estimating α will be done iteratively by comparing different values of α using cross validation.

The main purpose of cross validation is to avoid overfitting. A model that would just repeat the labels of the samples that it has just trained on would have a perfect score but would fail to predict anything useful on yet unseen data. In this case, a 10-fold splitting strategy is used. This means that the data is split into 10 equal parts and that 10 iterations of training will take place. Each iteration excludes 1 of the 10 parts and combines the other 9 as training data. The excluded part is used as test data and the mean squared error (MSE) on this part is calculated as accuracy measure. The average accuracy on the test folds over the 10 iterations is taken as the accuracy of the model. Typically, a 3 to 5 fold strategy is used. However, increasing the amount of folds helps generalising the accuracy for the price of some extra calculation time.

The next step is to tune the hyperparameter α. A list of premised α values is made where after the procedure of cross validation using folds as explained above is repeated for every α in the list. The value of α resulting in the highest accuracy is used as the definite hyperparameter of the model.

5.3.5 Prediction

The above mentioned procedure will now be used to predict the displacement for every sample.

Figure 5.19 shows the relative displacement based on the GPS signal as already illustrated in figure 5.1 together with data which will be used as ground truth. As stated in section 5.1, the theoretical mean for each region is calculated and used as ground truth because the inaccuracy of the GPS data. The ground truth for the samples outside the four regions, mostly containing some steps before the following region is started, are still gathered from the GPS data.

The 10-fold cross validation using the ground truth shown in figure 5.19 results in $\alpha = 3.6e^{-5}$. Figure 5.20 shows the prediction on the complete dataset resulting in an R^2 score of 96.4% and a mean absolute error (MAE) of 0.08m which means a mean error of about 8cm per sample is measured. The best possible score for the R^2 value is 100% and it can be negative because the model can be arbitrarily bad. A constant model that always

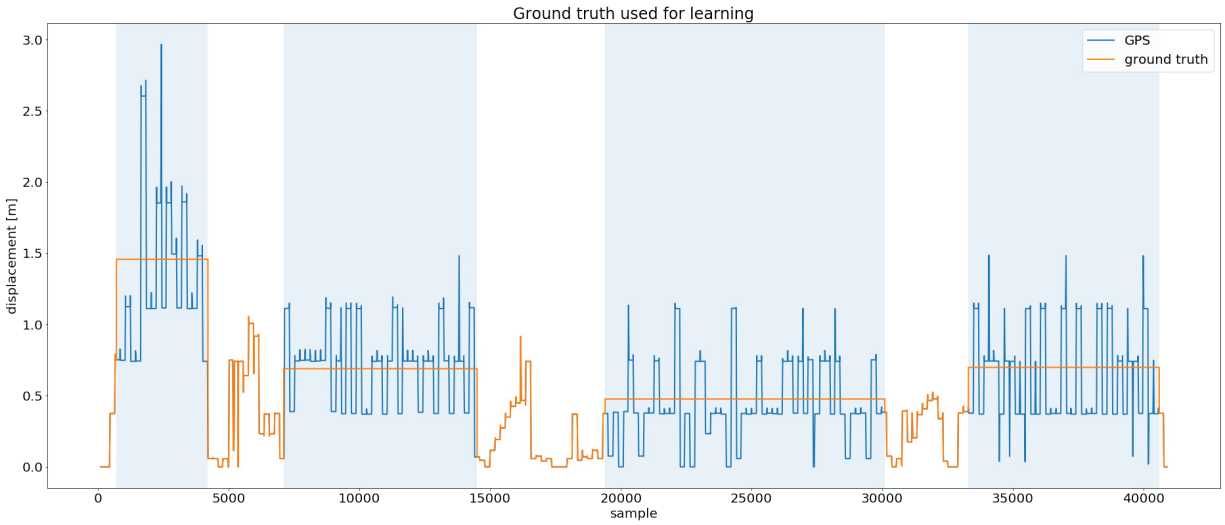


Figure 5.19: The relative GPS displacement of figure 5.1 (blue) and the transformed signal used as ground truth (orange).

predicts the expected value of y, disregarding the input features, would get an R^2 score of 0%. The MAE score is best compared to the one of another prediction since samples are apparently fluctuating above and below the true value, meaning a sample being 8cm larger than the true value and a subsequent sample being 8cm smaller results in a total error of 0cm instead of 16cm when just taking a sum. A constant prediction using a mean step length of 0.6m per 100 samples results in an R^2 value of -0.6% and an MAE of 0.25m per sample, which is more than times the MAE of the previous prediction. Of course, it has to be noticed that predicting samples which are also used as training data is very prone to overfitting. It is however a good first test to see if the model is able to learn the expected results taking into account that only a handful set of features was used and simplicity of the model was key, two principles which already help preventing overfitting.

Table 5.8 provides an important measure to test accuracy in this specific case. The total displacement for both the Lasso prediction and the mean step length prediction over each region is calculated. It is clear the Lasso prediction performs excellent on the training data.

The weights w_i as described in equation 5.1 are displayed in figure 5.21. It confirms the model fulfils the condition of sparsity as only 8 of the 26 features have weights different from zero. The three most important features are the three operations on the Y-component of the angular velocity. This exactly corresponds to the observations in section 5.2.2 concerning the Y-axis best capturing angular velocity due to the alignment with the rotation axis of

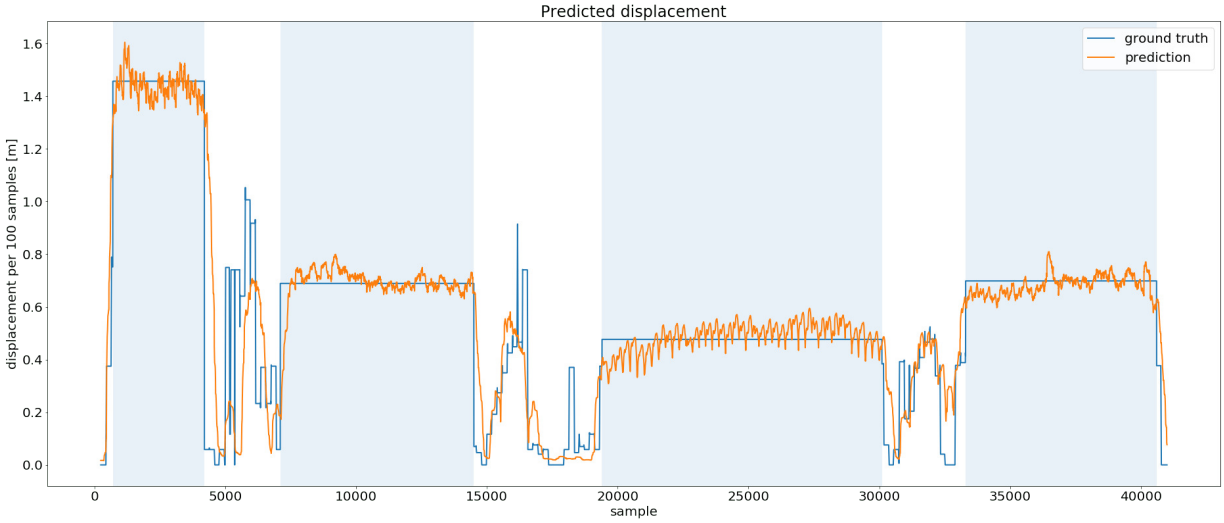


Figure 5.20: Predicted displacement per 100 samples.

region	Lasso pred. (m)	Mean step length pred. (m)	Lasso error (m)	Mean step length error (m)
fast	50.2	21.0	0.8	30
regular 1	51.1	44.4	0.1	6.6
slow	52.3	64.2	1.3	13.2
regular 2	49.9	43.8	1.1	7.2

Table 5.8: The total displacement and error for both predictions over the four regions. The real displacement equals 51m for every region.

the upper leg. Figure 5.22 investigates how cross-validation was performed. The mean squared error increases with increasing α. A value of $\alpha = 4e^{-4}$ (indicated in red) is taken as a trade-off between increasing MSE and decreasing amount of features. This again is a form of regularisation and should lower the performance on the training set slightly in return for a possible increase in accuracy on unseen data.

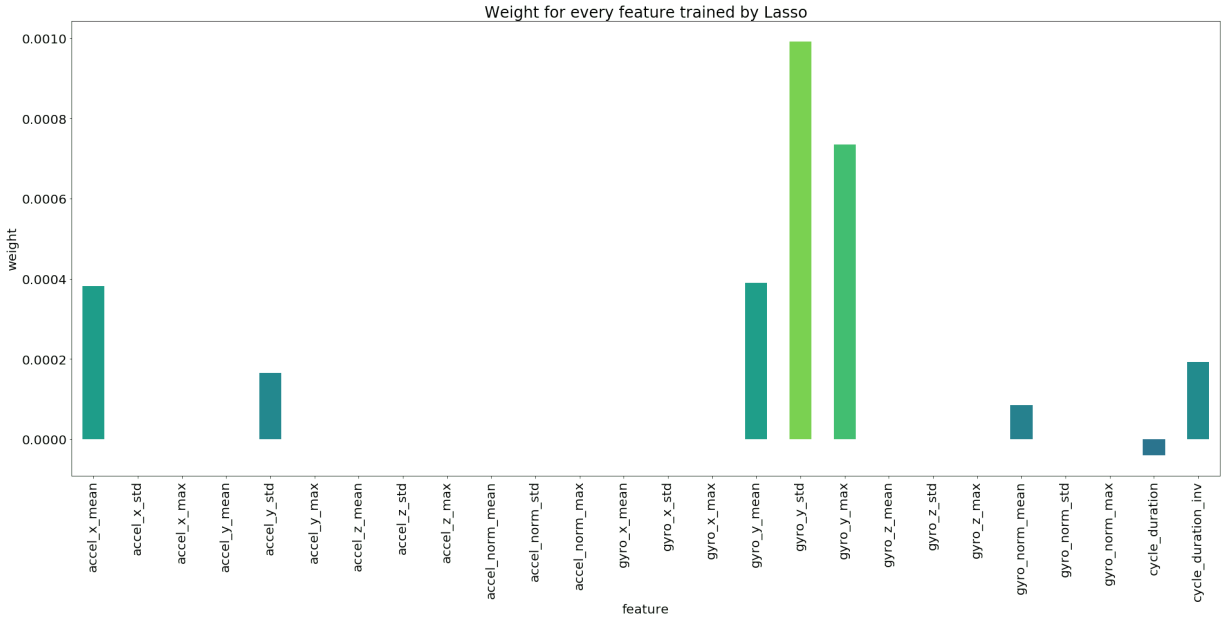


Figure 5.21: Weight w_i for each feature calculated by Lasso ($\alpha = 3.6e^{-5}$).

