Application of an automated sensor to segment alignment method for IMU-based kinematical joint angle estimation during treadmill cycling

JUNE 2017

A.P. van der Zwet
MENS EN TECHNIEK | HAAGSE HOGESCHOOL

Application of an automated sensor to segment alignment method for IMU-based kinematical joint angle estimation during treadmill cycling

Author:	A.P. van der Zwet
Student number:	12104582
Study:	Human Kinetic Technology
University:	The Hague University of Applied Sciences
Date:	14th June 2017
First supervisor:	A. Lagerberg
Second supervisor:	H. Faber

Preface

Being a professional cyclist for ten years, I know how big the impact of innovations are to the performance and results of the athletes. These innovations are not only found in the gear that they use, but also in the types of data about the athlete that is available to the coaches. A personal vision and goal is to make live kinematical data, online available to coaches, athletes, and media very soon. Hopefully the implementation of the suggested algorithm for sensor to segment alignment can be used in future movement analyses during cycling on the open road.

This work has been created in context of graduating as Human Kinetic technologist at the Hague University of Applied sciences, it is written for everyone who is connected to The Hague University and anybody who is interested in motion analysis with IMU's and in undertaking the challenges of sensor to segment alignment. This work and provided codes may be used by all who want to implement the algorithm in future studies.

Acknowledgments go to the teachers of Human Kinetic Technology, with special thanks to; Herre Faber for his help in implementing the algorithm, Aad Lagerberg as my project supervisor, and Mark Schrauwen for his support in providing the IMU hardware.

Enjoy reading!

Kind regards,

Arno van der Zwet
Graduate student Human Kinetic Technology

Nomenclature

Word	Definition
Segment	In this work a segment denotes an limb of the human body or a physical element which rotates around a certain axis, these can be hinge or spherical
White noise	Small variations in a signal, most occurring at analog sensors which can be caused by for instance magnetic fields
Functional calibration	Calibration process with specific or arbitrary motions which are used to align the sensors to the segments.
Anatomical frame	Anatomical reference, determined from anatomical landmarks
Neural Network	A programming paradigm which enables a computer to learn from observational data (machine learning/artificial intelligence)
Kinematic constrains	Constraints between rigid bodies that result in the decrease of the degrees of freedom of rigid body system.
Azimuth angle	An angle, measured from the x-axis in the x-y plane in this work denoted as phi (ϕ)
Zenith angle	An angle, measured from the z-axis (also known as inclination angle), in this work denoted as theta (θ).
Underdetermined system	The number of unknown variables is higher than the number of equations
Recording angle	The angle on which the camera is positioned to an object

Mathematical index

Symbol	Definition
$\overline{\hat{j}^i}$	Unit joint axis vector J, for each sensor $i = 1,2$
$\overline{g^i}(t)$	Angular velocity of sensor i at time t
$R_x(\alpha)$	Calculated rotation matrix around the x-axis,
$R_y(\beta)$	Calculated rotation matrix around the y-axis,
$R_z(\gamma)$	Calculated rotation matrix around the z-axis,
$\cdot$	Dot product of two vectors
$\times$	Cross product of two vectors
$R^{N\times 1}$	Column vector with the amount of rows (N)
$\lvert\cdots \quad \cdots \quad \cdots\rvert^T$	Transposed matrix
$\lVert \quad \rVert_E$	Euclidean norm
$\forall$	At all times / for all data points
$sign(x)$	Function that extracts the sign of a real number
$\frac{dy}{dx}$	Derivative
$\frac{\partial y}{\partial x}$	Partial derivative

Notation types used for referencing

(1)	Reference to Equations
(100)	Reference to a code line number in the written script, the code can be found in the appendix or software

Summary

Kinematical joint analysis during road cycling is a new trend. Reviewing the methods for 2D and 3D lower limb kinematical research, the sensor to segment alignment appeared to be the most challenging part due to unknown orientation with respect to the anatomical joints. A methodology that is suitable for future cycling kinematical research under match conditions, needs to be easy to use and fast. A method suggested by Seel et al. (2012) was found that uses a Newton/Gauss optimization protocol for sensor to segment alignment with arbitrary calibration motions. In this thesis only the methodology for the 2D hinge joint is examined, implemented and tested.

For the 2D joint kinematics, only the IMU's gyroscope data was used. The accelerometer data was used for synchronization. Magnetometers were excluded due to the chance magnetic field disturbances. In total, nine measurement were performed using two IMU's. Seven tests were used to test the alignment method using a self-made hinge joint with two segments. This joint-model was also used in the eighth test, which was filmed to test the accuracy of the angular rotation of a mechanical hinge joint. In the ninth and final test, angular joint rotations of a human knee were measured during treadmill cycling and compared to a video reference.

In all tests the Newton/Gauss optimization functioned accordingly and succeeded to find the joint axis direction. For the first six tests, an average of 15 iterations were needed for finding direction the joint axis. The angular rotation test of the mechanical joint showed a root mean squared error (RMSE) of 1.69^{o} and the human knee from the final test a RMSE of 4.3^{o}, which are slightly higher than the results from Seel. The only function from the method that did not reach the hypothesis was the automatic function to determine if the z-axes of the sensors point in the same global direction (sign of the joint axis).

The method shows promising results regarding the estimation of the joint axis direction and the measured angles compared to the video reference. The results may be improved by adding a Kalman filter to remove any drift and white noise and using an optitrack system to set as golden standard instead of 2D video. The function to determine the sign of the joint axis direction should be reviewed in future works.

Samenvatting

Bewegingsanalyses tijdens het fietsen op de openbare weg wordt een nieuwe trend. Tijdens de onderzoeksfase naar toepasselijke methodes, bleek het probleem van de onbekende plaatsing van de sensoren t.o.v. de anatomische assen een grote uitdaging. Na een uitgebreide literatuurstudie zijn er een aantal methodes gevonden die dit oplossen voor zowel 2D als 3D analyses. Eén methode van Seel et al. (2012) differentieerde zich van de andere studies, omdat zijn methode door een functionele kalibratie met willekeurige bewegingen de richting van een gewrichts-as kon vinden. De combinatie van een eenvoudige en snelle kalibratie in de praktijk zorgde voor het besluit het 2D gedeelte van de methode te analyseren, implementeren en te testen.

Tijdens deze studie, de accelerometer data werd alleen gebruikt voor synchronisatie en de rest van de methode maakt gebruik van alleen de gyroscoop data. In totaal zin er negen tests gedaan met twee IMU's. In de eerste zeven tests, werd de kalibratie en optimalisatie methode getest met een zelfgemaakt scharniergewricht. Dit scharniergewicht werd ook gebruikt voor het genereren van hoeken, waar na de kalibratie de berekende hoeken werden vergeleken met hoeken uit de video van de gefilmde test. Als laatste werd de methode getest op een menselijke knie tijdens het fietsen op een gefixeerde fiets.

Bij alle testen werden de richtingen van de gewrichtsassen succesvol gevonden door het Newton/Gauss algoritme. Tijdens de eerste zes testen waren er gemiddeld 15 berekeningen nodig om de richting van de gewrichts-as te bepalen. De bepaalde hoekverdraaiing van het mechanische scharniergewricht uit test acht had een RMSE van 1.69°. De hoekverdraaiing van het menselijke kniegewricht resulteerde in een RMSE van 4.3° vergeleken met de gouden standaard (video). De afwijkingen bleken iets hoger te zijn dan in het artikel van Seel. Het enige wat in het algoritme niet bleek te werken was de functie voor het automatisch bepalen of de z-assen van de sensoren in dezelfde richting gemonteerd zijn (sign/teken van de richting van de gewrichts-as).

De resultaten van het onderzoek laten zien dat er potentie in het algoritme zit voor het vinden van de richting van de gewrichts-as. De resultaten zouden kunnen verbeteren met het toevoegen van een Kalman-filter voor het verwijderen van drift en ruis, of door een andere gouden standaard te kiezen zoals optitrack in plaats van Kinovea. De sign functie voor het zoeken naar het verschil in richting van de bepaalde gewrichtsassen zal in een volgende studie herzien moeten worden.

Table of contents

1. Introduction

Innovation in the sport of cycling are a trending topic; every year, cycling gear manufacturers reveal their newest designs and materials. Also the market for sensory devices is innovating, used by athletes and their coach's sensors deliver more accurate, real-time data and in some cases over great distances using 3G networks. In-field kinematic measurements using Internal Measuring Units (IMU's) are already performed in other sports like alpine skiing[1], but may also be beneficial to the sports of cycling by for instance evaluating an athlete's position on the bike while turning into a corner. Nowadays, motion analyses in cycling are mainly used for optimizing the athlete's position on the bike (figure 1) with the use of optical motion capturing systems like optitrack. These methods have one thing in common: they are performed indoors in lab based setting[2,3]. The optical motion capturing systems in lab based settings make sense, due to the fact that most studies using IMU's are validated using the optical motion capturing systems as a golden standard[4, 5, 6, 7, 8]. To execute kinematic motion analyses during road cycling in the future, research is needed to test the capabilities of IMU-based kinematic motion measurements.

Figure 1, Fixed bike position for measuring the cyclists kinematic movements with a system from Retul. Source: http://motionfit.net/bike-fit/

Kinematics between two IMU's can be calculated with their accelerations, angular velocity rates and magnetic field vectors. To measure human body kinematic movements, IMU's are placed onto body-segments, to measure the movement between the global coordinate system (CS) of these particular segments. For analyzing kinematic movements of the shank relative to the thigh, two IMU's are required (figure 2).

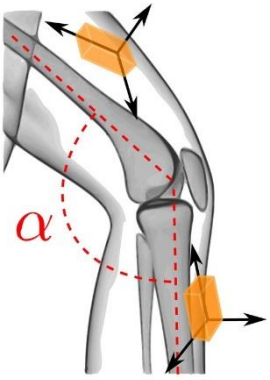

Figure 2, placement of the IMU's on arbitrary positions, the red-dotted line indicates the knee angle and the longitudinal axis of each segment. Source of the picture: [8]

Mounting two IMU's onto body-segments does not fulfill all requirements for performing accurate kinematical motion analyses. The main problem in all analyses on kinematical joint estimation, is the unknown orientation of the sensors with respect to the anatomical segments. To find the exact mounting orientation with respect to the anatomical segments, sensor to segment alignment is required. The sensor to segment alignment is the most crucial part of accurate kinematical joint analyses and many different methods have been found to do so.

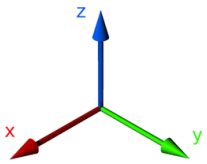

Figure 3, standard coordinate system, denoted in axes X, Y, and Z.

Kinematical research on sensor to segment alignment

When using IMU's, sensor to segment alignment is the most important challenge in performing accurate and valid kinematical research, therefore literature was reviewed to decide which method for sensor to segment alignment to use for this study. Only studies considering the kinematics of the lower limb were reviewed.

There is a difference in methods used for 3D and 2D kinematical research. Most methods in 3D kinematical research use accelerometer, gyroscope, and magnetometer data. Methods in 2D kinematics only use accelerometer and gyroscope data. Depending on the method, only accelerometer data[9,10], gyroscope data[11], or a combination of the two data-types[12] were used.

3D sensor to segment alignment can either be done anatomical or functional. Anatomical alignment was used in a study by Picerno et al., anatomical landmarks were used for sensor to segment alignment, these landmarks were recorded using a magnetometer aided IMU mounted on a pointing device. The research resulted in a RMSE of 1.9°, 2.8°, and 3.6° in the sagittal, frontal and transverse plane[13]. Functional calibration has been used by O' Donovan et al. and Favre et al. and found a higher accuracy in the sagittal and frontal plane, but a bigger deviation at the transverse plane compared to the anatomical approach[6,14]. Both anatomical and functional approaches for sensor alignment are very time consuming or require anatomical knowledge.

In 2D kinematical research, four different kind of methods for kinematic joint estimation have been found[15]:

1. Comparing accelerations of the proximal and distal segments of the hinge joint
2. Comparing planar orientation of articulating segments
3. By prediction of neural networking[16]
4. By combining the first and second method[12]

All of the above methods have their own benefits and drawbacks. A study by Willemsen et al. used only accelerometer data to calculate the sensor orientation manually, by using gravity during a static stance phase and picture frames[10]. A similar approach was used by Dejnabadi et al. where the shank was fixated during a calibration flexion of the thigh. This study showed a very low RMSE of 1.3° with respect to the photogrammetry reference[4].

Studies by Tong et al. used planar orientation of the articulating segments using single uniaxial gyroscopes. The gyroscope drift was cancelled using kinematical reset, but still had a high RMSE (6.42°) compared to the above accelerometer based studies[11]. The drift cancellation from Tong was improved by Cooper, using a Kalman filter and obtained an accurate measurement during gait walking (0.7° error)[17]. This shows the importance of the cancellation of drift due to amplification after numerical integration[15, 18]. The sensor fusion drift cancellation using the Kalman filter appeared to be popular because this was also used by other studies[7, 8, 15].

Findlow et al. introduced a new, deviant method by predicting angular rotation, this was done using a regression algorithm on accelerometer and gyroscope data. They predicted an angle with a deviation of 2.3°, but in one scenario a deviation of 7.8° degrees occurred[16].

The latest methodology in 2D kinematic joint estimation was presented by Seel et al. This study presented a method using both accelerometer and gyroscope data, which were used for numerical differentiation of the angular velocity and an estimation of the joint center's

acceleration. For the estimation, a Kalman filter was used to calculate the mediated average between the flexion-extension angles of the knee. This resulted in a very accurate measurement with a RMSE of less than one degree[15,12].

Seel's method was further examined during a following study, comparing gait analyses of a test subject with one prosthetic, and one healthy leg. Again, the method showed very high accuracy with a RMSE of <1^{o} on the prosthesis and a 3^{o} RMSE on the human leg, compared to an optical motion measuring system.

Seel's method stands out by the fact that the IMU's can be mounted on arbitrary positions, but still show high accuracy in the results, where other studies show a lower accuracy[19]. Also the functional alignment of the sensors using arbitrary motions is beneficial. In other methods an exact performance of a true sagittal motion is required[15]. Seel's functional calibration is preferred because this is less time consuming then the anatomical alignment procedure. The algorithm can find the direction of the joint axis in < 20 iterations with N<<4 data points. Another advantage is that the method does not rely on the use of magnetometers, which is beneficial especially in clinical settings[20].

The combination of the functional calibration and the accuracy of the 2D kinematical joint analyses make Seel's method favorable over the other methods which either require anatomical knowledge or are time consuming.

2. Methods

In this chapter, the mathematical and experimental methods are described. In the mathematical method, Seel's algorithm for finding the direction of the joint axis is mathematically explained. Methods for testing the programmed algorithm are explained in the experimental methods (chapter 2.2.4).

Determination of the coordinate system and segments

To simplify the explanation of the method used for calibration and calculation of the angular rotation in the knee joint, the global CS and segments are determined. The knee joint is approximated as hinge joint with a maximum angular rotation of 180°. The Z^J-axis is the rotational joint axis between the two articulating segments, The $X^{1,2}$-axis is the segment's longitudinal axis, both pointing in distal direction. The Y^i-axis is perpendicular to the $Z^{1,2}$- and $X^{1,2}$-plane. The segment numbers are determined from proximal to distal, in this case: the thigh denoted as segment 1 (S^1), and the shank as segment 2 (S^2), an overview of the segments and their CS's are shown in figure 4.