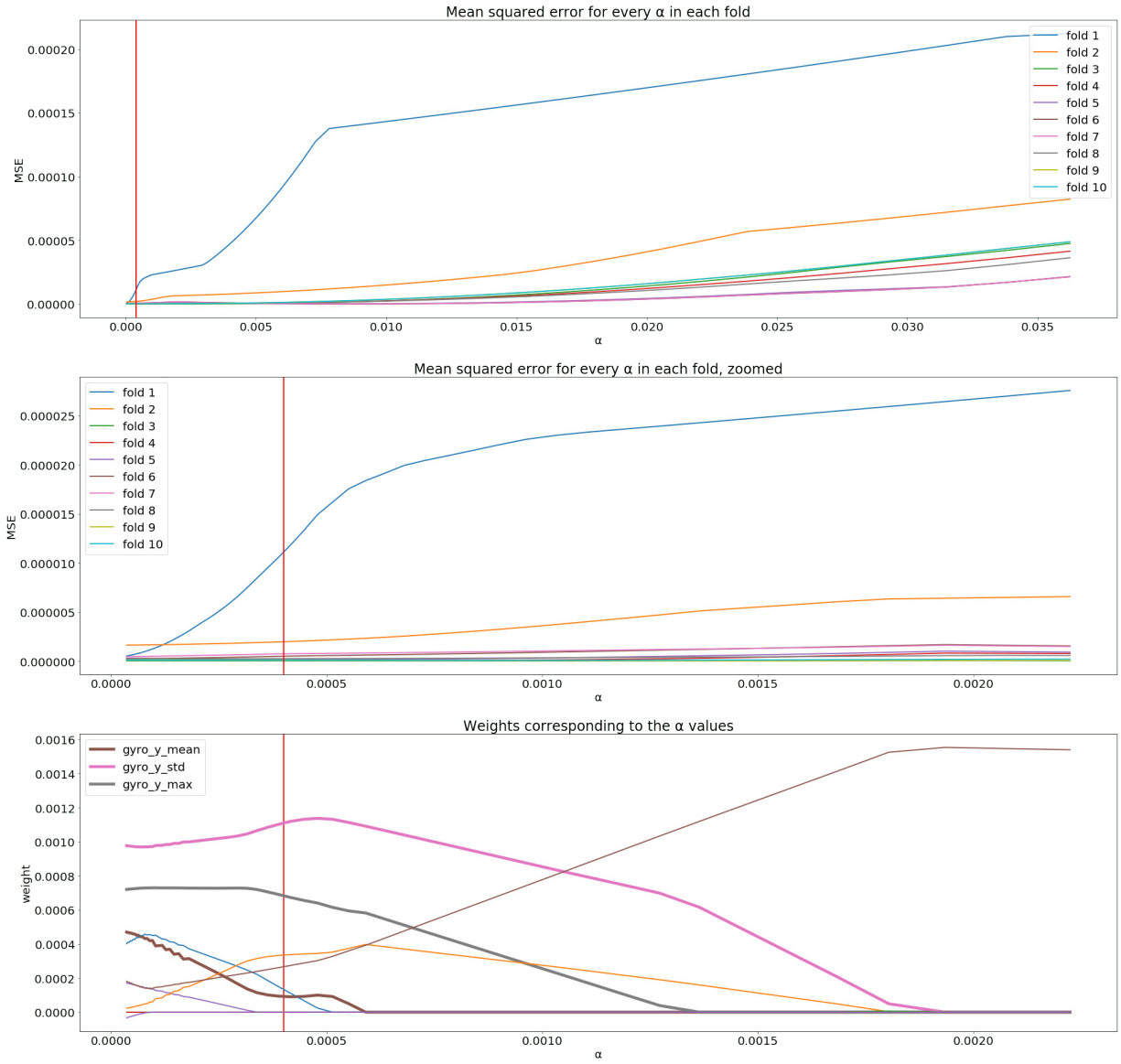


Figure 5.22: MSE for every α in each fold (top, zoomed middle). Weights corresponding to values of α with the lowest MSE (bottom). The red line indicates $\alpha = 4e^{-4}$.

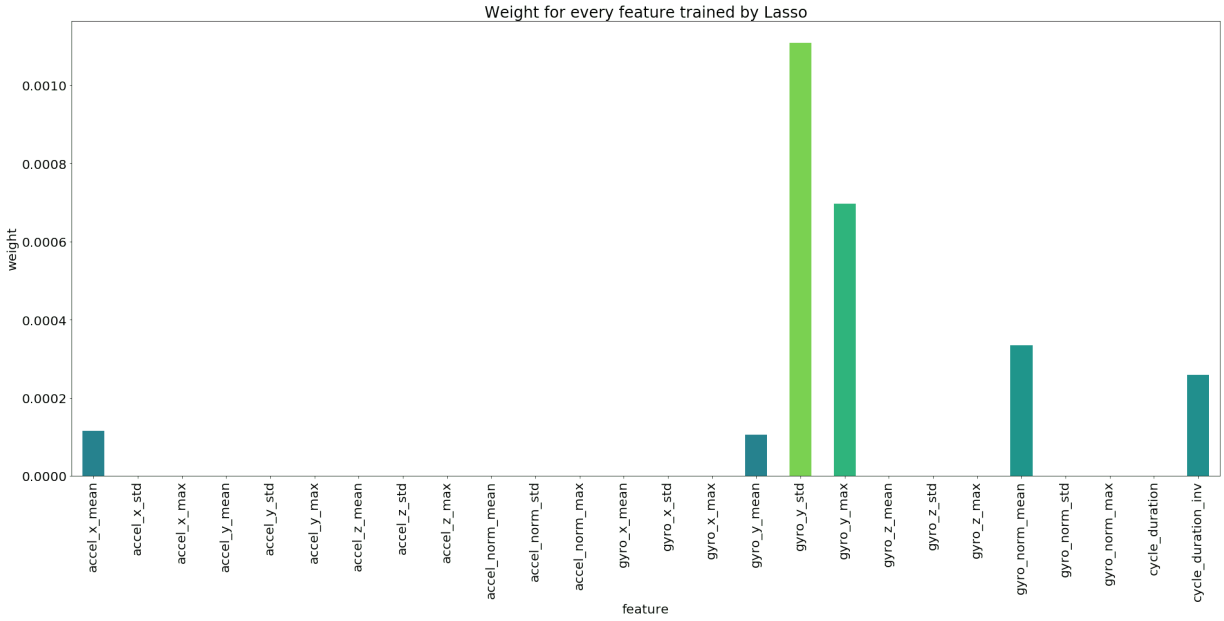


Figure 5.23: Weight w_i for each feature calculated by Lasso ($\alpha = 4e^{-4}$).

6 Dead reckoning results

In what follows, the orientation estimation of section 4 and position estimation of section 5 are combined to predict the coordinates of the walked path.

6.1 Tech Lane Ghent Science Park

The orientation for the walked path in the Tech Lane Ghent Science Park displayed in figure 3.3 was already calculated in section 4.2. This was further improved by adding a constant offset to the complete path as described in section 4.4 and displayed in figure 4.6.

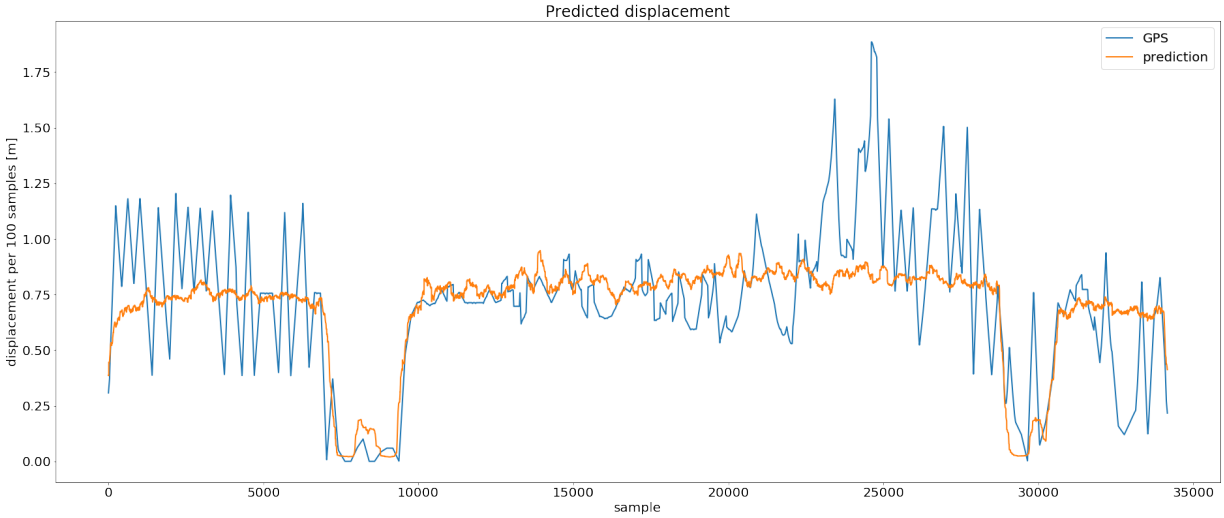


Figure 6.1: Prediction of the displacement of the path shown in figure 3.3.

For predicting the displacement, the Lasso model with hyperparameter $\alpha = 4e^{-4}$ trained on the dataset with the four different step regions is used. The result, shown in figure 6.1, looks very promising. On the one hand, similarities are visible between the GPS data and

the predicted sensor data. Note that the GPS data isn't used in any way to predict the displacement. On the other hand, the biggest difference is the large variance of the GPS data assuming the speed of walking was not steady at all while the prediction expects a more constant speed. Some notes giving more insight in the figure are enumerated below based on the sample range.

- **0 - 7000, 10000 - 23000:** the prediction approximates the mean value of the region.
- **7000 - 10000:** both the GPS and the prediction displacements drop to zero which corresponds to a moment of standing still.
- **23000 - 29000:** the GPS loses quality while the prediction assumes the same pace as before was kept.
- **29000 - 31000:** a second moment of standing still combined with a turn, reflected in both the GPS data and the prediction.
- **31000 - 35000:** a constant pace is visible in the prediction while a non-existing drop of the GPS occurs.

Finally, the angles of figure 4.6 and displacements figure 6.1 are combined in figure 6.2. Apart from the top right region, the predicted path clearly resembles the turns and speed of the ground truth. The error in orientation at the top right is due to the IMU actually thinking this was the correct orientation. A possible reason for the inaccuracy could be the preceded turn or the continuous changes due to stepping. The inaccuracy causes a total difference of 11.5 meters with the ground truth at the end. It is clear that accumulating errors are the biggest flaw in pedestrian dead reckoning.

Because this inaccuracy influences the the whole view, figure 6.3 shows an enhanced prediction where the samples of the top right coordinates are corrected by subtracting a value of 0.2 radians of the angles. Without the inaccuracy, the prediction seems to outperform the GPS signal most of the time. The maximum difference, at the bottom left, equals 4 meters. Note that the prediction isn't altered in any other way than subtracting a constant offset from the region at the top right.

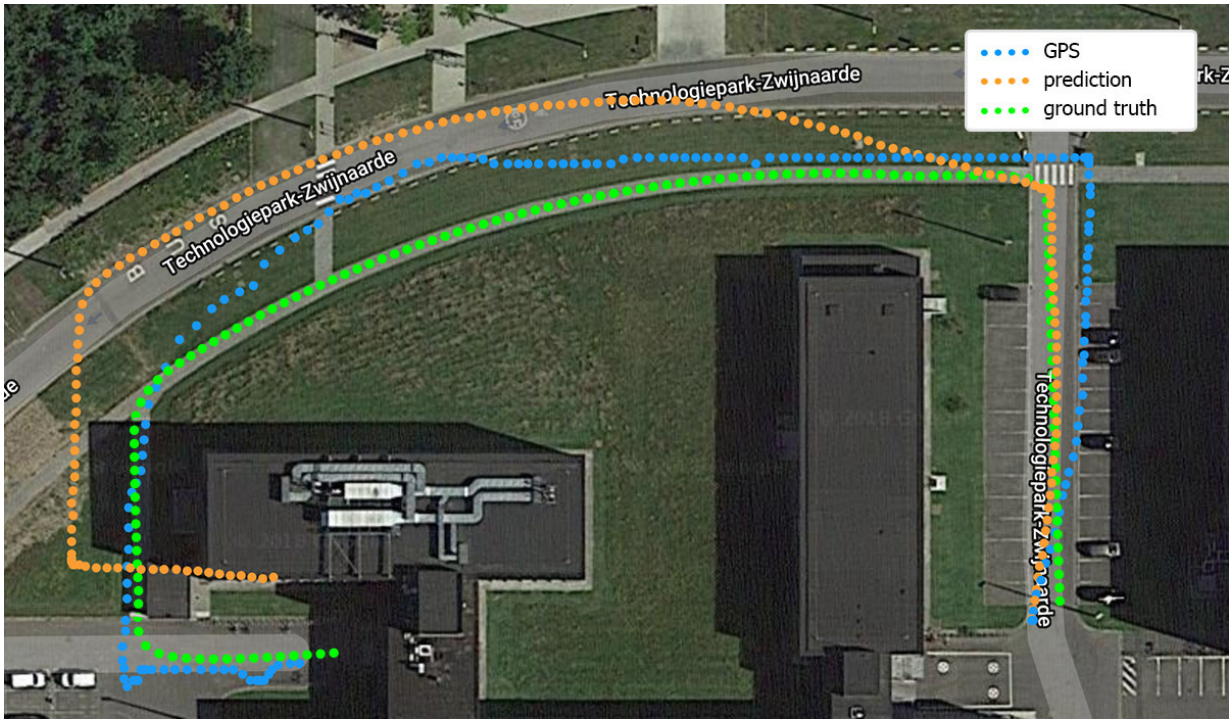


Figure 6.2: Coordinates of the predicted path.

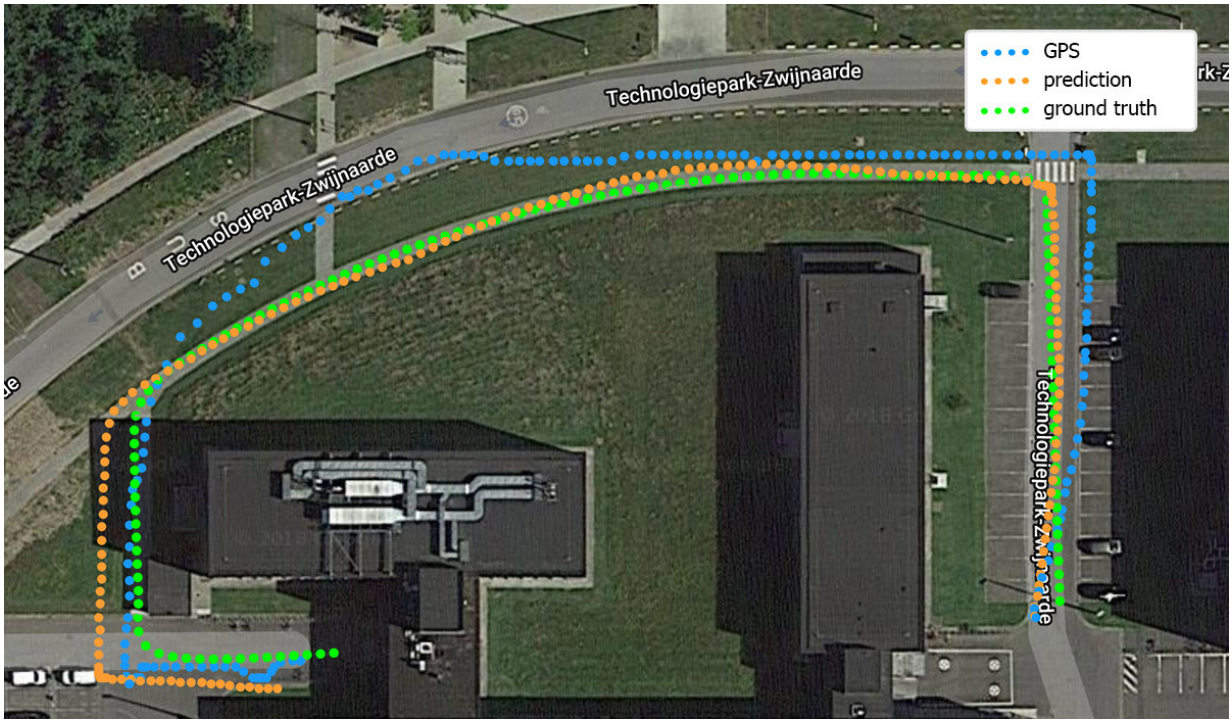


Figure 6.3: Coordinates of the predicted path after subtracting an angle offset of 0.2 radians to the samples at the top right.

6.2 A walk in the city

New data was collected by walking through the city of Ghent, Belgium in order to test the accuracy of the predictions for a real-life situation including obstacles, people and cars. The walked path contains multiple crosswalks, a public square, a steep alley and a lot of turns.

In an attempt to improve predictions, especially to reduce the error in orientation when the sensor loses quality shortly, a second sensor is added. This sensor is attached to the chest in order to reduce the changes due to stepping. It would now be possible to capture the displacements by the sensor on the leg while the orientation is provided by the sensor on the chest. The sample frequency hereby reduces from 200Hz to 100Hz, which is a small trade-off for having two data sources to rely on.

6.2.1 GPS

Figure 6.4 shows the path according to the GPS. Data capturing was stopped several times due to short-circuiting, something which was already observed in section 3. In total, 6 regions were recorded as can be seen by the different colours in figure 6.4. It has to be noted that sometimes movement took place between the start of a region and the end of the previous one. This explains the gaps between some coloured regions in the figure. Also, some parts are clearly diverging the real situation which has to be taken in mind when using the GPS data as ground truth. High buildings obstructing the signal are most likely the reason for this divergence.

6.2.2 Orientation

Figure 6.5 shows the orientation for all regions after calculating the offset. Instead of calculating the offset by providing all samples to the radial baseline algorithm, the first 9000 samples of every region were taken to calculate the offset for each region. In this way, the prediction could also be seen as a real-time prediction where the GPS data is requested a few times in the first 90 seconds of each region. Recall that the objective for outdoor pedestrian dead reckoning was to augment results and reduce GPS usage rather than completely disabling it. The striped part of every region in the figure indicates the use of the GPS. The other parts are completely calculated using the offsets determined in the first 9000 samples.