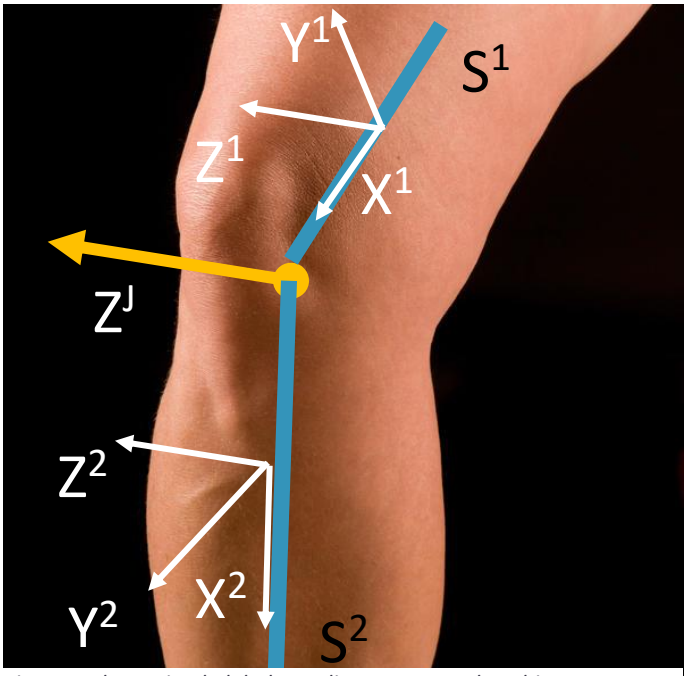

Figure 4, determined global coordinate system, the white arrows represents the anatomical/global CS's of each segment. Zj, is the rotational axis between the two articulating segments. Source of the knee picture: http://www.ambrace.eu/

2.1 Mathematical method

Seel's algorithm for sensor to segment alignment method contains two parts:

- Finding the joint axis direction (2D / hinge joint)
- Finding the joint position coordinates (3D / spheroidal joint)

In this work, only Seel's method for finding the joint axis direction is used to analyze 2D kinematical movements of the knee during cycling. The joint axis direction will be estimated by the use of gyroscope data only ($\overline{g^i}(i = 1,2)$). The algorithm for the joint position coordinates is used for spheroidal joints and requires additionally accelerometer data, but this method is excluded from this work. The algorithm for the joint axis direction is hereafter mathematically explained.

The extended version of this mathematical method, with additional equations and references to the Matlab script that is used for testing, can be found in appendix 1. The Matlab script can be found in appendix 2 and 3, and it's workflow plus detailed information is described in chapter 2.2.2.3.

Seel's algorithm

During the calibration, the direction of the joint axis (Z^j) is estimated. After mounting the IMU's at arbitrary positions, the local CS's of the sensors (figure 5) are not aligned with the segments CS (page 13, figure 4). To denote both local CS's of IMU1 and IMU2 into the global CS, the gyroscope data from each sensor needs to be corrected with a unit vector called J. Each sensor has its own unit joint axis vector, hereafter called $\overline{\hat{f}^1}$ for IMU1 on segment 1, and $\overline{\hat{f}^2}$ for UMU2 on segment 2.

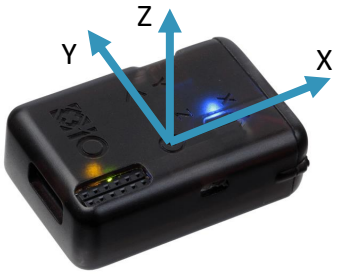

Figure 5, IO x-IMU with case and its representing local coordinate system.

The key to Seel's method, is that it uses the geometrical fact that $\overline{g^1}(t)$ and $\overline{g^2}(t)$, only differ by a rotation matrix (time variant) and the joint angle velocity. Therefore, their projections into the joint plane (parallel to the Z^j axis) have the same lengths for each moment in time. This results in equation (1).

$$\left\|\overline{g^1}(t) \times \overline{\hat{f}^1}\right\|_E - \left\|\overline{g^2}(t) \times \overline{\hat{f}^2}\right\|_E = 0 \; \forall t \qquad (1)$$

In (2), E represents the Euclidean norm: $\|\dots\|_E = \sqrt{{x_1}^2 + \cdots + {x_i}^2}$ and gives the projection length. In this system, $\overline{g^i}(i = 1,2)$ will be known and the $\overline{\hat{f}^i}(1,2)$ are unknown.

2.1.1 Algorithm implementation

Because $\overline{\hat{f}^i}(1,2)$ in the system are unknown, the calibration starts with an initial guess for $\overline{\hat{f}^i}(1,2)$ and a selection of gyroscope data. For this example, 50 data-points for $\overline{g^i}(t)$, $(i = 1,2, t = 1, \ldots, 50)$ are used. The Newton/Gauss method optimizes $\overline{\hat{f}^i}(1,2)$ with the following steps:

1. Initial estimate of $\overline{\boldsymbol{J^i}}$ (1,2) using spherical coordinates
2. Check the deviation/error of the estimates
3. Optimize the estimates using the Newton/Gauss method
   - 3.1 Determine Jacobian matrix from the partial derivatives of (4)
   - 3.2 Determine pseudoinverse of Jacobian Matrix
   - 3.3 Multiply the deviation/error of the estimates by the pseudoinverse of the Jacobian
   - 3.4 Subtract the previous step from the initial estimate spherical coordinates of $\overline{\boldsymbol{J^i}}$ (1,2)
4. Repeat the Newton/Gauss optimization until the deviation in (4) ≈ 0
5. Set $\overline{\boldsymbol{J^i}}$ (1,2) as constants for the angular rotation measurement

1. Initial estimates for J^1 and J^2

The vectors $\overline{\hat{f}^i}(1,2)$ are parameterized in spheroidal coordinates (appendix 1.2.1) from equation (2), these coordinates are used to define $\overline{\hat{f}^i}(1,2)$ in equation (3).

$$\overline{x} = (\varphi_1, \theta_1, \varphi_2, \theta_2)^T \tag{2}$$

$$\overline{\hat{f}^i} = |\cos(\varphi_i)\cos(\theta_i), \cos(\varphi_i)\sin(\theta_i), \sin(\varphi_i)|^T, \qquad i = 1,2 \tag{3}$$

2. Check the deviation/error of the estimates using (1)

After the initial values for $\overline{\hat{f}^i}(1,2)$ are determined, they are used in equation (1) to create the error vector (4). The errors are calculated for all the $\overline{g^i}(i = 1,2)$ data-points and put these in the error vector $\bar{e} = \mathbb{R}^{N\times 1}$. In this example $\bar{e} = \mathbb{R}^{50\times 1}$.

$$error = \bar{e}(k) = \left\| \overline{g^1}(t) \times \overline{\hat{f}^1} \right\|_E - \left\| \overline{g^2}(t) \times \overline{\hat{f}^2} \right\|_E, k = 1, .., N \tag{4}$$

3. Optimization by the Newton/Gauss method

The Newton/Gauss optimization is used for finding a minimum of nonlinear functions and therefore uses the pseudoinverse of a Jacobian matrix. The Jacobian matrix is in this case a $\mathbb{R}^{N\times 4}$ matrix filled with the partial derivatives of *e* with respect $\bar{x}$ $\left(\frac{de}{dx}\right)$. The dx, are calculated using the four angles used to express $\overline{\hat{f}^i}(1,2)$ in spheroidal coordinates: $\varphi_1, \theta_1, \varphi_2, \theta_2$. See appendix 1.2.3.1 for extra information how the angles are written in equation (4) which is derived in the next step.

3.1 Determine the Jacobian matrix containing partial derivatives of (4)

The partial derivatives for the Jacobian matrix were symbolically derived by the Matlab script from appendix 2. The Jacobian matrix filled with the partial derivatives from equation (5 & 6) and has the size of (7), in this example $Jacobian\left(\frac{de}{dx}\right) = \mathbb{R}^{50\times4}$.

$$\frac{\partial e}{\partial x_{\varphi_i}} =$$

$$\frac{\begin{matrix}2|g_x^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\cos(\theta_i)|\,sign(g_x^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\cos(\theta_i))(\,g_x^i\cos(\varphi_i)+g_z^i\cos\theta_i\sin(\varphi_i))+\\ 2|g_y^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\sin(\theta_i)|\,sign(g_y^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\sin(\theta_i))(g_y^i\cos(\varphi_i)+g_z^i\sin\varphi_i\sin(\theta_i))-\\ 2|g_y^i\cos(\varphi_i)\cos(\theta_i)-g_x^i\cos(\varphi_i)\sin(\theta_i)|\,sign(g_y^i\cos(\varphi_i)\cos(\theta_i)-g_x^i\cos(\varphi_i)\sin(\theta_i))(g_y^i\cos(\theta_i)\sin(\varphi_i)-g_x^i\sin(\varphi_i)\sin(\theta_i))\end{matrix}}{\sqrt{2|g_y^i\cos(\varphi_i)\cos(\theta_i)-g_x^i\cos(\varphi_i)\sin(\theta_i)|^2+2|g_x^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\cos(\theta_i)|^2+2|g_y^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\sin(\theta_i)|^2}},\ i=1,2 \qquad (5)$$

$$\frac{\partial e}{\partial x_{\theta i}} =$$

$$\frac{\begin{matrix}2|g_y^i\cos(\varphi_i)\cos(\theta_i)-g_x^i\cos(\varphi_i)\sin(\theta_i)|\,sign(g_y^i\cos(\varphi_i)\cos(\theta_i)-g_x^i\cos(\varphi_i)\sin(\theta_i))(g_x^i\cos(\varphi_i)\cos(\theta_i)+g_y^i\cos(\varphi_i)\sin(\theta_i))-\\ 2g_z^i|g_x^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\cos(\theta_i)|\,sign(g_x^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\sin(\theta_i))\cos(\varphi_i)\sin(\theta_i)+\\ 2g_z^i|g_y^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\sin(\theta_i)|\,sign(g_y^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\sin(\theta_i))\cos(\varphi_i)\cos(\theta_i)\end{matrix}}{\sqrt{2|g_y^i\cos(\varphi_i)\cos(\theta_i)-g_x^i\cos(\varphi_i)\sin(\theta_i)|^2+2|g_x^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\cos(\theta_i)|^2+2|g_y^i\sin(\varphi_i)-g_z^i\cos(\varphi_i)\sin(\theta_i)|^2}},\ i=1,2 \qquad (6)$$

$$Jacobian\left(\frac{de}{dx}\right) = \left[\frac{de^i}{d\bar{x}}\right] = \begin{bmatrix}\frac{\partial e^1}{\partial x_{\varphi_1}} & \frac{\partial e^1}{\partial x_{\theta_1}} & \frac{\partial e^1}{\partial x_{\varphi_2}} & \frac{\partial e^1}{\partial x_{\theta_2}}\\ \frac{\partial e^2}{\partial x_{\varphi_1}} & \vdots & \ddots & \frac{\partial e^2}{\partial x_{\theta_2}}\\ \frac{\partial e^N}{\partial x_{\varphi_1}} & \cdots & \cdots & \frac{\partial e^N}{\partial x_{\theta_2}}\end{bmatrix},\ i=1,\ldots,N \qquad (7)$$

3.2 Determine pseudoinverse of the Jacobian matrix

The next step in the Newton/Gauss method, is calculating the generalization of the Jacobian's inverse. This Moore-Penrose pseudoinverse of the Jacobian is used to compute an optimal solution to the system of linear equations, in a least square sense. In this case the pseudoinverse is denoted in (8) and is hereafter a $\mathbb{R}^{4\times50}$ matrix.

$$|pinv(Jacobian)| = |Jacobian|^+,\ \mathbb{R}^{4\times N} \qquad (8)$$

3.3 Multiply the pseudoinverse with the error vector

The pseudoinverse is now multiplied by the initial error vector from equation (4). Multiplying the $\mathbb{R}^{4\times50}$ matrix of the Jacobian's pseudoinverse by the $\mathbb{R}^{50\times4}$ matrix of the error vector resuts in a $\mathbb{R}^{4\times1}$ column vector. Which can be subtracted from the initial estimates of $\bar{x}$ ($\mathbb{R}^{4\times1}$) and is shown in equation (9)

3.4 Update the initial values of x

Finally the initial estimates of $\bar{x}$ are updated by subtracting the Jacobian's pseudoinverse time the error vector(9).

$$\overline{x}^{new} = \overline{x}^{old} - pinv\left(\frac{de}{dx}\right) * \overline{e} \qquad (9)$$

4. Repeat the optimization

The Newton/Gauss method repeats the steps 1 till 3.4. until the mean of the error vector from equation (4) ≈ 0. Each time the optimization is run, the estimates of $\overline{\hat{f}^i}(1,2)$ are closer to the actual direction of the joint axis.

5. Set joint axis direction estimate as constant

Once the Gauss/Newton optimization has run for 30 times, the spheroidal coordinates of x should have found the actual direction of the joint axis. The values of $\overline{\hat{f}^i}(1,2)$ $\mathbb{R}^{3\times1}$ are set as constants and are hereafter used for calculating the joint angle.

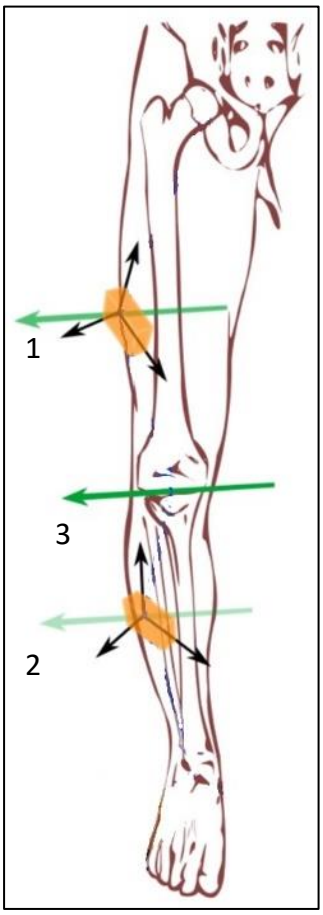

Figure 6, visual representation of the joint direction axes in each IMU (1&2), their local CS (black arrows), and the joint axis and its direction in the global CS (3).

2.1.2 Checking the sign of $\overline{\hat{f}^i}$ (1,2)

The angular velocities per IMU are calculated using equation (11). To check if the Z-axes of the IMU's are pointing in the same half space after mounting (figure 6), the signs of $\overline{\hat{f}^i}$ $(i = 1,2)$ can be checked. This is done by taking a data-point in where the angular velocity ($g^{1,2}(t)$) around the joint can be neglected. When the signs match, the outcome of equation (10) is positive. This means that both z-axes of the IMU's point into the same half space.

$$\overline{g^{temp}} = \begin{bmatrix} 0.001 \\ 0.001 \\ 0.001 \end{bmatrix}$$

$$SIGN = sign(\overline{g^{temp}} \cdot \overline{\hat{f}^1} * sign(\overline{g^{temp}} \cdot \overline{\hat{f}^2}) \quad (10)$$

2.1.3 Angle calculation

The angular rates of the gyroscope $\overline{g^i}(t), (i = 1,2)$ and the unit vectors $\overline{\hat{f}^1}$ and $\overline{\hat{f}^2}$ are used to calculate the angular rotations around the hinge joint, in the global CS. The dot product of the vectors $\overline{\hat{f}^i}$ (1,2) with the IMU's gyroscope rates $\overline{g^i}(1,2)$, will denote the rates into the global CS (11).

$$\overline{g^i(t)^{local}} \cdot \overline{\hat{f}^i} = \overline{g^i(t)^{global}},\ i = 1,2 \quad (11)$$

2.1.3.1 Angular velocity

To express the total angular velocity rates of the joint, the dot product between the IMU's angular velocity $\overline{g^i}(t)$ $(i = 1,2)$ and the direction of the joint axis $\overline{\hat{f}^1}$ and $\overline{\hat{f}^2}$ are taken and subtracted from each other. This results in the total angular velocity (ω) of the joint, given in the global CS.

$$\omega_{gyr} = \overline{g^1}(t) \cdot \overline{J^1} - \overline{g^2}(t) \cdot \overline{J^2} \quad (12)$$

2.1.3.2 Angular rotation

The angular rotations are calculated by numerical integration. The angular rotations at the joint axis are calculated by integrating Omega from equation (12) and is shown in equation (13).

$$\alpha_{gyr} = \int_0^t (\omega_{gyr}(t))dt \quad (13)$$

2.2 Experimental setup

The algorithm from chapter 2.1 and it's Matlab implementation is tested using the methods described in this chapter. The testing of the algorithm consists of three parts:

1. **Calibration test**
2. **Angular rotation test (mechanical joint)**
3. **Bike test (human joint)**

The 'calibration test' is used to test the function of the Newton-Gauss optimization, the 'angular rotation test' will test the accuracy of the calculated angular rotations of a mechanical hinge joint, and the 'Bike test' will test the accuracy of the angular rotations measured on a human knee during treadmill cycling. The results from both the angular rotation test and bike test were compared to a video measurement. The first two tests use a mechanical hinge joint which represents the knee joint.