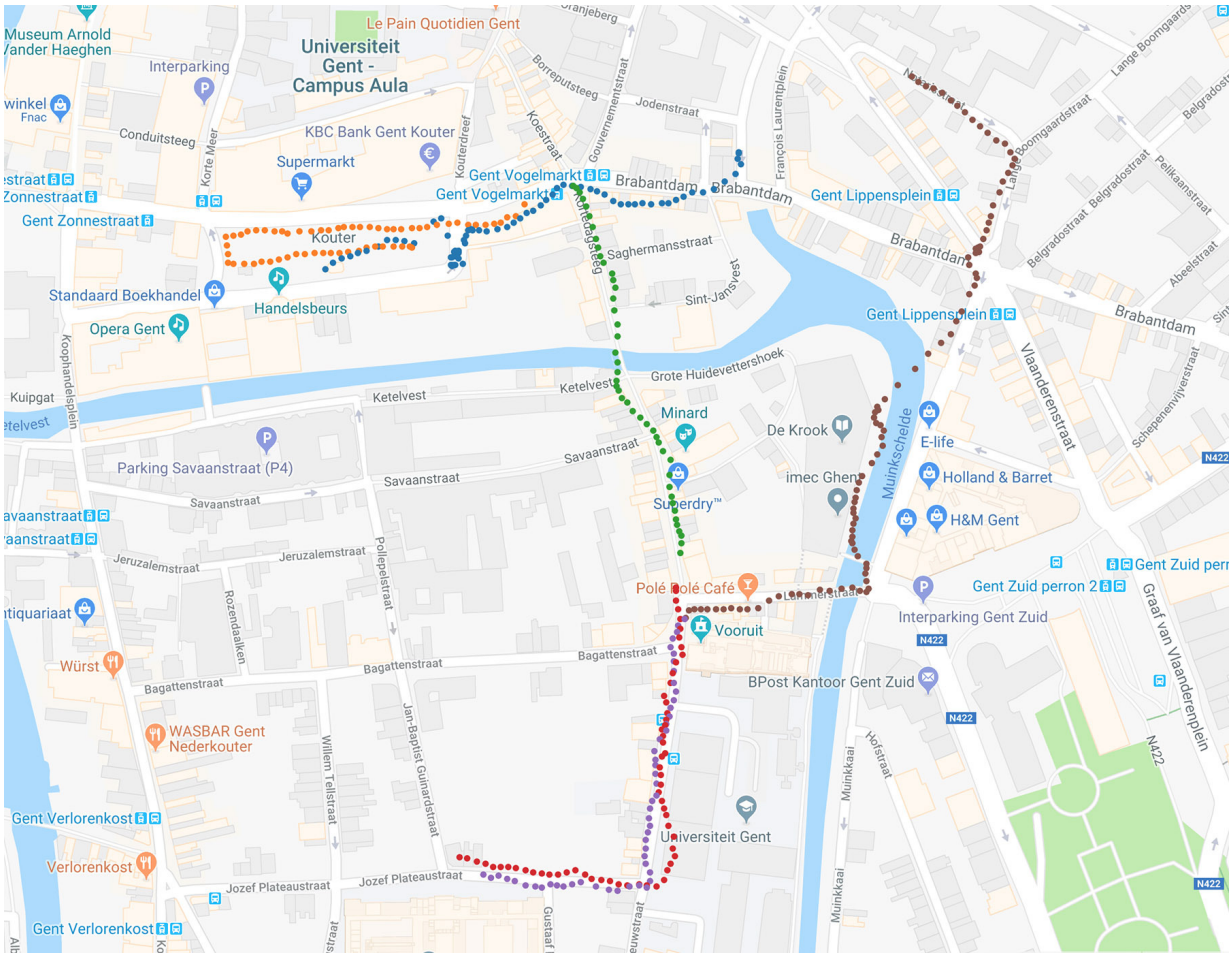


Figure 6.4: GPS data for the different regions walked in Ghent.

The estimates of both sensors seem to align most of the time. Similarities with the GPS data are visible although some differences are clearly present. This could be the diverging parts of the GPS data rather than assuming the sensors both provided inaccurate results.

6.2.3 Displacement

Displacement estimation is done in the same way as in section 5.3 using the Lasso model. It is cross-validated and trained on the sample range 60000-95000 lying in the green and red region since the ground truth doesn't show large anomalies there and still has some variation in walking to learn on. The training set thereby contains ±22% of the total samples, resulting in a large test set of ±78% of the total dataset.

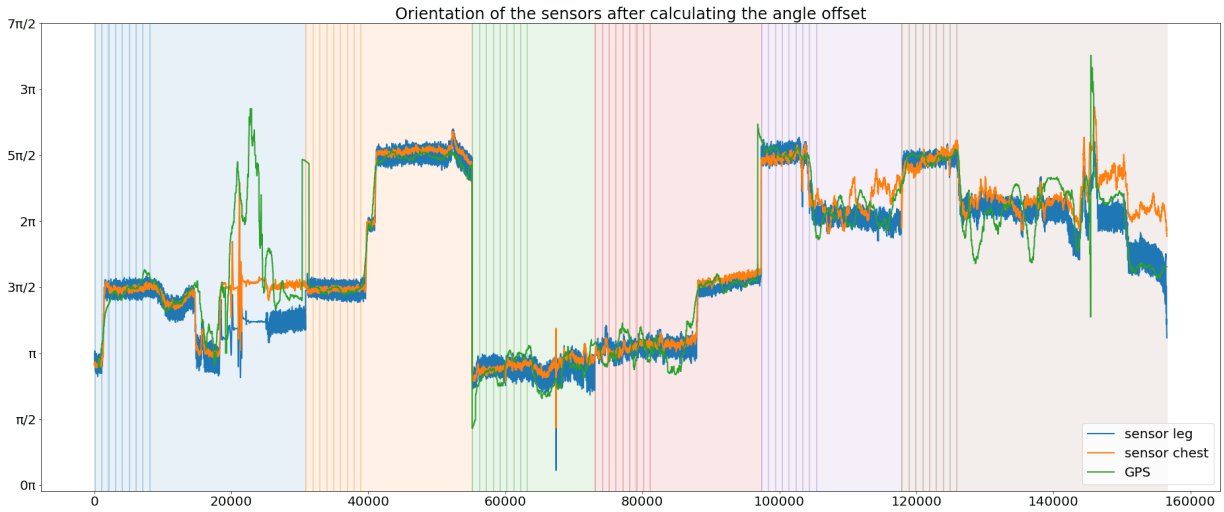


Figure 6.5: Orientation of the sensors after calculating the angle offset for every region.

An α value of 0.01 is chosen. This value is significantly larger than in the prediction on the previous dataset since the ground truth seems to fluctuate more heavily due to inaccuracy as explained in the GPS section above (5.3). The prediction and the fluctuations of the ground truth are visible in figure 6.6. Figure 6.7 shows the weight distribution which only contains two non-zero weights due to the large α value. Again, the Y-component of the angular velocity appears to be an important feature together with the norm.

6.2.4 Prediction

The combined prediction of the orientation and position sections above is shown in figure 6.8. A path for both the calculated orientation of the leg sensor and the chest sensor is drawn in order to compare their quality. The leg sensor unexpectedly seems to be more accurate on average, having only one divergent part at the public square (top left). The chest sensor performs great too but has two divergent parts at the last regions. The used training labels of the GPS are plotted in green.

Figure 6.9 shows the GPS data together with the prediction where the orientation of the chest sensor is used for the first region and the orientation of the leg sensor is used for the other regions. A future improvement would be to investigate a dynamic way to choose the best sensor at any moment. The red lines indicate the real path at points where the prediction diverges. In the other parts, the real path is perfectly resembled by the

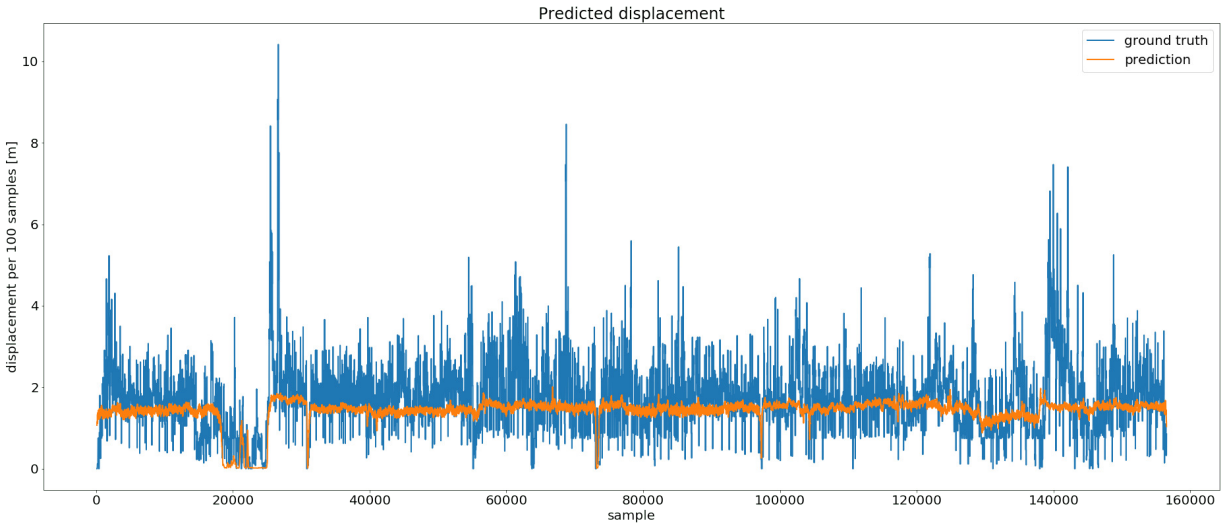


Figure 6.6: Predicted displacement per 100 samples.

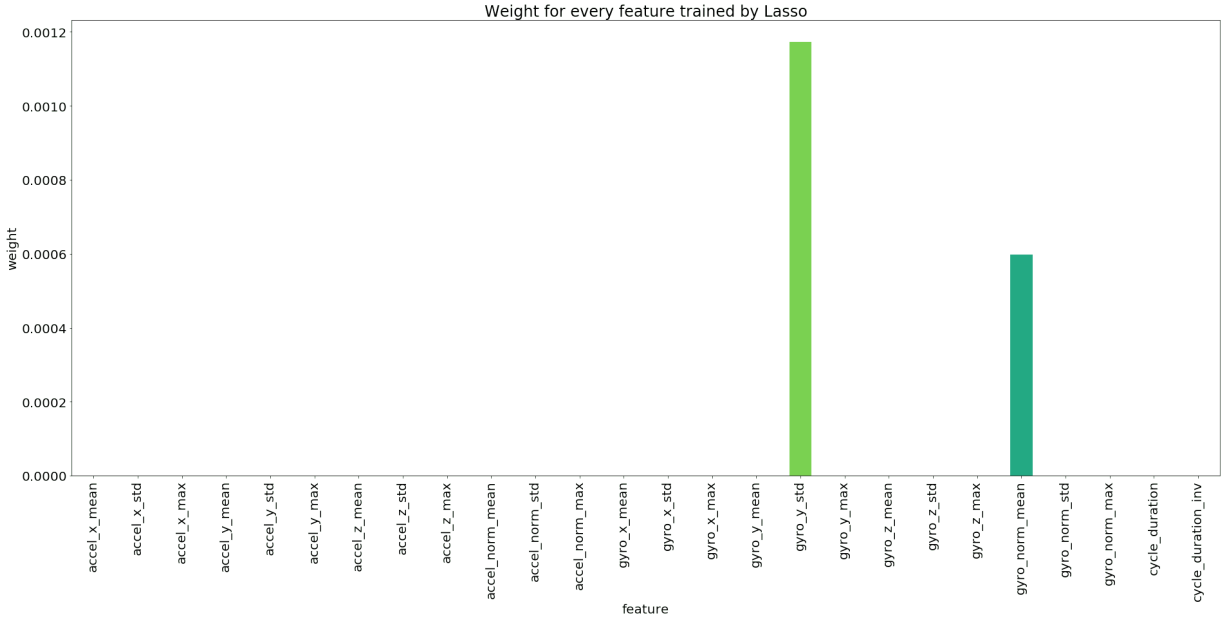


Figure 6.7: Weight w_i for each feature calculated by Lasso ($\alpha = 0.01$).

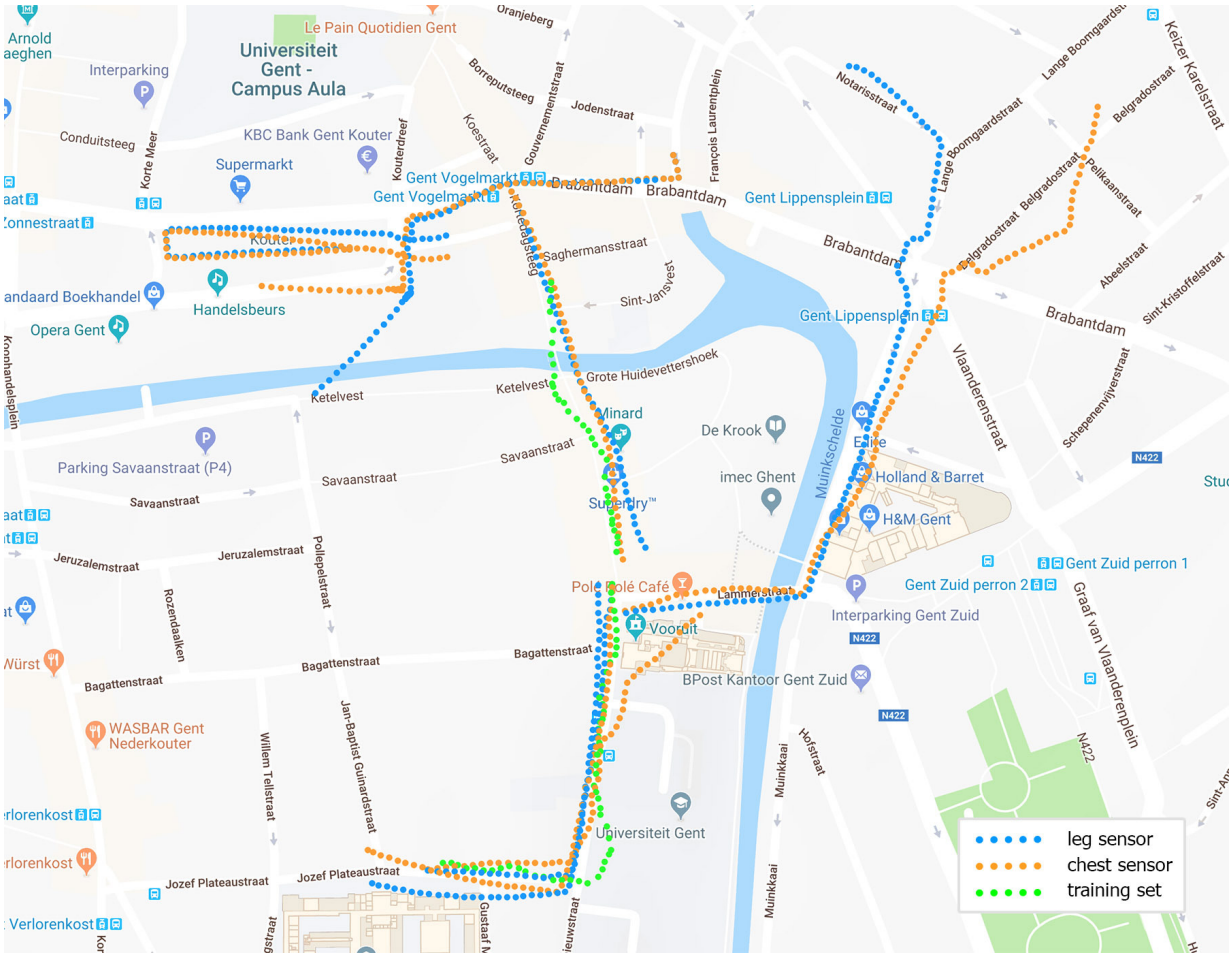


Figure 6.8: Prediction for the walked path in Ghent using the orientation of the leg sensor (blue) and the chest sensor (orange). The green lines indicate the training labels.

prediction!

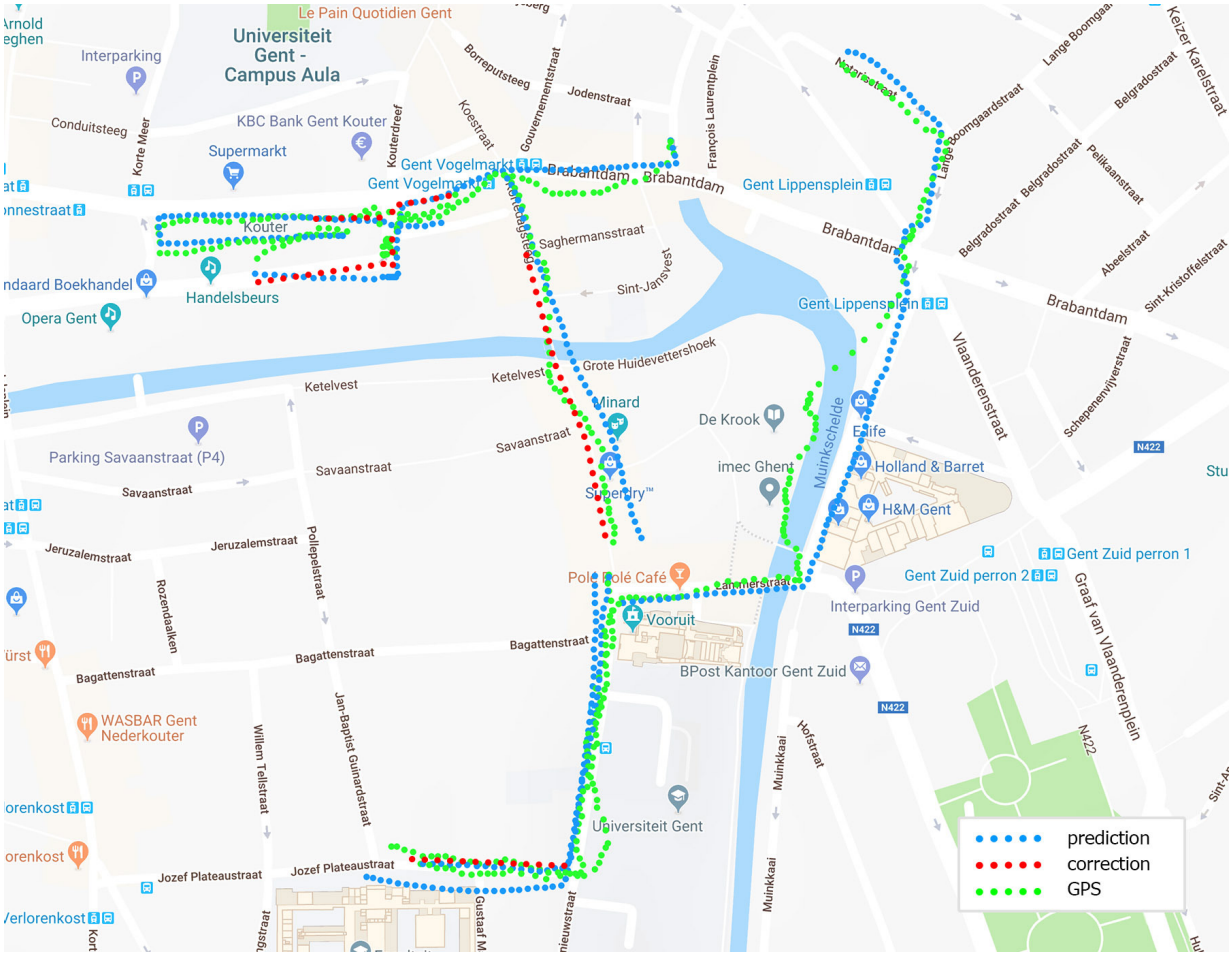


Figure 6.9: GPS signal and the combined prediction of the path in figure 6.8. The red lines indicate the real path at points where the prediction diverges.

6.3 Indoor examples

Indoor position tracking is still an ongoing research topic as typical systems used in outdoor tracking (GPS) are prone to unacceptable errors when used at small positioning scales or unfavourable environments (e.g. thick concrete walls). The goal here is to provide a more accurate and reliable result without the need of pre-installed hardware in the building such as beacons [26].

In what follows, a prediction of an indoor walk is made trained on the dataset of the previous outdoor example. Thereafter, the section is concluded by an example testing the performance when walking at a variety of different speeds.

6.3.1 Plateau

Figure 6.10 shows a map of the walked path inside the Plateau building in Ghent. No words are needed to see why an alternative for GPS can be useful inside.