2.2.1 Hardware

The Joint-model

For the calibration test and angular rotation test a mechanical hinge joint was used, this 'joint-model' was created to simulate 2D angular rotations (figure 7). Due to its small size, arbitrary motions for the calibration can easily be generated, as well as known angles for the algorithm validation. The IMU's are attached onto the joint-model using double-sided tape. Spacers are used to turn and transpose the IMU's mounting to place them at arbitrary and unknown orientations on the segments (page 23, figure 9).

Figure 7, The joint-model with a hinge-joint. The spacers can be used for arbitrary positioning the IMU's to unknown placements on each segment.

IMU hardware

Two IO x-IMU's from the Hague University of Applies Sciences were used for all measurements. Other hardware used for testing are; sports and Velcro tape for markers and sensor mounting on the human test subject, an iPhone 7plus and Sony Cyber-shot DSC-T99 for video registration (30 fps), and a Tacx flow trainer for fixating the Koga Kimera bike.

2.2.2 Software

For the experiments, several software programs have been used. For reading/writing the x-IMU's registers (table 1) and logging their data the x-IMU GUI (version 13.1) from IO technologies was used[21]. A self-written python script was used for the manual data synchronization, Kinovea is used for 2D video marker-tracking and a Matlab script is used for testing the method and calculate angular rotations from the gyroscope data.

Table 1, Registers that are checked/set before testing.

Register	Value
Inertial and Magnetic data rate	128 Hz
Quaternion Data rate	128 Hz
Algorithm mode (if available)	AHRS

2.2.2.1 Sensor output

After logging both IMI's, the x-UMU GUI saved the data into several files (depending on the enabled functions of the IMU). The files containing '_CalInternalAndMag.csv' from each sensor are used for finding the direction of the joint axis (J1 and J2), and to calculate the angular rotations between the two segments. The two CSV-files (comma separated value) contain: the package numbers (data index), accelerometer, gyroscope, and magnetometer data in all directions x-y-z from each sensor. When opened in Microsoft Excel the data is shown in a sheet (N x 1).

2.2.2.2 Synchronization software

A python script was initially designed to load the csv files, find the direction of the joint axis, and calculate the angular rotation, but during the study a Matlab script was requested by H. Faber. Thereafter, the Matlab script was used for testing the method due to Matlab's easy utilization for viewing and manipulating variables. The python script was only used for finding the synchronization point of the two CSV files (chapter 2.2.3).

Workflow

To start the Python program, open the 'main.py' file, it will open a GUI (appendix 4.1, figure 20) which contains buttons, graphs and forms. To open the data in the GUI, push the 'open file' button. Select the '_CalInternalAndMag.csv' file of IMU1 in the first dialog box, then the '_CalInternalAndMag.csv' file of IMU2 in the second dialog box. The UI will automatically load the graphs and show the gyroscope data, to show the accelerometer data, simply select 'accelerometer' in the drop-down list above each graph. Holding the left mouse button allows the user to scroll through the data. Holding the right mouse button allows the user to zoom in/out, the user can thereby easily zoom into the accelerometer-peak created to synchronize the data. The synchronization process is explained in chapter 2.2.3, the code of the python script with comments is added to this work in appendix 4.

2.2.2.3 Algorithm testing

To test Seel's method a script is programmed using Matlab conform chapter 2.1. The Matlab script contains a main program ('SensorAlignmentHinge.m') and two functions ('FindJ.m' and 'AngleG.m'). The main program first loads the IMU data, then runs all the functions to calibrate and find the direction of the joint axis, calculate angular rotations and plot the significant data. A detailed explanation of how the program works can be found below this paragraph and is simple displayed in figure 8. All references to the codes lines (...) can be found in appendix 3.

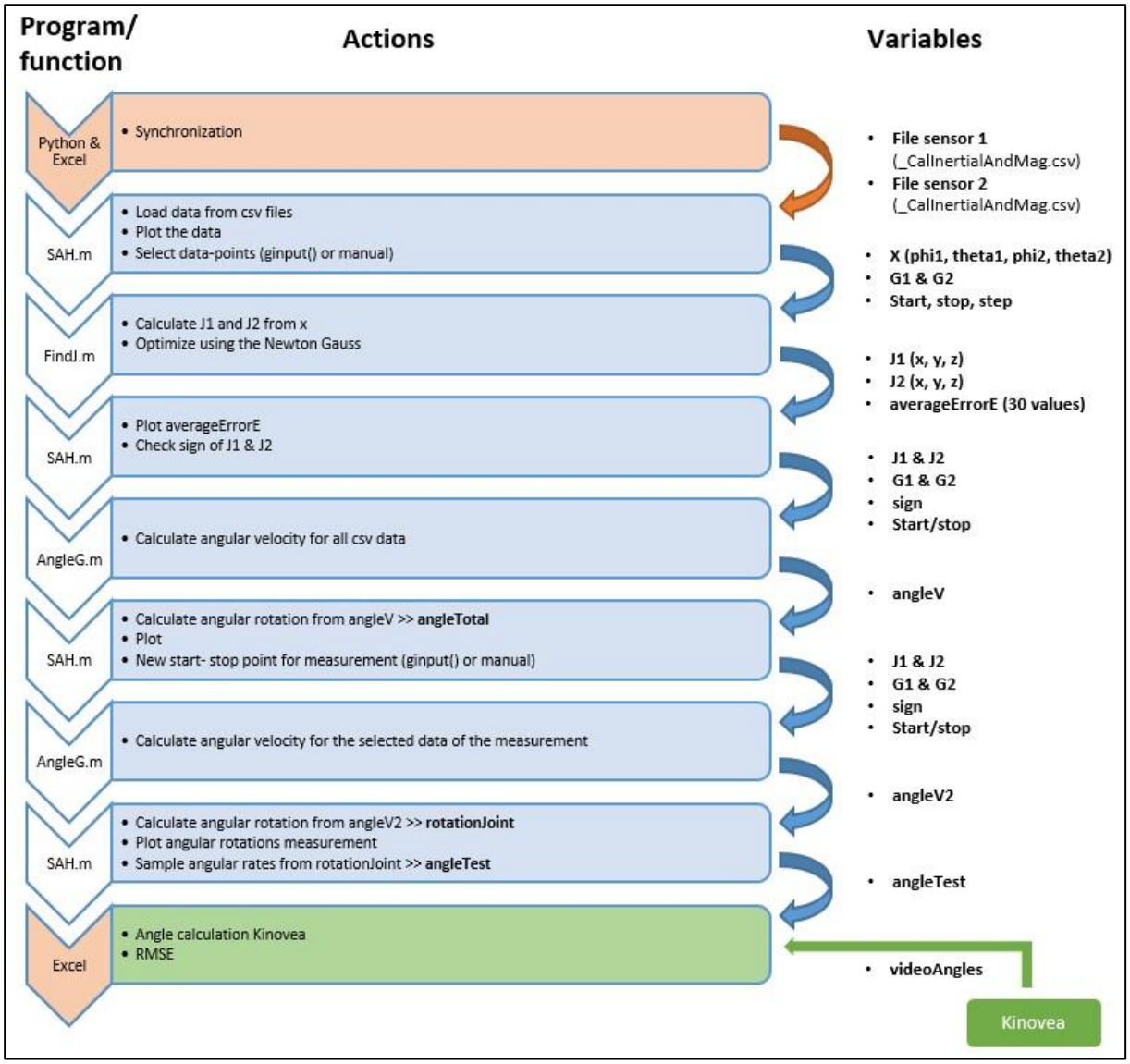

Figure 8, Schematic overview of the Matlab workflow. Blue = Matlab, Green = Kinovea/video, Red = Excel. The data processing starts on top and finished at the bottom of this scheme.

Loading the data

After the data is synchronized (chapter 2.2.3), the edited IMU CSV files are loaded into the workspace, using the original script provided by the HHS (1 … 111) [22]. Rules (27, 28, 35, and 36) contain the file- and folder name, these can be edited by the user if another file should be loaded. After loading the IMU data into two files the data is plotted (51 … 111). During plotting the accelerometer and gyroscope data from the two IMU's, the user is prompted to select two data-points from the graph using the *ginput()* function (124 … 127). The selection is used for finding the direction of the joint axis during the calibration. For repeatability, the start- and stop points can also be manually entered (129 … 131).

Calibration

After the data is selected, it is put into the function 'FindJ.m'. Besides the data-selection, 'FindJ.m' also takes the number of iterations (how many iterations the function should use for optimization) and an initial guess of the four spherical coordinates called 'x' (152-160).

To simplify the explanation of the function, this example will contain 200 data points for g(t).

Inside the FindJ.m function (Appendix 3.2) a first loop (17) runs 30 times (number of iterations given from the main program), during each cycle the initial guess of spherical coordinates are used to calculate the estimate of the direction of the joint axis (25/26).

The estimates of *J1* and *J2* are used in the second loop (32), which runs N times, depending on the amount of data point selected (in this case, N = 200 times). *J1* and *J2* are used to calculate the error (how much the initial estimate differs from the actual joint axis direction) for all .. data-points and given in a column vector called '*e'* (34). In this example, '*e'* has a size of 200*1. In the same loop the Jacobian matrix is created using the gyroscope data x,y,z from both sensors and the spherical coordinates from '*x'* (46). The Jacobian matrix contains now 200*4 data points.

The first loop (17) now continues, creating the pseudoinverse of the Jacobian matrix using the function *pinv()*(49). The function creates a matrix called *'pinvJ'*, which has a size of 4*200. The pseudoinverse now also have the right size to be multiplied by e and that column vector of 4*1 is subtracted from the initial guess of *x* (51). Finally the average error of all 200 data points are taken and added to a column vector called *averageErrorE* (54). This process is repeated 30 times before returning to 'SensorAlignmentHinge.m'.

The function 'FindJ.m' returns the last J1 and J2 as constant variables (1) and are hereafter used to calculate the angular velocities and rotations. The function also returns the column vector '*averageErrorE'* which contains 30*1 mean errors, these are later visualized.

Checking the sign of J

To check if the z-axis of both IMU's point in the same half space after mounting, their sign is checked. To simulate a data point where the angular rotation of the joint ≈ 0, a new variable with low gyroscope rates is created (164) named *gtemp*. This vector is used together with the J's to check their signs using the *sign()* function (165/166).

Angular velocity

The angular velocity of the joint is calculated inside the function 'AngleG.m' (appendix 3.3). The function takes the selected gyroscope data, J1 and J2 and the sign as input. The angular velocity of IMU two is either subtracted or added from/to the angular velocity of IMU1, depending on the sign (9-12 or 13-16). In the main program, the function is run two times and its output is set as variables '*angleV*' (all IMU data) and '*angleV2*' (selected data)

Angular rotation

To get the angular rotation, the angular velocity is integrated using the function *cumtrapz()*, which takes the angular velocity as input. The result of the 'cumtrapz' function is multiplied by the time-step for each data point: $timestep = \frac{1}{128} = 0.00781$ and added to the start angle. '*AngleV'* (181) has a start-angle of 0, and '*AngleV2'* has the start-angle from in the video (208).

The angular rotation is calculated two times: the first time for the whole dataset using *AnlgleV* as input (181), and a second time after selecting the specific data points (206/207) used for the measurement, using *AngleV2* (212). The angular rotation of the whole dataset is used to check if the direction of the rotation is correct, the angle should start at 0 and move into the positive direction. If the angular rotation is not corresponding with the expected results, the sign could be checked.

Angular rotation data for video reference synchronization

To synchronize the IMU data to the video reference, two starting points can be used. Either the accelerometer peak is used in the video to synchronize the IMU data, or a max/min peak from the video's angle is taken. In that case the starting angle will be at a point where there is no rotation measured.

The start-angle used for the second angular rotation will be entered manually (206/207) for the 'angular rotation test' and the 'bike test'. This start-angle is obtained from the video reference. The synchronization point for which point the start-angle should be entered is either a point where there was no angular rotation or a maximum/minimum angle. The variable '*rotationJoint*' contains the final angle of all IMU data-points and corresponds to the measured angle measured using the video.

Datapoints

The video used to compare the angles from the IMU data is shot with 30 fps, the IMU data was collected on a 128 Hz rate. To take the same time-steps in the IMU data corresponding to the video, each 4,27th row needs is selected to synchronize the data (229-231). The variable *angleTest* contains the data-points that can be compared with the angle calculated with Kinovea. Comparing the measured angles and the deviation is done using Excel and is explained in chapter 2.4.2.

2.2.3 Data synchronization

The data from each test that is executed is synchronized using the same method for all the tests. The initial starting point for each measurement will be a point where the IMU's are only translated to create an accelerometer peak. Segment 1 and 2 of the joint-model, or the IMU's cases are hold parallel to each other (figure 9).

Figure 9, Parallel segments of the hinge joint-model before hitting the model to a flat surface, used to create an accelerometer peak.

Creating accelerometer peak: During logging, an accelerometer peak is created by holding the parallel segments of the joint-model or the cases of the two IMU's parallel, vertically in the air before dropping them gently on a flat surface. The sudden stop of the joint-model/IMU's onto the surface will result in a accelerometer peak depending on the direction of the local coordinate system.

Search the peak using Python: After the peak is created, the measurements are executed (chapter 2.4). After each test the logged data is saved at the computer's hard drive and the file '_CalInertialAndMag.csv' are opened using the python script (appendix 4) as described in chapter 2.2.2.2. The accelerometer data is selected in de drop-down-list above each graph and the accelerometer peak is manually searched using the zoom function of the graph (figure 10). The corresponding row-numbers are noted and used for cutting the data-file in excel.

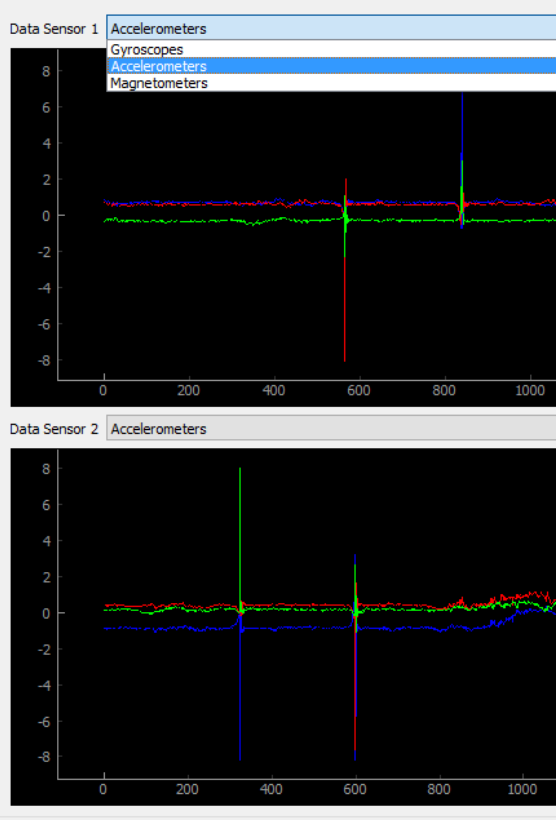

Figure 10, Graphs of original IMU data files (1,2), these graphs show two accelerometer peaks because a drop was executed two times.