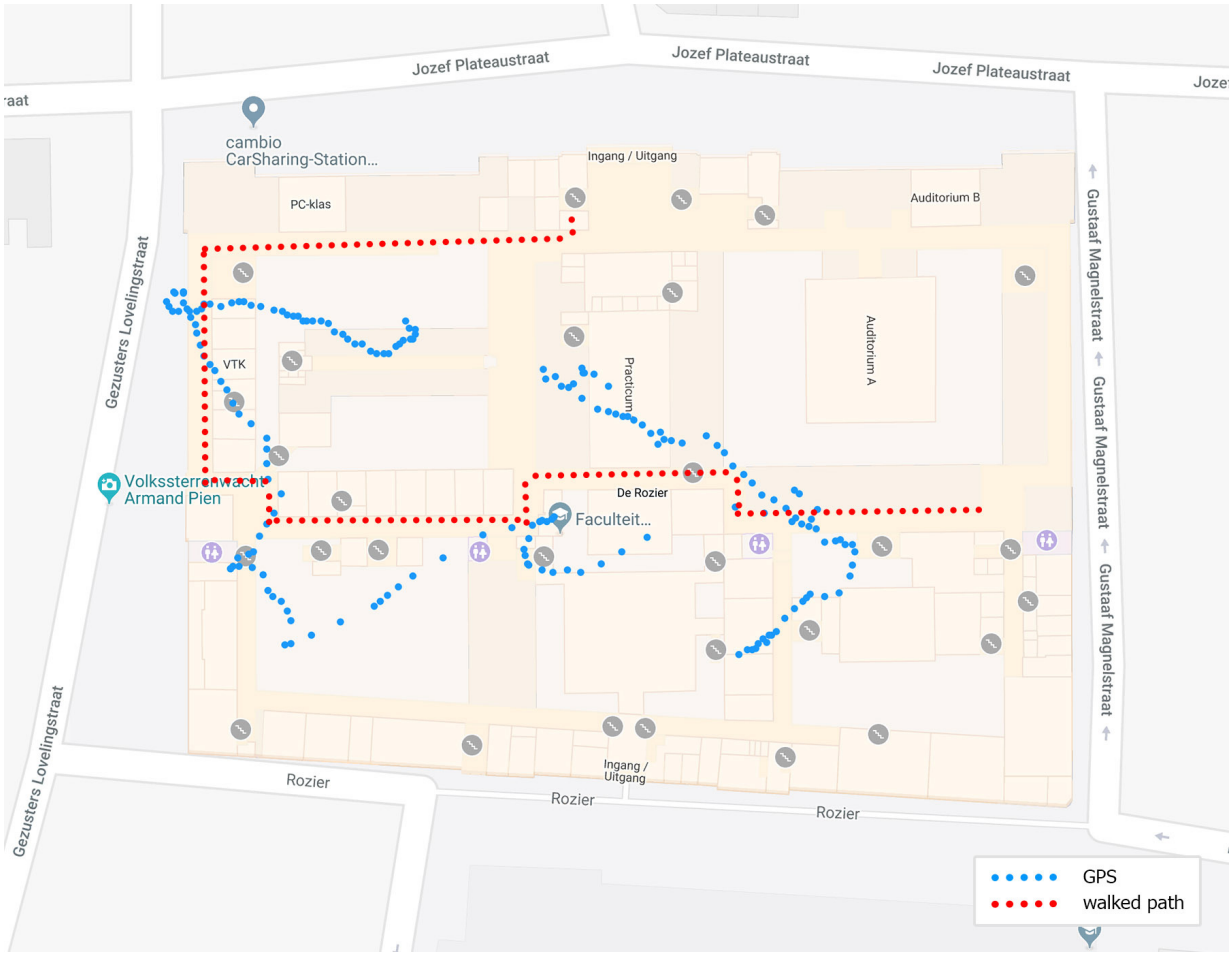


Figure 6.10: Map showing the walked path indoor (red) together with the GPS data (blue).

Figure 6.11 shows the prediction of the displacement based on the trained Lasso model of the city walk in Ghent above. Results lie around 0.6m per step. Figure 6.12 concludes this

example by showing the predicted path. The result couldn't be in bigger contrast to the result of the GPS data. Note the prediction is completely based on outdoor training data except from the start coordinates which were manually set since the GPS data couldn't provide enough accuracy.

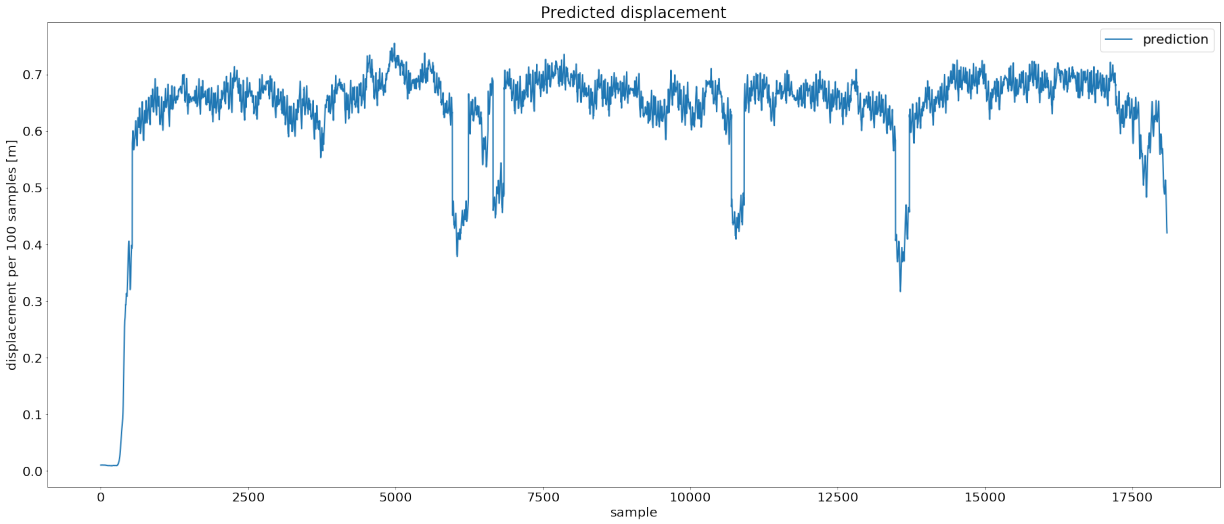


Figure 6.11: Prediction of the displacement of the path in figure 6.10.

6.3.2 Speed test

This last example is the indoor counterpart of the four-region (fast, regular, slow, regular) outdoor dataset of section 5.1. It exists of 28 regions walking a distance of 7 meters at different speeds in order to test the model for short term walks with a lot of variance in speed.

First, the 28 step regions are extracted from the data by calculating a rolling standard deviation over the acceleration components and using a threshold to determine a state of walking. This technique was already used in section 5.2.3 explaining step detection. Figure 6.13 shows the results of the segmentation together with the estimated velocity based on the duration of each region and the fixed distance of 7 meters. The raise in acceleration in between each region is due to a 180° turn to walk the path back and forward.

A value of $\alpha = 1.5e^{-5}$ is chosen for the Lasso model in the same way as before by cross-validation. The only difference is the use of 8 folds in order to give a reliable score of

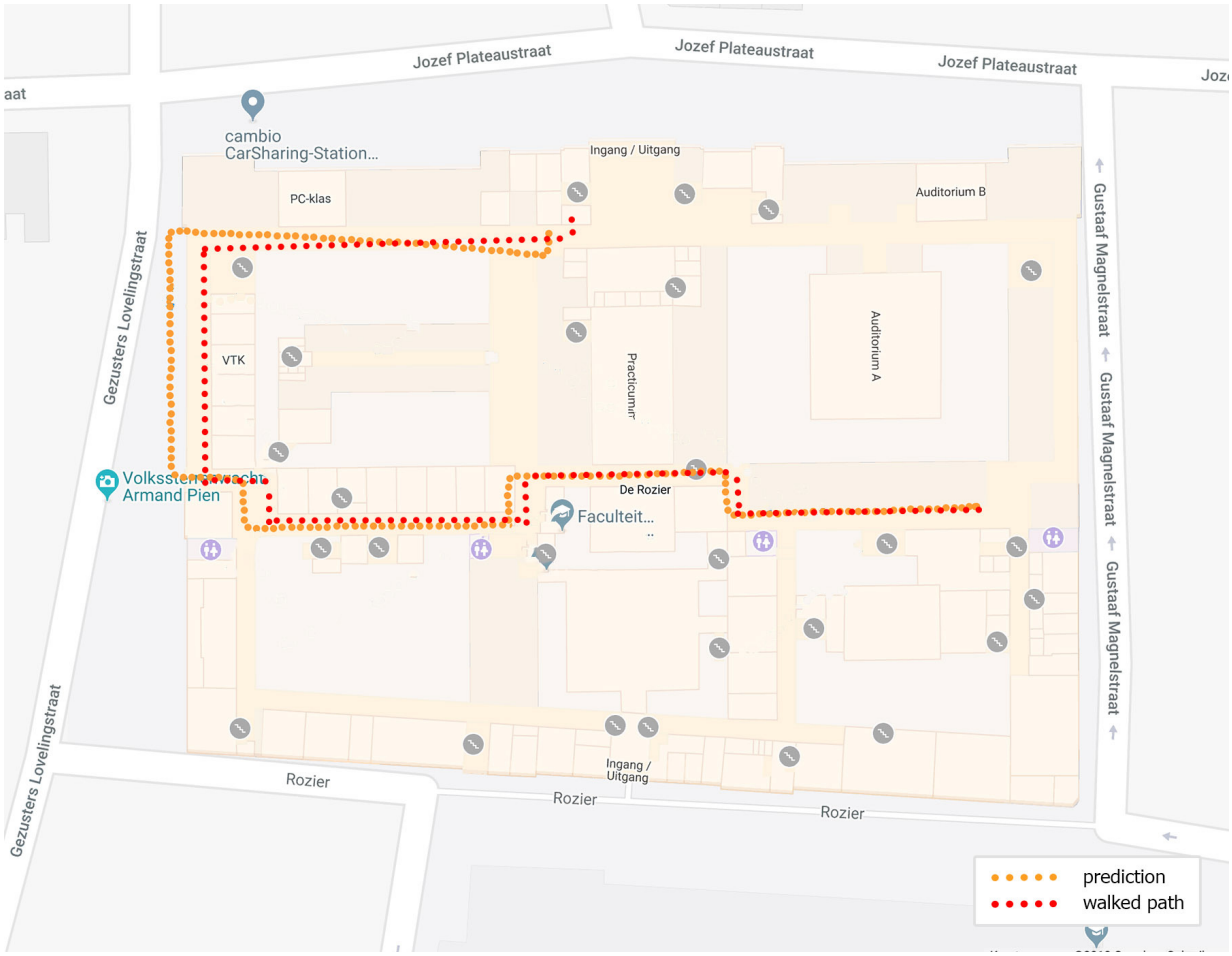


Figure 6.12: Predicted path for the indoor walk in the Plateau building.

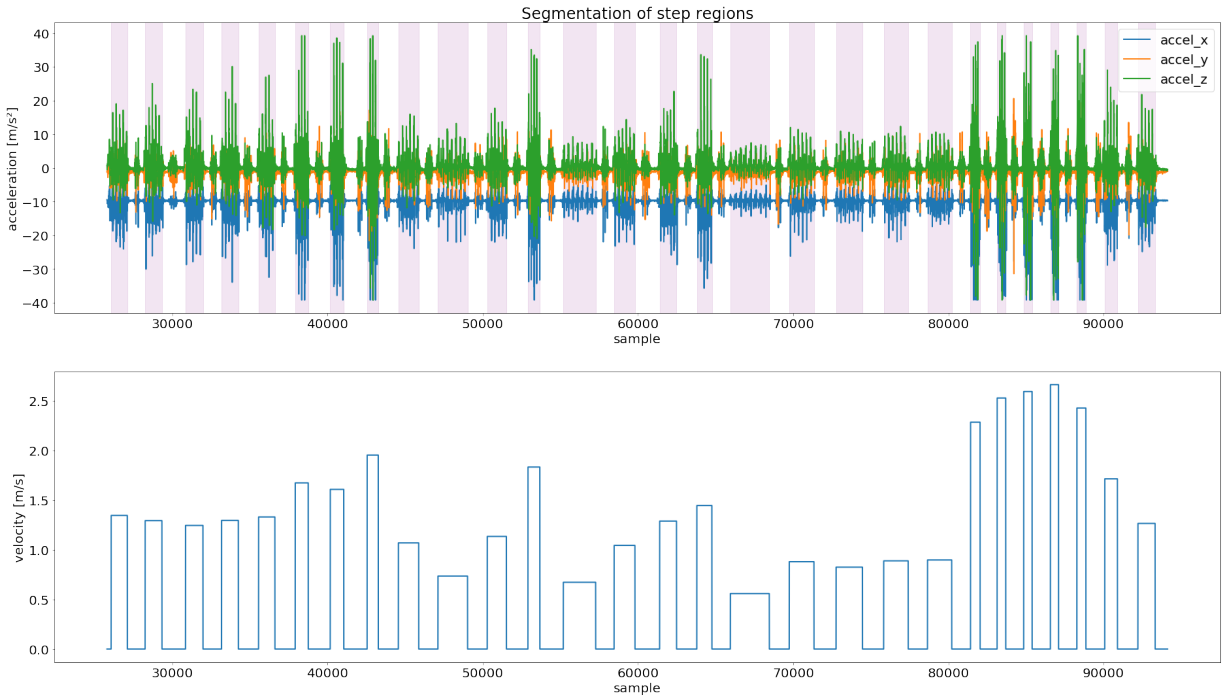


Figure 6.13: Segmentation of the 28 step regions together with the acceleration components (top). The speed for each region based on the duration and the fixed distance of 7 meters (bottom).

accuracy on the dataset itself. Figure 6.14 shows the prediction together with the 8 folds, all coloured differently. Figure 6.15 shows the corresponding weights.

Figures 6.16 and 6.17 conclude this example by plotting the R^2 score per fold and the displacement error over each region. The mean R^2 score equals 89.0% and the mean displacement error equals 0.38m.

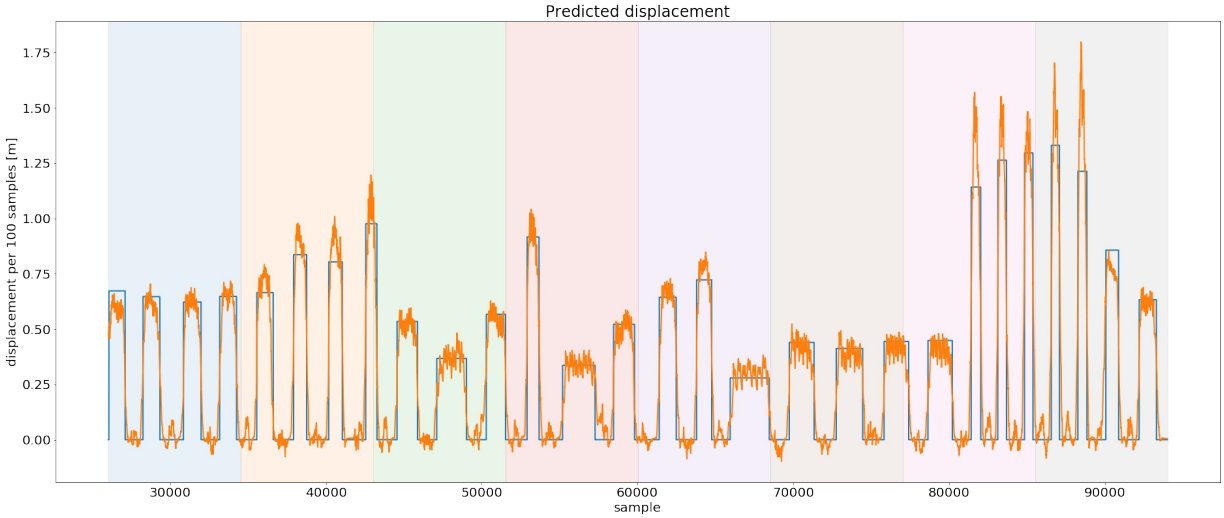


Figure 6.14: Prediction of the 28 segments. The 8 folds are indicated by the background colors.

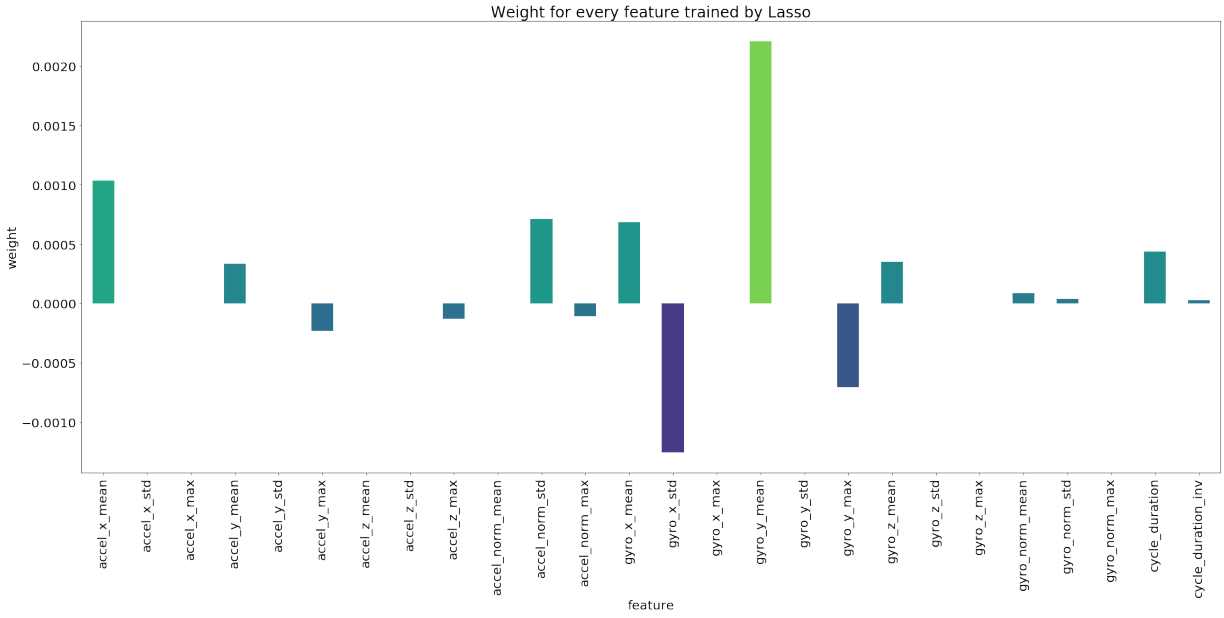


Figure 6.15: Weight w_i for each feature calculated by Lasso ($\alpha = 1.5e^{-5}$).

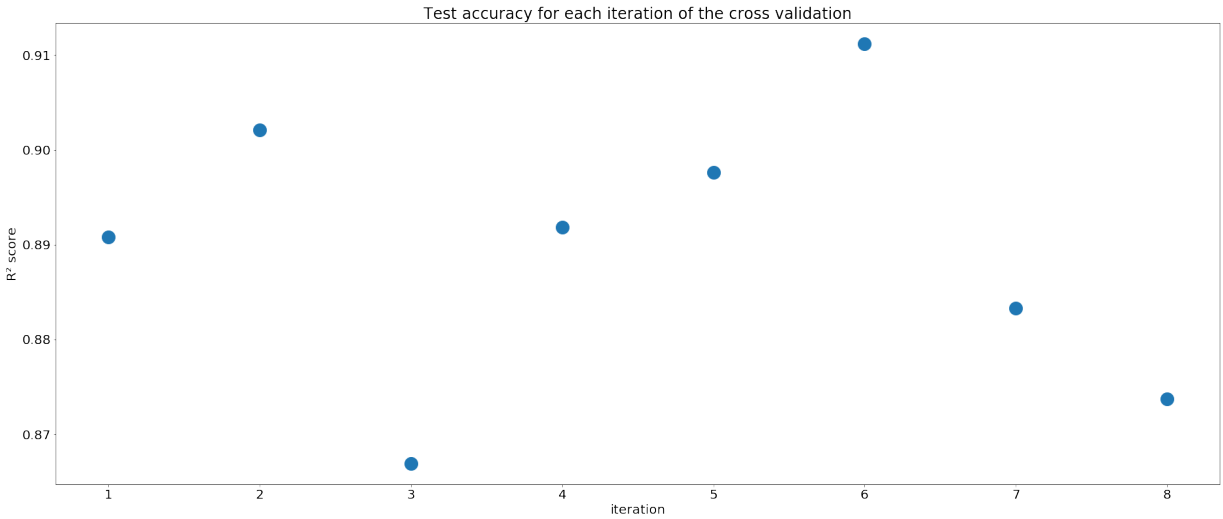


Figure 6.16: R^2 score for every fold of the cross-validation.