Cutting the original data-file: Once the row number of each file containing the accelerometer peak is known, the '_CalInertialAndMag.csv' files of each sensor are opened in Excel and the rows before the accelerometer peak are deleted. The CSV files are thereafter saved and are ready to be used in the Matlab script.

2.2.4 Experiments

As described at the beginning of this chapter, three different kind of tests are performed. The 'calibration test' and 'angular rotation' test are executed using a mechanical hinge joint to test various mounting positions of the IMU's and several estimate values for the spherical coordinates for $\overline{x}$.

2.2.4.1 Calibration test (optimization)

The 'calibration test' tests the amount of necessary iterations for optimizing the joint axis direction, using different situations like: sensor placement, amount of selected data, initial guesses of the spherical coordinates ($\overline{x}$), and the use of different data-points. The angular rotations in these tests are only used for indicating the direction of the rotations and are not compared with video-tracking data.

For the Newton/gauss optimization to be tested, two IMU's, the hinge joint-model with spacers, double sided tape and the iPhone 7 are used to test the optimization progress of the algorithm.

Sensor placement

In total five measurements are executed, using five different sensor placements. The placement of the IMU's are shown in table 2.

Table 2, Overview of the different sensor placement at the five measurements. The mounting deviation of the IMU's with respect to the global CS increases with each following test. Positive can also be notified as lateral and negative as medial.

Measurement number	IMU placement	IMU 1 x-axis	IMU 2 x-axis	IMU 1 z-axis	IMU 2 z-axis
1	Parallel to segments	Distal	Distal	Positive	Positive
2	Parallel to segments	Distal	Distal	Positive	Negative
3	Random, slight rotation	Distal	Mainly distal	Positive	Positive
4	Random, big rotation	Mainly distal	Mainly distal	Positive	Positive
5	Random, big rotation and translation	Mainly distal	Mainly distal	Positive	Positive

Five measurements, seven tests

The five measurements from table 2 are used for seven calibration tests in total. The data from the first four tests are used one time by the Matlab program using the same initial values for $\overline{x}$, but random data selections for optimization. The fifth measurement is run three times; the first time using the same initial values for *x* and random data selections, the second time with different initial values for $\overline{x}$, and a third time using a small selection of data to be used for the optimization. The manual small data selection is entered at rule (129-131) of the Matlab program.

Motions

To generate gyroscope data for finding the joint axis direction during calibration, ten arbitrary motions are generated by holding the joint-model in the air and generate angular rotations around the global the z-axis of the system/joint-model, thereby the system is tilted and turned in free space. After the arbitrary motions, angular rotations are generated.

In the first four measurements, one segment of the joint-model rests parallel to a flat table surface, the second segment is alternately rotated between approximately 180° to 90 degrees (figure 11). In the fifth measurement the generated angular rotations are generated with rotating and tilting both segments of the joint-model in free space.

Figure 11, Generation of angular rotations around the hinge joint by turning one of the two segments.

Hypothesis

In all cases, the direction joint axis for each IMU (J1 & J2) should be found with less than 20 iterations. The amount of iterations used to approach the error ≈ 0, will increase with the higher deviation of the IMU's local CS to the global CS. As in pre-experimental measurements was found, the expected J's are flipped when the z-axes are pointing in the same half space ([0,0,1] and [0,0,-1]). An overview of the hypothesis is found in appendix 5, table 7.

2.2.4.2 Angular rotation test (mechanical joint IMU versus video)

To test the accuracy of the angular rotation on a mechanical hinge joint, the IMU angles are compared with angles that are calculated from video-tracking-data that is obtained by the use of Kinovea and Excel. The hinge joint-model is used to generate angular rotations and the rotations of the measurement are recorded using a iPhone 7plus at 30 fps[23].

Sensor placement

The IMU's are mounted randomly at the joint-model, with both z-axes pointing in the same half space. Markers are placed near the IMU's at the distal end of each segment using sport tape and an Edding marker. The screw of the joint functions as the joint's marker.

Figure 12, Joint board with markers for Kinovea tracking, the joint's screw is used as the third marker.

Motions

After ten arbitrary motions are generated for calibration, angular rotations are generated by rotating both segments of the joint-model. While rotating, both segments of the joint-model are held up in air, but will not be tilted or skewed to ensure that the recording angle does not alter too much (figure 12). In total five rotations are generated for the angle measurement.

Kinovea angles

Kinovea trajectory is used to track the two makers and the joint screw of the joint-model. The output of Kinovea is a txt-file with pixel coordinates of all three markers at each frame. The txt-file is imported in Excel and there the coordinates are converted to lengths. The lengths of each section is used to calculate the angle using the arctan2 function from equation (14), where the x and y coordinates from each segment marker are calculated with the joint-coordinates as starting point.

$$angle = arctan2(\mathrm{x1}, \mathrm{y2}) + arctan2(x2, y2) \quad (14)$$

The angles from the video are set as golden standard and compared to every 4,27th IMU sample by calculating the root mean squared error (RMSE). The RMSE is explained in chapter 2.5.

Hypothesis

Finding the joint axis direction should take less than 20 iterations. The joint's flexion/extension angle from the IMU's gyroscope data is expected to have a RMSE of < 1 degree compared to the predicted video angles.

2.2.4.3 Bike test (human knee IMU versus video)

Sensor placement

The accuracy of the algorithm was higher at mechanical hinge joints. To evaluate if the algorithm is sufficient enough to measure the angular rotation of a knee joint during treadmill cycling, the 'bike test' is performed. A healthy male cyclist (30+ years) rides a Koga Kimera racing bike that is fixated into a Tacx flow trainer. Makers are placed on the Trochanter Major, lateral knee-joint groove, and Malleolus Lateralis of the right leg. The test is recorded using a Sony Cyber-shot DSC-T99 (30 fps).

Figure 13, IMU placement at the bike test. IMU1 is placed lateral/distal on the thigh and IMU2 is placed frontal/proximal on the tibia.

Sensor placement

To minimize the influence of soft tissue movement (skin and muscles), IMU1 is mounted on the lateral distal side of the thigh, using sports tape. IMU2 is mounted on the proximal frontal side of the shank, on top of the tibia using Velcro tape (figure 13). IMU1 was fixated using sports tape because the conical shape of the thigh which caused the Velcro tape to slide.

Motions

The cyclist rides the bike with a maximum cadence of 60 rpm, while the feet are fixated with cycling shoes Shimano SPD-SL pedals. By fixating the feet, lateral and medial rotation of the shank is reduced and thereby the knee will act more likely as a hinge joint. The cyclist pedals the bike for at least five rotations, the pedal strokes of the cyclist are used as arbitrary motions for calibration, and for angular rotation measurement.

Kinovea

The angle measurement in Kinovea uses same method as the angle test using the hinge joint-model. For the data synchronization, the second maximum angle peak of the angles is taken as synchronization point.

Hypothesis

Finding the joint axis direction should take less than 20 iterations. For the flexion/extension angle of the knee, a RMSE of 3° is expected.

2.5 Root mean squared error

The angle from kinovea video tracking is set as golden standard for this work, thereby the angles measured by Kinovea are set as the prediction values. The angles calculated from the IMU's gyroscope data are set as the estimate values. The RMSE is calculated from the predicted kinovea angles and the measured IMU angles with the use of equation (15).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(f_i - o_i)^2}{N}}, f = prediction\ value, o = estimated\ value \quad (15)$$

3. Results

3.1 Calibration test results

During the seven calibration tests, all joint axis directions have been found by the Newton/Gauss optimization within 30 iterations. For all the tests, an average of 196 data-points were used for optimization. The best results with the lowest amount of 11 iterations was found in test 1 where the IMU's were mounted flat on the segments (figure 14). A maximum of 25 iterations was needed in test 7, where only six data-points were used for calibration (figure 15). The iterations from test 7 is seen as outlier and is excluded from the iteration results. Over six tests, an average of 15 iterations for optimization with the use of 228 data-points is within the expectation of < 20 iterations.

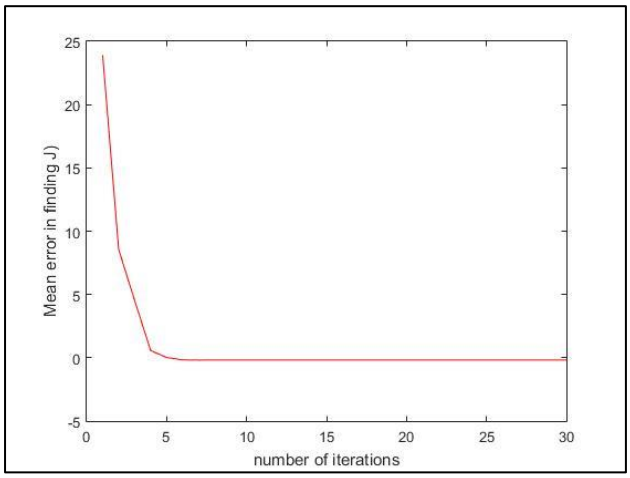

Figure 14, Plot of the mean error in each iteration in Test 1. The optimization runs smooth towards an error of 0.

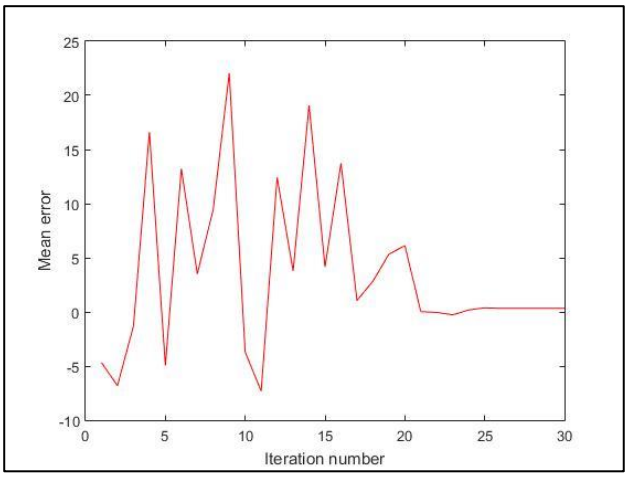

Figure 14, Plot of the mean error in each iteration in Test 6. Only 6 data-point resulted in an unstable optimization.

As explained in the previous paragraph, all values for J^i were found. In test 2, arbitrary motions for calibration were generated by rotating both segments. During this test, the calculated orientation of the z-axis (sign) was as expected, but the angular rotation during rotation of both segments was shown incorrectly (data-point 0 .. 2100 in figure 16). Only when one segment rotated with respect to the other segment, the angular rotation was correct. In other the other six tests the angle was showed correctly (figure 17).

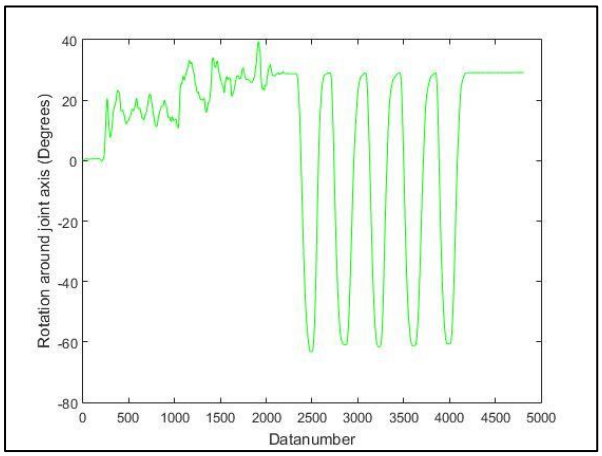

Figure 16, Plot of the angular rotation of the entire dataset of test 4. The angular rotation is only shown correctly when one of the two segments rotates.

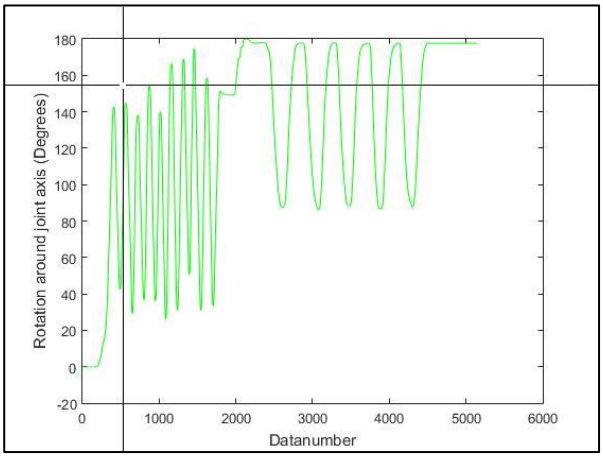

Figure 15, Plot of the angular rotation of the entire dataset of test 3. The angular rotation is shown correctly, also when both segments rotate during the arbitrary motions (data 400..1900)

In test six, choosing different initial values for x than in test five did not resulted a different direction of the joint axis (J1, J2), only three more iterations were needed for optimization. An overview of the results is shown in table 3. All plots, results overview and comprehensive data can be found in appendix 6.1

Table 3, Overview of the Calibration test results. The used data-points for calibration were selected randomly using the ginput() function. The amount of data is the total amount of gyroscope data that is run through the optimization loop 30 times.

Result Calibration								
Testnr	Data used calibration (start, step, stop)	Amount of data	Iterations for J	Sign	Determined J1	Determined J2	Deviation of predicted J from hypothesis	
1	425	205	11	-	-0,040	-0,043	-0,040	-0,043
	5				0,060	-0,047	0,060	-0,047
	1449				0,997	-0,998	-0,003	0,002
2	433	236	13	+	0,012	0,000	0,012	0,000
	5				0,006	0,017	0,006	0,017
	1609				1,000	1,000	0,000	0,000
3	445	255	16	-	0,123	0,123		
	5				-0,099	0,078		
	1719				-0,987	0,989		
4	1478	229	18	-	-0,136	-0,283		
	5				0,102	0,219		
	2622				0,985	-0,934		
5	423	222	15	-	0,255	0,309		
	5				0,167	0,283		
	1530				-0,952	0,908		
6	423	222	18	-	0,255	0,309	0,000	0,000
	5				0,167	0,283	0,000	0,000
	1530				-0,952	0,908	0,000	0,000
7	859	6	25	-	0,235	0,227	-0,020	-0,081
	5				0,359	0,294	0,192	0,011
	886				-0,903	0,928	0,049	0,020

3.2 Angular rotation results (mechanical joint IMU versus video)

In the angular rotation test, the joint axis direction was optimized successfully within 14 iterations. Synchronization between the video and gyroscope data was performed by using the accelerometer peak in the IMU signal and the visual impact of the model in the video. From the IMU angles, every 4,3th sample was taken for angle-comparing. The 392 angles from the video and IMU varied only by an average of 0.18 degrees. The maximum angles differed only by 0.2 degrees, as well as the minimum angle. Visualizing the two flexion/extension angles of the hinge joint also proofs that the process of the angles are similar (figure 18). This is proven by the deviation of the predicted and estimated values. 392 flexion/extension angles were used for the deviation and resulted in a **RMSE** of **1,69 degrees** (table 4), which is a bit higher than the expected <1°. An overview of the used data and comprehensive figures can be found in appendix 6.2.

Table 4, Result overview of the angular rotation test.