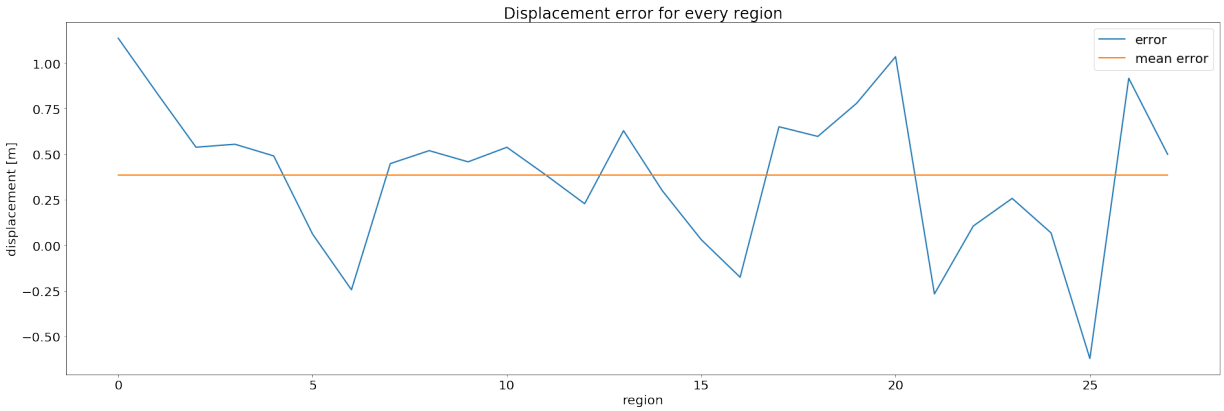


Figure 6.17: Displacement error for the 28 step regions.

7 | Conclusion & future work

In this work a system for capturing inertial motion data via low-cost inertial measurement units was built. The development of this system was described in the first part of this thesis. A hardware system containing multiple IMUs along with the necessary components to extract data from them was presented first. Next, the software to orchestrate the communication between these components as well as data transfer to an external device was described. Finally, we have shown that this system is capable of recording data with a high enough quality to be used in practical use cases.

The use case investigated in the second part of this thesis was pedestrian dead reckoning navigation, a technique to estimate the position of a moving person by capturing data such as acceleration. The current position was then calculated based on the previous position and the relative movement which can be extracted from the captured data. This data was captured by using one or more inertial measurement units as described in the first part.

The transformation to relative movements from the sensor data was done in two main parts: orientation estimation and displacement estimation. In the orientation part, the person's walking direction was estimated by using the IMU's fusioned data consisting of quaternions. In the displacement part, the data from the accelerometer and gyroscope were used to extract features from. A step detection algorithm was then used to extend this set of features by calculating the duration of every step cycle. Finally, these features were provided to Lasso, a linear machine learning model used for predicting the displacement.

The results of combining the orientation and displacement data prove that a low-cost IMU is capable of estimating movement of a walking human. Comparing the results to the data from a GPS module shows that the predicted position using an IMU can even outperform the result of the GPS module. In general, the prediction provides a much smoother and less noisy result which most of the time better describes the real situation.

However, the result will always be prone to drift since every position is based on the

previous one. Therefore, a hybrid form combining the GPS data with the IMU data was used as shown in the example of walking in the city of Ghent. The result definitely is an improvement since the GPS signal presented multiple regions of inaccuracy where the IMU data provided precise positions.

When it comes to indoor prediction, the result showed very detailed changes, especially when looking at the orientations. This is in contrast to the GPS data where the walked path isn't visible at all.

Although these results thus look positive, a few crucial improvements can be made. Looking at the weak spots of the prediction, the most important flaw is the orientation provided by the IMU which sometimes loses quality. Investigating the cause(s) of this quality drop could probably drastically reduce drift. In the city walk example, this problem was assessed by using two IMUs instead of one. This improved the result since the sensors were never inaccurate at the same time. However, the improvement was made by manually selecting the best performing sensor. A future improvement could be the dynamic selection of the best performing sensor at any time. Another possibility would be the prediction of weights in order to combine the outcomes of the sensors in one resulting signal.

This inaccuracy is inevitably correlated to the calibration of the sensors. Using the calibration statuses in a dynamic way to get an estimate of how accurate each sensor behaves at each moment could further reduce the drift problem.

Finally, the interaction between the GPS data and the IMU data could be enhanced for the outdoor use cases. Instead of focusing on reducing GPS requests as much as possible, results could be a continuous combination of both in order to improve the inaccuracy of the GPS, e.g. when walking in a city with a lot of obstructions for the GPS signal.

Bibliography

[1] XSens Technologies. Mti user manual, 2018.

[2] XSens Technologies. Understanding sensor bias (offset), 2018.

[3] Embedded Navigation Solutions VectorNav. Support library: Gyroscope, 2018.

[4] Vittorio Passaro, Antonello Cuccovillo, Lorenzo Vaiani, Martino De Carlo, and Carlo Edoardo Campanella. Gyroscope technology and applications: A review in the industrial perspective. 17:2284, 10 2017.

[5] I2C-bus.org. I2c addressing, 2018.

[6] Arduino.cc. Arduino tutorials: Sketch, 2018.

[7] Arduino.cc. Arduino hacking: Bootloader, 2018.

[8] Arduino.cc. Arduino learning: Memory, 2018.

[9] Daniel Roetenberg, Henk Luinge, and Per Slycke. Xsens mvn: full 6dof human motion tracking using miniature inertial sensors. xsens motion technologies bv. Technical report, 2009.

[10] D. M. Karantonis, M. R. Narayanan, M. Mathie, N. H. Lovell, and B. G. Celler. Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. *IEEE Transactions on Information Technology in Biomedicine*, 10(1):156–167, Jan 2006.

[11] H. Zhou and H. Hu. Reducing drifts in the inertial measurements of wrist and elbow positions. *IEEE Transactions on Instrumentation and Measurement*, 59(3):575–585, March 2010.

[12] M. Nabil, M. B. Abdelhalim, and A. AbdelRaouf. A new kalman filter-based algorithm to improve the indoor positioning. In *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 236–242, Sept 2016.

[13] A. R. Pratama, Widyawan, and R. Hidayat. Smartphone-based pedestrian dead reckoning as an indoor positioning system. In *2012 International Conference on System Engineering and Technology (ICSET)*, pages 1–6, Sept 2012.

[14] Dongjun Hyun, Hyun Seok Yang, Gyung Hwan Yuk, and H. S. Park. A dead reckoning sensor system and a tracking algorithm for mobile robots. In *2009 IEEE International Conference on Mechatronics*, pages 1–6, April 2009.

[15] Seong Yun Cho and Chan Gook Park . Mems based pedestrian navigation system. 59:135 – 153, 01 2006.

[16] L. Ojeda and J. Borenstein. Personal dead-reckoning system for gps-denied environments. In *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, pages 1–6, Sept 2007.

[17] R. Diamant and Y. Jin. A machine learning approach for dead-reckoning navigation at sea using a single accelerometer. *IEEE Journal of Oceanic Engineering*, 39(4):672–684, Oct 2014.

[18] S. Beauregard. A helmet-mounted pedestrian dead reckoning system. In *3rd International Forum on Applied Wearable Computing 2006*, pages 1–11, March 2006.

[19] M. Edel and E. Köppe. An advanced method for pedestrian dead reckoning using blstm-rnns. In *2015 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–6, Oct 2015.

[20] Y. Kim, O. S. Eyobu, and D. S. Han. Ann-based stride detection using smartphones for pedestrian dead reckoning. In *2018 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–2, Jan 2018.

[21] M. S. Pan and H. W. Lin. A step counting algorithm for smartphone users: Design and implementation. *IEEE Sensors Journal*, 15(4):2296–2305, April 2015.

[22] S. H. Shin, C. G. Park, J. W. Kim, H. S. Hong, and J. M. Lee. Adaptive step length estimation algorithm using low-cost mems inertial sensors. In *2007 IEEE Sensors Applications Symposium*, pages 1–5, Feb 2007.

[23] X. Yun, E. R. Bachmann, H. Moore, and J. Calusdian. Self-contained position tracking of human movement using small inertial/magnetic sensor modules. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2526–2533, April 2007.

[24] Agata Brajdic and Robert Harle. Walk detection and step counting on unconstrained smartphones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 225–234, New York, NY, USA, 2013. ACM.

[25] M. Kourogi and T. Kurata. A method of pedestrian dead reckoning for smartphones using frequency domain analysis on patterns of acceleration and angular velocity. In *2014 IEEE/ION Position, Location and Navigation Symposium - PLANS 2014*, pages 164–168, May 2014.

[26] Rainer Mautz. Overview of current indoor positioning systems. *Geodezija ir Kartografija*, 35(1):18–22, 2009.

[27] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara. A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu. pages 37–42, Aug 2009.

[28] C. Qiu and M. W. Mutka. Self-improving indoor localization by profiling outdoor movement on smartphones. In *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–9, June 2017.

[29] Raul Feliz, Eduardo Zalama, and Jaime Gómez-García-Bermejo. Pedestrian tracking using inertial sensors. 3, 01 2009.

[30] T. H. Riehle, S. M. Anderson, P. A. Lichter, W. E. Whalen, and N. A. Giudice. Indoor inertial waypoint navigation for the blind. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5187–5190, July 2013.

[31] R. Zhang, H. Yang, F. Höflinger, and L. M. Reindl. Adaptive zero velocity update based on velocity classification for pedestrian tracking. *IEEE Sensors Journal*, 17(7):2137–2145, April 2017.

List of Figures

List of Tables