Angular rotation test (mechanical)		
	Video	**IMU**
Maximum angle	183,3	183,5
Minimum angle	33,3	33,1
Mean angle	108,6	108,8
Average angle deviation (deg)	0,18	
Angles used for RMSE	392	
Sum squared deviation	1117	
RMSE	1,69	

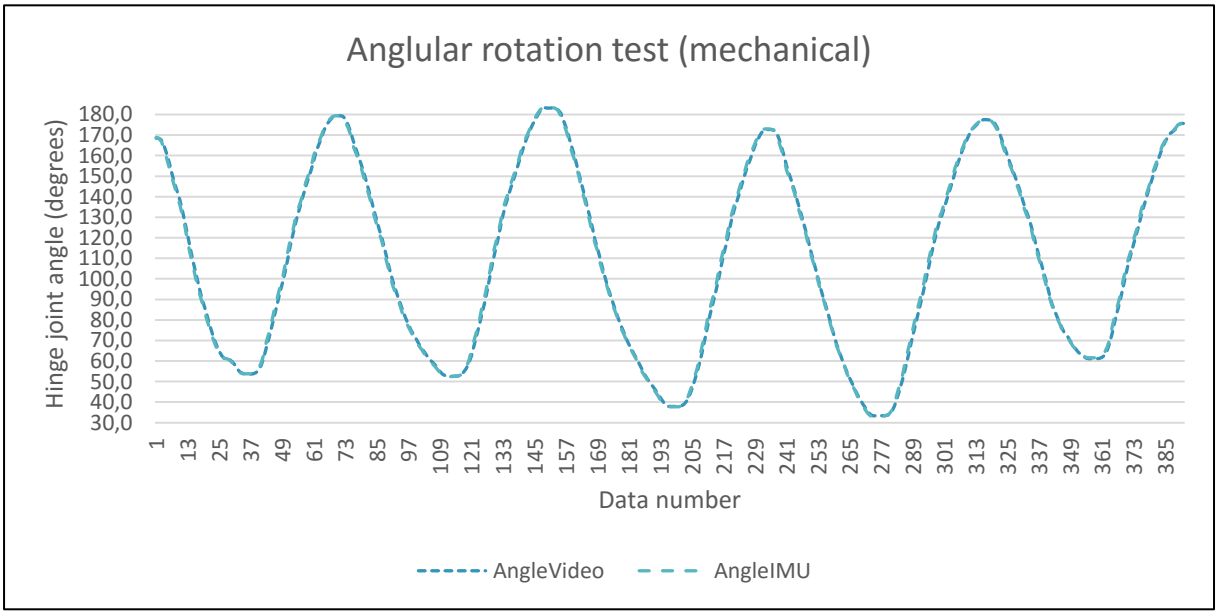

Figure 16, Plot of both the angle from the video as the angle from the IMU's gyroscope data.

3.3 Bike test results (human knee IMU versus video)

In the bike test, 511 angles were determined from the video. From these angles, the first maximum angle peak at video-frame 91 was taken as synchronization point. From the IMU data, the joint axis direction was determined successfully within 17 iterations (table 5). The peak at data-point 9865 was set as synchronization point for the start-angle (155.15°) and after inverting the sign, 1164 angles have been calculated from which every 4,3th sample was taken for comparing to the video angles. In total 272 angles showed an average angle deviation of -3.38 degrees. The maximum angles deviated -0.4 degrees and the minimum -5.3 degrees, which indicates that the total angular rotation of the IMU was higher than rotation from the video, this is clearly shown in figure 19. For the deviation of the predicted and estimated values, 272 flexion/extension angles of the human knee with a total 5033 squared error sum resulted in a RMSE of 4.3 degrees, which is higher than the 4° hypothesis (table 6). An overview of the used data, results and comprehensive figures can be found in appendix 6.3.

Table 5, determined J1 & J2 of the bike test after calibration.

	J1	J2
x	-0,38	-0,02
y	0,29	-1,00
z	0,88	-0,06

Table 6, Result overview of the bike test.

Bike test (human joint)		
	Video	IMU
Maximum angle	157,7	157,3
Minimum angle	81,4	76,1
Mean angle	116,1	112,7
Average angle deviation	-3,38	
Angles used for RMSE	272	
Sum squared deviation	5033	
RMSE	4,30	

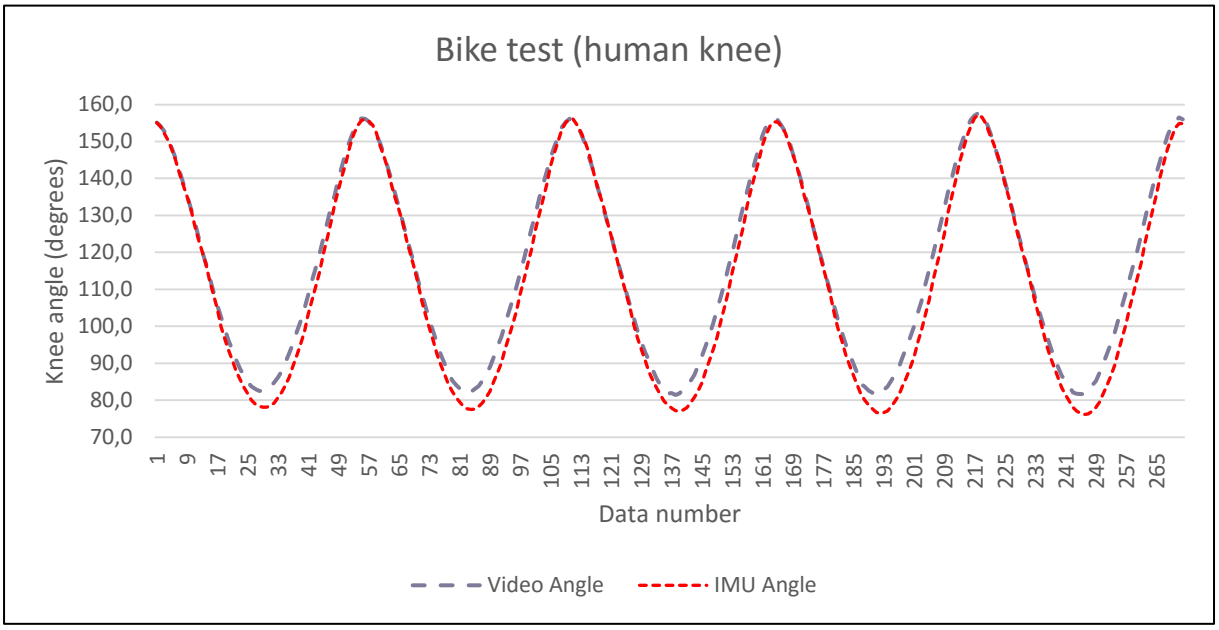

Figure 17, Plot of both the video angle and the IMU's gyroscope angle. A deviation is noticeable at the minimum angles.

4. Discussion

A functional, automated sensor to segment alignment method was taken from the literature and has been further examined in this work. The goal was to implement and test the optimization process of the functional sensor to segment alignment method proposed by Seel et al[12]. The 2D angular joint kinematics estimation method was examined with the use of three different test methods and shows promising results. The partial derivatives for the Jacobian matrix were successfully determined with Matlab. The direction of the joint axis was found at all tests within 20 iterations, from gyroscope data collected using arbitrary motions. Finally angular rotations have been measured accordingly to the expectations and the deviation between the predicted values (video) and the estimated values (IMU) have been found using the root mean squared error (RMSE).

Sign function

During the last two test methods, all flexion/extension angles were calculated correctly compared to the video reference data, measured with Kinovea. However, some problems occurred while processing the IMU data in the second (calibration test) and last (bike test) measurement. During the first run of the IMU's gyroscope data from the bike test, the positive sign of the joint axis direction resulted in a false angle calculation (appendix 6.3, figure 32). After manually invert the sign, the right formula in *angleG.m* (appendix 3.3) was used. So eventually the angles of the bike test were calculated correctly, but after inverting the sign.

RMSE

The deviation between the video and IMU data was successfully determined. Both the RMSE of the mechanical joint and human joint were a bit higher than expected. The RMSE of the mechanical joint was ±0.7° higher and the RMSE of the human joint was ±1.3° higher than predicted. The first explanation for this difference may be the use of Kinovea as golden standard for the predicted joint angle value, which is not as accurate as an optical motion capturing system like optitrack. Other explanations for the higher RMSE, could be the missing Kalman filter used for cancelling sensor drift cancellation, or the presence of white noise. Both the drift and white noise are amplified by numerical integration[15, 18].

Finally a difference between the two joint-types (mechanical vs. human) of 2.61 RMSE was found. This is probably due to the shifting of soft tissue and the fact that the knee is not a true hinge joint and does not function the same as a mechanical hinge joint.

5. Conclusion

The results of this work show that the basic implementation of the optimization algorithm was successful. The Newton/Gauss optimization, number of iterations, axis orientation and calculated angles are in agreement with the results found in the original method. The algorithm shows promising opportunities for in field IMU kinematical research for cycling. Because of the functional approach for the sensor to segment alignment practically no anatomical knowledge is needed to place the sensors. Additionally, the fact that only six data-points were sufficient to find the direction of the joint axis shows that, compared to other methods, the method can be time-saving as well.

However, the method is only suitable for 2D angular kinematical joint estimations of a hinge joint. It is recommended to extend the 2D, to 3D kinematics due to the fact that the body has more spheroidal joints than hinge joints. For future 3D kinematical joint estimations of spheroidal joints, it is recommended to implement the 3D methodology from the same studies of Seel et al.[8,12]. With the mathematical explanation of the Newton/Gauss optimization in this work, the 3D methodology should be implemented easily into a next work. The method also showed better results on a mechanical hinge joint than the human knee, therefore there are some recommendations for future studies concerning this method.

It is recommended to add a Kalman filter or drift cancellation by sensor fusion, and to review the function to automatically search for the mounting orientation of the IMU's z-axis (for 2D kinematics). Reviewing the sign function is in no way obligatory since the sensor's mounting position can always be obtained by the eye.

Summarizing, this work and it's methodology are promising. In particular the functional calibration, but it needs some reviewing before successful implementation in the field of kinematical cycling measurements is possible.

Bibliography

1. Brodie, M., Walmsley, A. & Page, W. Fusion motion capture: a prototype system using inertial measurement units and GPS for the biomechanical analysis of ski racing. *Sport. Technol.* **1,** 17–28 (2008).
2. BioRacer. BioRacer Motion. (2016). Available at: https://bioracermotion.com/?page_id=7#bioracer-motion-3d-system. (Accessed: 20th September 2016)
3. Retul. Retul Vantage 3D Motion Capture System. (2016). Available at: https://www.retul.com/retul-products/vantage-motion-capture-system/. (Accessed: 5th February 2016)
4. Dejnabadi, H., Jolles, B. M. & Aminian, K. A new approach to accurate measurement of uniaxial joint angles based on a combination of accelerometers and gyroscopes. *IEEE Trans. Biomed. Eng.* **52,** 1478–1484 (2005).
5. Mohammadzadeh, F. F., Liu, S., Bond, K. A. & Nam, C. S. Feasibility of a Wearable, Sensor-based Motion Tracking System. *Procedia Manuf.* **3,** 192–199 (2015).
6. O'Donovan, K. J., Kamnik, R., O'Keeffe, D. T. & Lyons, G. M. An inertial and magnetic sensor based technique for joint angle measurement. *J. Biomech.* **40,** 2604–2611 (2007).
7. Yuan, Q. & Chen, I. Sensors and Actuators A : Physical Localization and velocity tracking of human via 3 IMU sensors. *Sensors Actuators A. Phys.* **212,** 25–33 (2014).
8. Seel, T., Raisch, J. & Schauer, T. IMU-based joint angle measurement for gait analysis. *Sensors (Basel).* **14,** 6891–6909 (2014).
9. Djurić-Jovičić, M. D., Jovičić, N. S. & Popović, D. B. Kinematics of gait: New method for angle estimation based on accelerometers. *Sensors* **11,** 10571–10585 (2011).
10. Willemsen, A. T. M., van Alsté, J. A. & Boom, H. B. K. Real-time gait assessment utilizing a new way of accelerometry. *J. Biomech.* **23,** 859–863 (1990).
11. Tong, K. & Granat, M. H. A practical gait analysis system using gyroscopes. *Med. Eng. Phys.* **21,** 87–94 (1999).
12. Seel, T. & Schauer, T. Joint Axis and Position Estimation from Inertial Measurement Data by Exploiting Kinematic Constraints. 0–4 (2012).
13. Picerno, P., Cereatti, A. & Cappozzo, A. Joint kinematics estimate using wearable inertial and magnetic sensing modules. *Gait Posture* **28,** 588–595 (2008).
14. Favre, J., Aissaoui, R., Jolles, B. M., de Guise, J. A. & Aminian, K. Functional calibration procedure for 3D knee joint angle description using inertial sensors. *J. Biomech.* **42,** 2330–2335 (2009).
15. Picerno, P. 25 years of lower limb joint kinematics by using inertial and magnetic sensors: A review of methodological approaches. *Gait Posture* **51,** 239–246 (2017).
16. Findlow, A., Goulermas, J. Y., Nester, C., Howard, D. & Kenney, L. P. J. Predicting lower limb joint kinematics using wearable motion sensors. *Gait Posture* **28,** 120–126 (2008).
17. Cooper, G. *et al.* Inertial sensor-based knee flexion/extension angle estimation. *J. Biomech.* **42,** 2678–2685 (2009).

18. Woodman, O. J. An Introduction to Inertial Navigation. *Univ. Cambridge* 1–37 (2007). doi:10.1017/S0373463300036341

19. Takeda, R. *et al.* Gait analysis using gravitational acceleration measured by wearable sensors. *J. Biomech.* **42,** 223–233 (2009).

20. de Vries, W. H. K., Veeger, H. E. J., Baten, C. T. M. & van der Helm, F. C. T. Magnetic distortion in motion labs, implications for validating inertial magnetic sensors. *Gait Posture* **29,** 535–41 (2009).

21. x-IO technologies. x-IMU GUI V13.1. (2017).

22. Schrauwen, M. ReadXIMU.m. (2013).

23. Apple. Apple iPhone specs website. (2017). Available at: https://www.apple.com/lae/iphone-7/specs/.

Appendix 1, Mathematical method

In this appendix, the mathematical steps of the used algorithm are explained with the necessary equations, just as in chapter 2.1. Only in this appendix contains additional equations, plus references to the code lines where equations are applied in the script from appendix 2 and 3.

1.1 Seel's algorithm

During the calibration, the direction of the joint axis (Z^j) is estimated. After mounting the IMU's at arbitrary positions, the local CS's of the sensors (page 11, figure 5) are not aligned with the segments CS (page 10, figure 4). To denote both local CS's of IMU1 and IMU2 into the global CS, the gyroscope data from each sensor needs to be corrected with a unit vector called J. Each sensor has its own unit joint axis vector, hereafter called $\overline{\hat{J}^1}$ for IMU1 on segment 1, and $\overline{\hat{J}^2}$ for UMU2 on segment 2.

The key to Seel's method, is that it uses the geometrical fact that $\overline{g^1}(t)$ and $\overline{g^2}(t)$, only differ by a rotation matrix (time variant) and the joint angle velocity. Therefore their projections into the joint plane (parallel to the Z^j axis) have the same lengths for each moment in time. This results in equation (1).

$$\left\|\overline{g^1}(t) \times \overline{\hat{J}^1}\right\|_E - \left\|\overline{g^2}(t) \times \overline{\hat{J}^2}\right\|_E = 0 \ \forall t \qquad (1)$$

In (2), E represents the Euclidean norm: $\|\dots\|_E = \sqrt{{x_1}^2 + \cdots + {x_i}^2}$ and gives the projection length. In this system, $\overline{g^i}(i = 1{,}2)$ will be known and the $\overline{\hat{J}^i}(1{,}2)$ are unknown.

1.2 Algorithm implementation

Because $\overline{\hat{J}^i}(1{,}2)$ for the system are unknown, the calibration starts with an initial guess for $\overline{\hat{J}^i}(1{,}2)$ and a selection of gyroscope data. For this example, 50 data-points for $\overline{g^i}(t)$, $(i = 1{,}2,$ $t = 1, \dots, 50)$ are used. Then $\overline{\hat{J}^i}(1{,}2)$ are optimized by a Newton/Gauss method that uses the following steps:

1. **Initial estimate of $\overline{J^i}$ (1,2) using spherical coordinates**
2. **Check the deviation/error of the estimates**
3. **Optimize the estimates using the Newton/Gauss method**
   - **3.1 Determine Jacobian matrix from the partial derivatives of (4)**
   - **3.2 Determine pseudoinverse of Jacobian Matrix**
   - **3.3 Multiply the deviation/error of the estimates by the pseudoinverse of the Jacobian**
   - **3.4 Subtract the previous step from the initial estimate spherical coordinates of $\overline{J^i}$ (1,2)**
4. **Repeat the Newton/Gauss optimization until the deviation in (4) ≈ 0**
5. **Set $\overline{J^i}$ (1,2) as constants for the angular rotation measurement**

1.2.1 Initial estimates of J1 and J2

The vectors $\overline{\hat{f}^i}(1,2)$ are parameterized in spheroidal coordinates in $\overline{x}$ from equation (2), these coordinates are used to define $\overline{\hat{f}^i}(1,2)$ in equation (3).

Spheroidal coordinates

To estimate $\overline{\hat{f}^1}$ and $\overline{\hat{f}^2}$, spherical coordinates are used. Spherical coordinates use two angles (azimuth/phi and zenith/theta), which can define a unit vector in any direction within a three dimensional space (page 11, figure 6). The phi angle represents the inclination angle with respect to the x-y plane. The theta angle represents the rotation around the z-axis. By only using these two angles for the unit vector, its length and rotation around its own axis is irrelevant. In equation (2), a column-vector $\overline{x}$ is filled with a total of four angles, of which the values are used to determine the estimate vectors $\overline{\hat{f}^1}$ and $\overline{\hat{f}^2}$ in equation (3).

$$\overline{x} = (\varphi_1, \theta_1, \varphi_2, \theta_2)^T \tag{2}$$

$$\overline{\hat{f}^i} = |\cos(\varphi_i)\cos(\theta_i), \cos(\varphi_i)\sin(\theta_i), \sin(\varphi_i)|^T, \qquad i = 1,2 \tag{3}$$

Equation (3) can be written in equation (1), this is shown in equation (16).

$$\left\|\overline{g^1}(t) \times \overline{\hat{f}^1}\right\|_E - \left\|\overline{g^2}(t) \times \overline{\hat{f}^2}\right\|_E = \left\| \begin{matrix} g^x \\ g^y(t) \\ g^z \end{matrix} \times \begin{matrix} \cos(\varphi_i)\cos(\theta_i) \\ \cos(\varphi_i)\sin(\theta_i) \\ \sin(\varphi_i) \end{matrix} \right\|_E - \left\| \begin{matrix} g^x \\ g^y(t) \\ g^z \end{matrix} \times \begin{matrix} \cos(\varphi_i)\cos(\theta_i) \\ \cos(\varphi_i)\sin(\theta_i) \\ \sin(\varphi_i) \end{matrix} \right\|_E \tag{16}$$

```
Equation (2 & 3) are used in line(s): 152..156 (appendix 3.1) and  18..26
(appendix 3.2)
```

1.2.2 Check the deviation/error of the estimates using (1)

After the first values for $\overline{\hat{f}^i}(1,2)$ are determined, they are used in equation (1) to create the error vector (4). The error is calculated for all the $\overline{g^i}(i = 1,2)$ data and put in the error vector $\bar{e} = \mathbb{R}^{N\times 1}$. In this example $\bar{e} = \mathbb{R}^{50\times 1}$. The greater the outcome of equation (4), the greater the deviation of what $\overline{\hat{f}^i}$ $(i = 1,2)$ in equation (1) should be. If the estimates of $\overline{\hat{f}^i}$ $(i = 1,2)$ are correct, equation (1 and 4) equal 0.

$$error = \bar{e}(k) = \left\|\overline{g^1}(t) \times \overline{\hat{f}^1}\right\|_E - \left\|\overline{g^2}(t) \times \overline{\hat{f}^2}\right\|_E, k = 1,..,N \tag{4}$$

```
Equation (4) is used in line(s): 34 (appendix 3.2)
```

After the initial estimate of both unit vectors, $\overline{\hat{f}^1}$ and $\overline{\hat{f}^2}$ will be optimized before addressing them as joint axis direction. In equation (4), the gyroscope data is known, but the four initial angles in $\overline{x}$ are guessed. To create an overdetermined system of equations and thereby increasing the accuracy of the estimation, multiple datasets are selected for $(g^1(t),\ g^1(t))^N$, $N \gg 4$ and for each dataset, the error(e) is put in the vector $\overline{e}$ (5). The length of the vector $\overline{e}$ depends on the amount of data points used for the optimization.

$$\overline{e} = R^{N\times 1} = |e^1 \quad \cdots \quad e^N|^T \tag{17}$$

Once the direction of $\overline{\hat{f}^i}$ $(i = 1,2)$, and thereby the direction of the joint axis is found, equation (1) is fulfilled for all time frames.

```
Equation (5) is used in line(s): 32 (loop) & 34 (appendix 3.2)
```

1.2.3 Optimization by the Newton/Gauss method

The Newton/Gauss optimization is used for finding a minimum of nonlinear functions and therefore uses the pseudoinverse of a Jacobian matrix. This Jacobian matrix is in this case a $\mathbb{R}^{N\times 4}$ matrix filled with the partial derivatives of $\frac{de}{dx}$. dx are the four angles used to express $\overline{\hat{f}^i}(1,2)$ in spheroidal coordinates: $\varphi_1, \theta_1, \varphi_2, \theta_2$. The optimization will minimize the error of equation (4) and thereby find the direction of the joint axis.

1.2.3.1 Determine the Jacobian matrix containing partial derivatives of (4)

The partial derivatives for the Jacobian matrix were symbolically derived by the script from appendix 2. The Jacobian matrix filled with the partial derivatives from equation (5 & 6) and has the size of (7), in this example $Jacobian\left(\frac{de}{dx}\right) = \mathbb{R}^{50\times 4}$.

$$e = \left\| \begin{matrix} g^x \\ g^y(t) \\ g^z \end{matrix} \times \begin{matrix} \cos(\varphi_i)\cos(\theta_i) \\ \cos(\varphi_i)\sin(\theta_i) \\ \sin(\varphi_i) \end{matrix} \right\|_E - \left\| \begin{matrix} g^x \\ g^y(t) \\ g^z \end{matrix} \times \begin{matrix} \cos(\varphi_i)\cos(\theta_i) \\ \cos(\varphi_i)\sin(\theta_i) \\ \sin(\varphi_i) \end{matrix} \right\|_E \tag{18}$$

In equation (18), the error equation (4) is written with spheroidal coordinates from $\bar{x}$ (2). Equation (5) shows the partial derivative of (18) with respect to phi, and equation (6) the partial derivative of (18) with respect to theta.

$$\frac{\partial e}{\partial x_{\varphi_i}} =$$

$$\frac{\begin{matrix} 2|g_x^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\cos(\theta_i)|\, sign(g_x^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\cos(\theta_i))(g_x^i\cos(\varphi_i) + g_z^i\cos\theta_i\sin(\varphi_i)) + \\ 2|g_y^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\sin(\theta_i)|\, sign(g_y^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\sin(\theta_i))(g_y^i\cos(\varphi_i) + g_z^i\sin\varphi_i\sin(\theta_i)) - \\ 2|g_y^i\cos(\varphi_i)\cos(\theta_i) - g_x^i\cos(\varphi_i)\sin(\theta_i)|\, sign(g_y^i\cos(\varphi_i)\cos(\theta_i) - g_x^i\cos(\varphi_i)\sin(\theta_i))(g_y^i\cos(\theta_i)\sin(\varphi_i) - g_x^i\sin(\varphi_i)\sin(\theta_i)) \end{matrix}}{\sqrt{2|g_y^i\cos(\varphi_i)\cos(\theta_i) - g_x^i\cos(\varphi_i)\sin(\theta_i)|^2 + 2|g_x^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\cos(\theta_i)|^2 + 2|g_y^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\sin(\theta_i)|^2}},\ i = 1,2 \tag{5}$$

$$\frac{\partial e}{\partial x_{\theta i}} =$$

$$\frac{\begin{matrix} 2|g_y^i\cos(\varphi_i)\cos(\theta_i) - g_x^i\cos(\varphi_i)\sin(\theta_i)|\, sign(g_y^i\cos(\varphi_i)\cos(\theta_i) - g_x^i\cos(\varphi_i)\sin(\theta_i))(g_x^i\cos(\varphi_i)\cos(\theta_i) + g_y^i\cos(\varphi_i)\sin(\theta_i)) - \\ 2g_z^i|g_x^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\cos(\theta_i)|\, sign(g_x^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\sin(\theta_i))\cos(\varphi_i)\sin(\theta_i) + \\ 2g_z^i|g_y^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\sin(\theta_i)|\, sign(g_y^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\sin(\theta_i))\cos(\varphi_i)\cos(\theta_i) \end{matrix}}{\sqrt{2|g_y^i\cos(\varphi_i)\cos(\theta_i) - g_x^i\cos(\varphi_i)\sin(\theta_i)|^2 + 2|g_x^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\cos(\theta_i)|^2 + 2|g_y^i\sin(\varphi_i) - g_z^i\cos(\varphi_i)\sin(\theta_i)|^2}},\ i = 1,2 \tag{6}$$

$$Jacobian\left(\frac{de}{dx}\right) = \left[\frac{de^i}{d\bar{x}}\right] = \begin{bmatrix} \frac{\partial e^1}{\partial x_{\varphi_1}} & \frac{\partial e^1}{\partial x_{\theta_1}} & \frac{\partial e^1}{\partial x_{\varphi_2}} & \frac{\partial e^1}{\partial x_{\theta_2}} \\ \frac{\partial e^2}{\partial x_{\varphi_1}} & \vdots & \ddots & \frac{\partial e^2}{\partial x_{\theta_2}} \\ \frac{\partial e^N}{\partial x_{\varphi_1}} & \cdots & \cdots & \frac{\partial e^N}{\partial x_{\theta_2}} \end{bmatrix},\ i = 1, \ldots, N \tag{7}$$

```
Equation (5, 6 & 7) are used in line(s): 46 (appendix 3.2)
```

1.2.3.2 Determine pseudoinverse of the Jacobian matrix

Next in the Newton/Gauss method is calculating the generalization of the Jacobian's inverse. This Moore-Penrose pseudoinverse of the Jacobian is used to compute an optimal solution to the system of linear equations, in a least square sense. In this case the pseudoinverse is denoted in (8) and is hereafter a $\mathbb{R}^{4\times50}$ matrix.

$$|pinv(Jacobian)| = |Jacobian|^{+},\ \mathbb{R}^{4\times N} \tag{8}$$

Equation (9) is used in line(s): 49 (appendix 3.2)

1.2.3.3 Multiply the pseudoinverse with the error vector

The pseudoinverse is now multiplied by the initial error vector from equation (4). Multiplying the $\mathbb{R}^{4\times50}$ matrix of the Jacobian's pseudoinverse by the $\mathbb{R}^{50\times4}$ matrix of the error vector resuts in a $\mathbb{R}^{4\times1}$ column vector. Which can be subtracted from the initial estimates of $\bar{x}$ ($\mathbb{R}^{4\times1}$) and is shown in equation (9)

1.2.3.4 Update the initial values of x

The last step of the optimization is to update the Theta^{i} and Phi^{i} (i = 1,2) angles in $\overline{x}$ (2), by subtracting the multiplication of the pseudoinverse of the Jacobian (8) and error vector (4) from $\overline{x}$ (2), this is shown in equation (9).

$$\overline{x}^{new} = \overline{x}^{old} - pinv\left(\frac{de}{dx}\right)\cdot\overline{e} \tag{9}$$

Where: $\overline{x} = \mathbb{R}^{4\times1},\ pinv\left(\frac{de}{dx}\right) = \mathbb{R}^{4\times N},\ \overline{e} = \mathbb{R}^{N\times1}$

Equation (10) is used in line(s): 51 (appendix 3.2)

1.2.4 Repeat the optimization

The Newton/Gauss method repeats the steps of 1.2.3 till 1.2.3.4.4 until the mean of the error vector of equation (4) ≈ 0. Each time the optimization is run, the estimates of $\overline{\hat{f}^{i}}$(1,2) are closer to the actual direction of the joint axis.

1.2.5 Set joint axis direction estimate as constant

Once the Gauss/Newton optimization has run for 30 times, the spheroidal coordinates of x should have found the actual direction of the joint axis. The values of $\overline{\hat{f}^{i}}$(1,2) $\mathbb{R}^{3\times1}$ are set as constants and are hereafter used for calculating the joint angle.

1.3 Checking the sign of $\overline{\hat{f}^{i}}$ (1,2)

The angular velocities per IMU are calculated using equation (11). To check if the Z-axes of the IMU's are pointing in the same half space after mounting (page 11, figure 6), the signs of $\overline{\hat{f}^{i}}$ $(i = 1,2)$ can be checked. This is done by taking a data-point in where the angular velocity ($g^{1,2}$(t)) around the joint can be neglected(19). When the signs match, the outcome of equation (10) is positive. This means that both z-axes of the IMU's point into the same half space.

$$\overline{g^{1}}(t)\cdot\overline{\hat{f}^{1}} \approx 0,\ \overline{g^{2}}(t)\cdot\overline{\hat{f}^{2}} \approx 0 \tag{19}$$

$$\overline{g^{temp}} = \begin{bmatrix} 0.001 \\ 0.001 \\ 0.001 \end{bmatrix}$$

$$SIGN = sign(\overline{g^{temp}} \cdot \overline{\hat{J}^1} * sign(\overline{g^{temp}} \cdot \overline{\hat{J}^2}) \quad (10)$$

$$sign(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

Equation (13) is used in line(s): 165 (appendix 3.1)

1.4 Angle calculation

For 2D kinematic joint estimation and the rotation around a hinge joint, gyroscope data only will be sufficient to calculate the angular rotations at the joint center. The angular rates of the gyroscope $\overline{g^i}(t), (i = 1,2)$ and the unit vectors $\overline{\hat{J}^1}$ and $\overline{\hat{J}^2}$ are used to calculate the angular rotations around the hinge joint, in the global CS. The dot product of the vectors $\overline{\hat{J}^i}$ $(1,2)$ with the IMU's gyroscope rates $\overline{g^i}(1,2)$, will denote the rates into the global CS (11).

$$\overline{g^i(t)^{local}} \cdot \overline{\hat{J}^i} = \overline{g^i(t)^{global}}, \; i = 1,2 \quad (11)$$

1.4.1 Total angular velocity

To express the total angular velocity rates of the joint, the dot product between the IMU's angular velocity $\overline{g^i}(t)$ $(i = 1,2)$ and the direction of the joint axis $\overline{\hat{J}^1}$ and $\overline{\hat{J}^2}$ are taken and subtracted from each other. This results in the total angular velocity (ω) of the joint, given in the global CS (12).

$$\omega_{gyr} = \overline{g^1}(t) \cdot \overline{J^1} - \overline{g^2}(t) \cdot \overline{J^2} \quad (12)$$

$$\omega_{gyr} = (g_x^1 * J_x^1 + \; g_y^1 * J_y^1 + g_z^1 * J_z^1) - (g_x^2 * J_x^2 + \; g_y^2 * J_y^2 + g_z^2 * J_z^2) \quad (20)$$

Equation (12) is used in line(s): 15 (appendix 3.3)

1.3.3 Total angular rotations

Now that the angular velocities of the hinge joint are known, the angular rotations are calculated by numerical integration. The angular rotations at the joint axis are calculated by integrating Omega from equation (12) and is shown in equation (13). The equation for applying

$$\alpha_{gyr} = \int_0^t (\omega_{gyr}(t))dt \quad (13)$$

$$\alpha_{gyr} = cumtrapz\big(\omega(t)\big) * \left(\frac{1}{frequency}\right) \quad (21)$$

Equation (21) is used in line(s): 181, 212 & 216 (appendix 3.1)

Appendix 2, Matlab (symbolic Jacobian of de/dx)

This Matlab script was used to symbolically partial derive equation (1), towards the four angles used to initiate the unit vectors J1 and J2.

```
%% compute symbolic jacobian
clc, clear all, close all

%% create symbols for x1, x2, x3, and x4, representing Phi1, Phi2, Theta1, Theta2
x = cell(1,4);                  %create empty variable with 4 cells
for i = 1:4
   x{i,1} = sprintf('x%d%d',i);    %put x1 .. x4 into cells as string
end
x = x(1:4)' ;                   %select first column containing strings
x = sym(x, 'real');             %set strings to symbols

%% create G1 with 3 values, representing the angular velocity in axis x, y, and z of IMU1
G1 = cell(3,1);                 %create empty vector with 3 rows
for i = 1:3
   G1{i,1} = sprintf('G1_%d%d',i); %put 1,2,3 representing x,y,z into cells as string
end
G1 = G1(:);                     %select all columns and rows containing strings
G1 = sym(G1, 'real');           %set strings to symbols

%% create G2 with 3 values, representing the angular velocity in axis x, y, and z of IMU 2
G2 = cell(3,1);                 %create empty vector with 3 rows
for i = 1:3
   G2{i,1} = sprintf('G2_%d%d',i); %put 1,2,3 representing x,y,z into cells as string
end
G2 = G2(:);                     %select all columns and rows containing strings
G2 = sym(G2, 'real');           %set strings to symbols

%% create the function which dertermines J1 and J2
J1 = [cos(x(1))*cos(x(2)); cos(x(1))*sin(x(2)); sin(x(1))];
J2 = [cos(x(3))*cos(x(4)); cos(x(3))*sin(x(4)); sin(x(3))];

%% create the function which takes the G values of the IMU and J1 & J2 to calculte the error
e = norm(cross([G1(1) G1(2) G1(3)],J1))-...  % velocity of segment 1 (thigh) at joint axis
   norm(cross([G2(1) G2(2) G2(3)],J2));     % velocity of segment 2 (shank) at joint axis

%% let Matlab calculate the symbolic Jacobian matrix for de/dx by giving e and x into the function.
JACO = jacobian(e,x);   %use built-in function of matlab to calculate sybolic Jacobian

%% After running the avove script, the symbolic jacobian is determined in JACO
JACO      %This will show the symbolic Jacobian in the command window
```

Appendix 3, Matlab program

The Matlab script used for testing the method and perform data analyses, consist out of the following programs:

Main program:

3.1 SensorAlignmentHinge.m

Functions:

3.2 FindJ.m
3.3 AngleG.m

3.1 SensorAlignmentHinge.m (main)

```
% ReadXIMU (rules 1 – 111)
%
%
%

%   Copyright (C) 2013  M. Schrauwen (mjschrau@hhs.nl)
%
%   This program is free software: you can redistribute it and/or modify
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation, either version 3 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You should have received a copy of the GNU General Public License
%   along with this program.  If not, see <http://www.gnu.org/licenses/>.

%%
close all;
clear;
clc;

%% File data sensor 1
fileName1= 'LoggedDataBikeSb_CalInertialAndMag.csv' ; %type here the name of the CSV file to use
pathName1 = 'C:\Users\Arno\Documents\Metingen\';   %type here the folder
PosUnderscore1 = max(find(fileName1=='_'));   %get the prefix
prefix1 = fileName1(1:PosUnderscore1-1);   %get the letters in front of the underscore
dataLocation1 = [pathName1 prefix1];      %put location + prefix together
data1 = BTDataXIMU(dataLocation1);      %load data

%% File data sensor 2
fileName2 = 'LoggedDataBikeSo_CalInertialAndMag.csv';   %filename(.csv)
pathName2 = 'C:\Users\Arno\Documents\Metingen\';   %folder
PosUnderscore2 = max(find(fileName2=='_'));   %get the prefix
prefix2 = fileName2(1:PosUnderscore2-1);   %get the letters in front of the underscore
dataLocation2 = [pathName2 prefix2];      %samenstellen locatie + prefix
data2 = BTDataXIMU(dataLocation2);      %data ophalen

%% Set data for plotting
L1 = length(data1{2});
xas1 = 0:L1-1;
labelXas1 = 'amount of samples';
L2 = length(data2{2});
xas2 = 0:L2-1;
```

```
48.  labelXas2 = 'amount of samples';
49.
50.  %% PLOTS
51.  aantalSubPlots = 4;
52.
53.  %% Plot Accelleration Data sensor 1
54.  figure('units','normalized','outerposition',[0 0 1 1],'Name', 'X-IMU data1 and data2');
55.  hold on;
56.  subplot(aantalSubPlots,1,1);
57.  plot(xas1, data1{:,2}, 'r');
58.  hold on;
59.  plot(xas1, data1{:,3}, 'g');
60.  hold on;
61.  plot(xas1, data1{:,4}, 'b');
62.  hold on;
63.  ylabel('Acceleration (g)');
64.  title('Accelerometer 1');
65.  legend('X', 'Y', 'Z');
66.  xlabel(labelXas1);
67.  grid on
68.  grid minor
69.
70.  %% Plot Gyroscope data sensor 1
71.  subplot(aantalSubPlots,1,2);
72.  plot(xas1, data1{:,5}, 'r');
73.  hold on;
74.  plot(xas1, data1{:,6}, 'g');
75.  hold on;
76.  plot(xas1, data1{:,7}, 'b');
77.  ylabel('Angular Velocity');
78.  title('Gyroscope 1');
79.  legend('X', 'Y', 'Z');
80.  xlabel(labelXas1);
81.  grid on
82.  grid minor
83.
84.  %% Plot Accellerometer data sensor 2
85.  subplot(aantalSubPlots,1,3);
86.  plot(xas2, data2{:,2}, 'r');
87.  hold on;
88.  plot(xas2, data2{:,3}, 'g');
89.  hold on;
90.  plot(xas2, data2{:,4}, 'b');
91.  hold on;
92.  ylabel('Acceleration (g)');
93.  title('Accelerometer 2');
94.  legend('X', 'Y', 'Z');
95.  xlabel(labelXas2);
96.  grid on
97.  grid minor
98.
99.  %% Plot Gyroscope data sensor 2
100. subplot(aantalSubPlots,1,4);
101. plot(xas2, data2{:,5}, 'r');
102. hold on;
103. plot(xas2, data2{:,6}, 'g');
104. hold on;
105. plot(xas2, data2{:,7}, 'b');
106. ylabel('Angular Velocity');
107. title('Gyroscope 2');
108. legend('X', 'Y', 'Z');
109. xlabel(labelXas2);
110. grid on
111. grid minor
112.
```

```
%%
%     From this point on, the code is written by Arno van der Zwet,
%     graduate student of Human Kinetic Technology. The code is used to
%     select data and calculate angular velocity and rotations between
%     two IMU's that are mounted onto two segments, connected by a hinge
%     joint. This code will optimize the direction of the joint axis
%     between the two segemtns, given in each seperate sensor.


%% get ginput values for calibration, use 1 for selecting in plot and 2 for manual point
%% 1 data-points from plot
selection = ginput(2);
start = round(selection(1,1))
step = 5;
stop = round(selection(2,1))
%% 2 manual data-points
%start = 9952;   % start calibration at this point
%step = 5;       % take every 5th datapoint
%stop = 11091;   % end calibration at this point

%% get the gyroscope data and put these into new arrays
dataG1x = data1{:, 5};
dataG1y = data1{:, 6};
dataG1z = data1{:, 7};
dataG2x = data2{:, 5};
dataG2y = data2{:, 6};
dataG2z = data2{:, 7};

minimum = min(length(dataG1x), length(dataG2x));    %search for sensor with lowest amount of data and set it as minimum
G1 = [];          %create new vector for gyroscope data of sensor 1
G2 = [];          %create new vector for gyroscope data of sensor 2

for i=1:minimum-1   %use minumum to get G1 and G2 even amount of data
    G1 = [G1; dataG1x(i), dataG1y(i), dataG1z(i)];
    G2 = [G2; dataG2x(i), dataG2y(i), dataG2z(i)];
end

% create initial X to start the calibration with, x contans four angles
% that are used to create spherical coordinates
x1 = 0.4;      % Phi 1, initial guess
x2 = 0.2;      % Theta 1, initial guess
x3 = 0.1;      % Phi 2, initial guess
x4 = 0.7;      % Theta 2, initial guess
x = transpose([x1,x2,x3,x4]);   %create column vector for x

iterations = 30;    %set number of iterations (loops to optimize de/dx)

% run the function to find J1 and J2 and return: mean errors(30), J1 and J2:
[J1, J2, averageErrorE] = FindJ(x, G1, G2, start, step, stop, iterations );

%% check sign to see if Z axis of IMU are pointing in same halfspace
tempG = [0.001;0.001;0.001];  %take/make gyroscope data where rotation is near to 0
sgn = dot(tempG,J1) * dot(tempG,J2) %Check if J's are in the same halfspace
SIGN = sign(sgn); %check sign of sgn

%%  plot mean error values over 30 iterations
xAsJ = 1:1:length(averageErrorE);
figure('name', 'Average optimisation error to number of iterations')
plot(xAsJ, averageErrorE, 'r');
hold on
ylabel('Mean error');
xlabel('Iteration number');

%% calculate angle velocity with g1, g2, j1, and j2
stopper = length(G1)-1; %use length of G1 data to make sure the angle function stops in time
```

```
[angleV] = AngleG(J1, J2, G1, G2, SIGN, 1, stopper);    %run AngleG function, return velocity

%% Calculate angular rotation from angular velocity
angleTotal = (cumtrapz(angleV)*0.0078125); %Take the derivative of the angular velocity with cumtrapz, 0.0078 is the timestep

%% Check the direction of the calculated rotation
directioncheck = mean(angleTotal) %the mean angle should be posive
if directioncheck < 0  %if mean angle is negative
    angleTotal = angleTotal*-1;   %invert the rotations by multiplication by -1
end

%% plot angles in rad and degrees
xAsAngle = 1:length(angleV); %set x-axis values
figure('name', 'Angular rotation of the whole dataset')
plot(xAsAngle, angleTotal, 'g')
hold on
ylabel('Rotation around joint axis (Degrees)');
xlabel('Datanumber');

%% This part can be used for manual selecting the start and stoppoint of the data
%movement = ginput(2); %select two point in the plot for rotation calculation
%startpoint = round(movement(1,1))  %select start from plot
%stoppoint = round(movement(2,1))   %select end from plot
%promt = 'fill in the starting angle: ';    %promt will show in command window
%startAngle = input(promt); %user defines startangle (moslty 180 degrees)

%% The folowing lines are used for manually give the start and end points of the tests
%here, the startpoint can precicely be given for aligning the data to the video timeframe
startpoint = 9865  % manually give startpoint for angle calculation
stoppoint = 11028  %manually give stoppoint for angle calculation
startAngle = 155.1515596; %give in starting angle (0, 180, or angle from videoanalisys)

%% calcuate angle velocity and anglular rotation for the new selection
[angleV2] = AngleG(J1, J2, G1, G2, SIGN, startpoint, stoppoint);
rotationJoint = (cumtrapz(angleV2)*0.0078125+startAngle); %use cumtrapz to transist from velocity to angular rotation (0,00718.. is the time inter
val)

directioncheck2 = mean(rotationJoint) %check the average angle
if directioncheck2 > 160 %if average is above 160 rotation should be inverted
    rotationJoint = (cumtrapz(angleV2*-1)*0.0078125+startAngle);
end

%% plot angles in degrees
xAsAngle = 1:length(rotationJoint); %set x-axis length for plot
figure('name', 'angle selected data')
plot(xAsAngle, rotationJoint, 'g')
hold on
ylabel('Rotation around joint axis (Degrees)');
xlabel('Datanumber');

%% Select datapoint for video comparing
angleTest = [];   %create a new variable for datapoints to be used for RMSE
for i=1:4.266667:(length(rotationJoint)) %take every 4.267th sample from angleG4  (128hz / 30 fps)
angleTest = [angleTest ; rotationJoint(round(i),:)];  %copy all datapoints with rounded i
end
```

3.2 FindJ.m

```
function [ J1, J2, averageErrorE ] = FindJ( x,G1,G2, start, step, stop, iterations )

%FindJ Summary of this function goes here
%  This function is used to find the direction of the joint axis in each
%  seperate sensor, given by J1 and J2. In the first loop the inital
%  angles from the main program are used, these coornates are optimized in
%  this function.
%  The input of the function are:
%       - initial guess for x
%       - Gyroscope data from sensor 1 and sensor 2
%       - Selected datanumbers to use for calibration (start/step/stop)
%       - Number of iterations

iter = 0;   %set number of iterations to 0
averageErrorE = [];  %make a new variable to fill with average error

%% start if the optimization
for i=1:iterations     %run loop untill number of iterations is reached (30)
   phi1    = x(1,1);   %get angle Phi(1) from x
   theta1  = x(2,1);   %get angle Theta(1) from x
   phi2    = x(3,1);   %get angle Phi(2) from x
   theta2  = x(4,1);   %get angle Theta(2) from x

   %Use Phi(1,2) and Theta (1,2) to calculate the direction
   %of the joint axes (J1, J2) in spherical coordinates
   J1 = transpose([cos(phi1).*cos(theta1),cos(phi1).*sin(theta1),sin(phi1)]);
   J2 = transpose([cos(phi2).*cos(theta2),cos(phi2).*sin(theta2),sin(phi2)]);

   e = []; %create empty variable e to fill with error values
   Jacobian = [];  %create variable to fill with jacobian matrix

   %%calculate error in this loop:
   for t=start:step:stop   %start and stop are selected data for calibration
      %calculate the error of J1 and J2 in each timestep
      e = [e; (norm(cross(G1(t, :),J1))-norm(cross(G2(t, :),J2)))];

      %from sensor 1:
      g1x = G1(t,1);  %get gyroscope x data at time t
      g1y = G1(t,2);  %get gyroscope y data at time t
      g1z = G1(t,3);  %get gyroscope z data at time t
      %from sensor 2:
      g2x = G2(t,1);  %get gyroscope x data at time t
      g2y = G2(t,2);  %get gyroscope y data at time t
      g2z = G2(t,3);  %get gyroscope z data at time t

      %fill the jacobian matrix with the 4 values calculated with the jacobian equation:
      Jacobian = [Jacobian; (2*abs(g1x*sin(phi1) - g1z*cos(phi1)*cos(theta1))*sign(g1x*sin(phi1) - g1z*cos(phi1)*cos(theta1))*(g
1x*cos(phi1) + g1z*cos(theta1)*sin(phi1)) + 2*abs(g1y*sin(phi1) - g1z*cos(phi1)*sin(theta1))*sign(g1y*sin(phi1) - g1z*cos(phi1)
*sin(theta1))*(g1y*cos(phi1) + g1z*sin(phi1)*sin(theta1)) - 2*abs(g1y*cos(phi1)*cos(theta1) - g1x*cos(phi1)*sin(theta1))*sign(g
1y*cos(phi1)*cos(theta1) - g1x*cos(phi1)*sin(theta1))*(g1y*cos(theta1)*sin(phi1) - g1x*sin(phi1)*sin(theta1)))/(2*(abs(g1y*cos
(phi1)*cos(theta1) - g1x*cos(phi1)*sin(theta1))^2 + abs(g1x*sin(phi1) - g1z*cos(phi1)*cos(theta1))^2 + abs(g1y*sin(phi1) - g1z*
cos(phi1)*sin(theta1))^2)^(1/2)), -
(2*abs(g1y*cos(phi1)*cos(theta1) - g1x*cos(phi1)*sin(theta1))*sign(g1y*cos(phi1)*cos(theta1) - g1x*cos(phi1)*sin(theta1))*(g1
x*cos(phi1)*cos(theta1) + g1y*cos(phi1)*sin(theta1)) - 2*g1z*abs(g1x*sin(phi1) - g1z*cos(phi1)*cos(theta1))*sign(g1x*sin(phi1)
 - g1z*cos(phi1)*cos(theta1))*cos(phi1)*sin(theta1) + 2*g1z*abs(g1y*sin(phi1) - g1z*cos(phi1)*sin(theta1))*sign(g1y*sin(phi1) -
g1z*cos(phi1)*sin(theta1))*cos(phi1)*cos(theta1))/(2*(abs(g1y*cos(phi1)*cos(theta1) - g1x*cos(phi1)*sin(theta1))^2 + abs(g1x*
sin(phi1) - g1z*cos(phi1)*cos(theta1))^2 + abs(g1y*sin(phi1) - g1z*cos(phi1)*sin(theta1))^2)^(1/2)), -
(2*abs(g2x*sin(phi2) - g2z*cos(phi2)*cos(theta2))*sign(g2x*sin(phi2) - g2z*cos(phi2)*cos(theta2))*(g2x*cos(phi2) + g2z*cos(the
ta2)*sin(phi2)) + 2*abs(g2y*sin(phi2) - g2z*cos(phi2)*sin(theta2))*sign(g2y*sin(phi2) - g2z*cos(phi2)*sin(theta2))*(g2y*cos(phi
2) + g2z*sin(phi2)*sin(theta2)) - 2*abs(g2y*cos(phi2)*cos(theta2) - g2x*cos(phi2)*sin(theta2))*sign(g2y*cos(phi2)*cos(theta2) -
 g2x*cos(phi2)*sin(theta2))*(g2y*cos(theta2)*sin(phi2) - g2x*sin(phi2)*sin(theta2)))/(2*(abs(g2y*cos(phi2)*cos(theta2) - g2x*c
os(phi2)*sin(theta2))^2 + abs(g2x*sin(phi2) - g2z*cos(phi2)*cos(theta2))^2 + abs(g2y*sin(phi2) - g2z*cos(phi2)*sin(theta2))^2)^
```

```
    (1/2)), (2*abs(g2y*cos(phi2)*cos(theta2) - g2x*cos(phi2)*sin(theta2))*sign(g2y*cos(phi2)*cos(theta2) - g2x*cos(phi2)*sin(theta
    2))*(g2x*cos(phi2)*cos(theta2) + g2y*cos(phi2)*sin(theta2)) - 2*g2z*abs(g2x*sin(phi2) - g2z*cos(phi2)*cos(theta2))*sign(g2x*si
    n(phi2) - g2z*cos(phi2)*cos(theta2))*cos(phi2)*sin(theta2) + 2*g2z*abs(g2y*sin(phi2) - g2z*cos(phi2)*sin(theta2))*sign(g2y*sin
    (phi2) - g2z*cos(phi2)*sin(theta2))*cos(phi2)*cos(theta2))/(2*(abs(g2y*cos(phi2)*cos(theta2) - g2x*cos(phi2)*sin(theta2))^2 +
    abs(g2x*sin(phi2) - g2z*cos(phi2)*cos(theta2))^2 + abs(g2y*sin(phi2) - g2z*cos(phi2)*sin(theta2))^2)^(1/2))];
46.     end
47.     %% get the pseudoinverse of the jacobian matrix
48.     pinvJ = pinv(Jacobian);
49.     % update x with the pseudoinverse of the Jacobian Matrix * errors
50.     x = x - pinvJ*e;
51.
52.     %% the following code is additional to the optimisation
53.     averageErrorE = [averageErrorE; mean(e)]; %fill the error-vector with 30* the mean average of the selected data
54.     iter = iter+1;  %add 1 to the amount of iterations
55. end
```

3.3 AngleG.m

```
function [ AngleG ] = AngleG( J1, J2, G1, G2, SIGN, start, eind )
%AngleG is used to calculate the angular velocity around the joint axis.
%   The velocity is calculated by, adding up/subtracting (depending on the
%   sign) the dot product between the gyroscope data (xyz) and the corresponding
%   J (xyz) to each sensor.

AngleG = []; % first create empty variable to put the data in

if SIGN <0   %if sign is negative, add up the angular velocities
    for i=start:1:eind
        AngleG = [AngleG; (dot(J1,G1(i,:)'))+(dot(J2,G2(i,:)'))];
    end
else       %if sign is positive, subtract the angular velocities
    for i=start:1:eind
        AngleG = [AngleG; (dot(J1,G1(i,:)'))-(dot(J2,G2(i,:)'))];
    end
end
```

Appendix 4, Python code

4.1 Introduction

Python is an open-source programming language with many mathematical functionalities and comes with several modules pre-installed for mathematical functions. For the thesis (main.py) program, the modules “PyQt5”, “numpy”, “pyqtgraph”, “scipy” and “math” are installed to create a user interface (UI) with graphics and handling the data, arrays, and vectors. Adding modules, open source development of the python language, and the fact that it runs on all platforms (Linux, Microsoft Windows and Apple OS) make it more versatile than Matlab.

The program runs from the main.py and it will open a user interface, created with PyQt5. It runs with five files in total:

4.2 **Main.py** (main program, runs all the functions and starts the user UI)

4.3 **ThesisApp.py** (contains the code for the UI and runs some basic tasks of the UI)
4.4 **Data.py** (opens the csv files and runs several tasks to prepare them for processing)
4.5 **Calibrate.py** (optimizes the initial estimate of the joint axis direction)
4.6 **Angle.py** (calculates the angle)

The other python files with their functions are loaded into the main.py file, importing them the same way as the modules using the following command: **import** *modulename*

The algorithm for optimizing the joint axis direction works the same as the Matlab script, therefore this is not further explained. The total code and comments are given further in this appendix.

Executable

The python program from this appendix is converted into an executable (main.exe) file using cx_freeze. Users of the program do not need to install python on their computers.

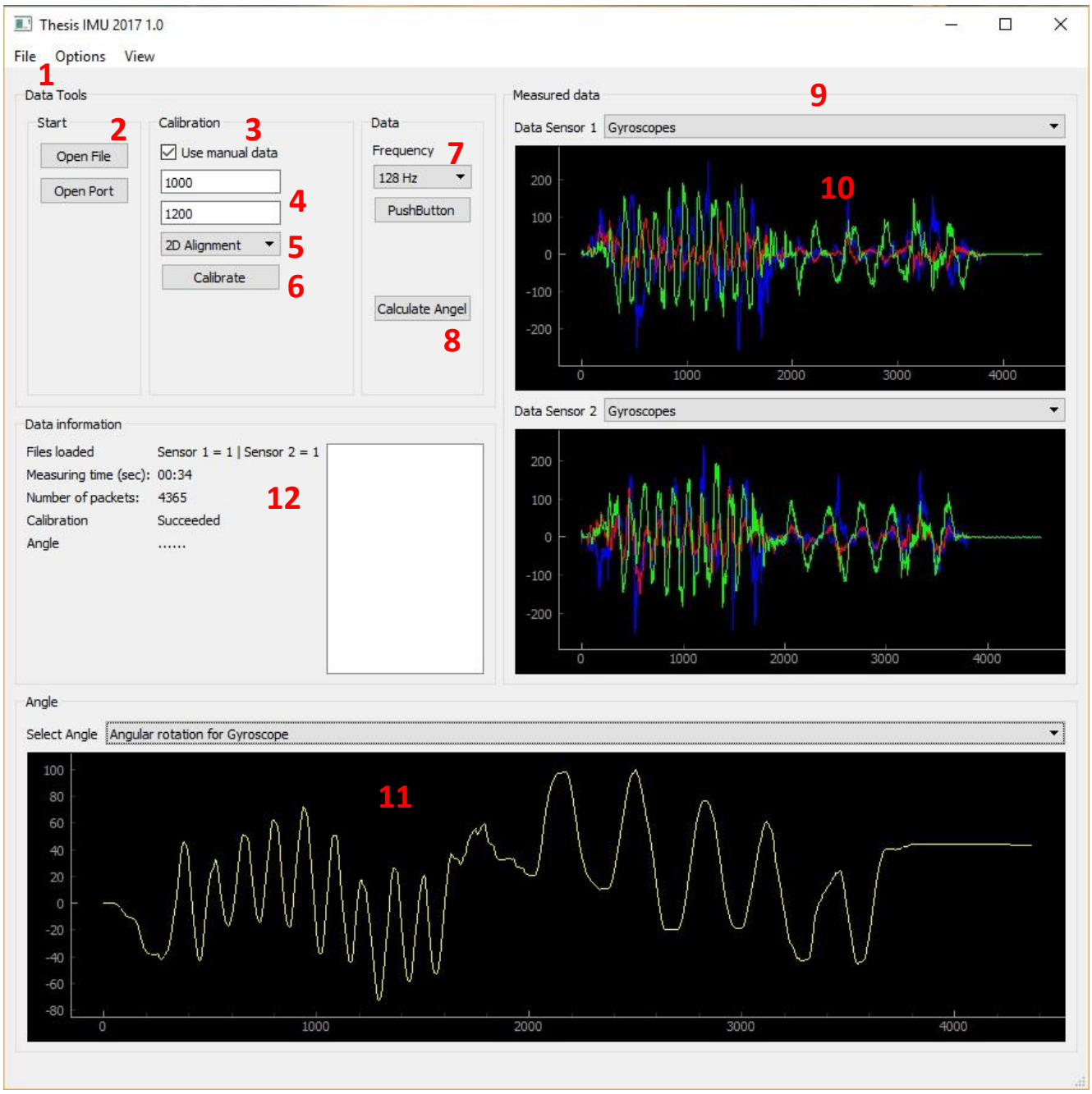

Figure 18, Python user interface, created with the PyQt5 module. The numbers can be found in with the legend below this picture

Legend:

1. Menubar
2. Open file button
3. Use manual data-point for calibration
4. Start/stop number for calibration
5. Select 2D/3D kinematics
6. Run calibration
7. Frequency selector
8. Pushbutton to start angle calculation
9. Data selector for sensor graph
10. Graph sensor data
11. Graph angular rotation/velocity
12. File/data information

4.2 Main.py

```
__author__ = "Arno van der Zwet"
__copyright__ = "Copyright 2017, Thesis Human Kinetic Technology"
__credits__ = ["Arno van der Zwet"]
__license__ = "GPL"
__version__ = "1.0"
__maintainer__ = "Arno van der Zwet"
__email__ = "arnovanderzwet@gmail.com"
__status__ = "First experimental version"
__title__ = 'Thesis Human Kinetic Technology'

from PyQt5 import QtWidgets, QtCore
import sys
import time
import numpy as np
import ThesisApp
import Data as data
import Calibration as calb
import Angle as angle


class MainWindow(QtWidgets.QMainWindow, ThesisApp.Ui_MainWindow):

    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)
        self.setupUi(self)
        # set shortcuts for fast actions in the program
        self.actionQuit.setShortcut("Ctrl+Q")                   # Ctrl+q = Close Window
        self.actionOpen_File.setShortcut("Ctrl+O")              # Ctrl+O = Open Files
        self.actionSave.setShortcut("Ctrl+S")                   # Ctrl+S = Save File

        # set here which function or class should run when the action in the window is performed.
        self.openFileButton.clicked.connect(self.selectFiles)          # selectFiles when PushButtons = clicked
        self.actionOpen_File.triggered.connect(self.selectFiles)       # run openFiles when open file is selected in menubar
        self.actionSave.triggered.connect(self.saveFiles)
        self.calibrateButton.clicked.connect(self.calibrate)           # run 'calibrate' whem calibrateButton = clicked
        self.angleButton.clicked.connect(self.angleCalculation)         # run angleCalculate when angleButton = clicked
        self.openPortButton.clicked.connect(self.port)
        self.pushButton.clicked.connect(self.angleCalculation)

        # set initial value to the amount of file selected (0)
        self.amount_sensor1Files = 0
        self.amount_sensor2Files = 0
        self.setGraph1Show = 1
        self.setGraph2Show = 1
        self.setGraph3Show = 1
        self.frequencyComboBox.setCurrentIndex(7)   # at inintialze, set the standard frequency to 128 Hz (value 7)

        '''''every tenth of a second a dataset must be taken:'''
        self.setDataJump = self.setFrequency/10
        self.dt = 3                          # set here the timestep for calibration, every 10th sec is best

    ''''' Here the files are opened, but first is checked to make sure there are no files used already'''
    def selectFiles(self):
        self.reload1 = False
        self.reload2 = False
        ''''' files from sensor 1 are selected here'''
```

```
        if self.amount_sensor1Files > 0:
            rest = QtWidgets.QMessageBox.question(self, 'Overwriting Files', "You're about to load new files for IMU 1, are you sure?",
                                                  QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No)
            if rest == QtWidgets.QMessageBox.Yes:
                self.reload1 = True
                self.amount_sensor1Files = 0
                self.sensor1FilesSelect = QtWidgets.QFileDialog.getOpenFileNames(self, 'Open files sensor 1',
                                                                                 "C://",
                                                                                 'CSV Files (*.csv)')
                self.sensor1Files = []
                self.sensor1Files = dict()
                for i in range(0, len(list(self.sensor1FilesSelect)[0])):
                    self.sensor1Files[i] = list(self.sensor1FilesSelect)[0][i]
                self.amount_sensor1Files = len(self.sensor1Files)
                if self.amount_sensor1Files < 1:
                    print('no files for sensor 1 loaded')
            else:
                self.reloadCancel1 = False
        '''''If there are no files selected yet, this part is run for sensor 1'''
        if self.amount_sensor1Files == 0:
            self.reloadCancel1 = False
            self.sensor1FilesSelect = QtWidgets.QFileDialog.getOpenFileNames(self, 'Open files sensor 1',
                                                                             "C://",
                                                                             'CSV Files (*.csv)')
            self.sensor1Files = dict()
            ''''' Check if there is a file selected, if non, pass'''
            self.emptycheck1 = not all(self.sensor1FilesSelect)
            if self.emptycheck1 == True:
                pass
            for i in range(0, len(list(self.sensor1FilesSelect)[0])):
                self.sensor1Files[i] = list(self.sensor1FilesSelect)[0][i]
                self.amount_sensor1Files = len(self.sensor1Files)
            if self.amount_sensor1Files < 1:
                print('no files for sensor 1 loaded')

        ''''' files from sensor 2 are selected here'''
        if self.amount_sensor2Files > 0:
            rest = QtWidgets.QMessageBox.question(self, 'Overwriting Files',
                                                  "You're about to load new files for IMU 2, are you sure?",
                                                  QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No)
            if rest == QtWidgets.QMessageBox.Yes:
                self.reload2 = True
                self.amount_sensor2Files = 0
                self.sensor2FilesSelect = QtWidgets.QFileDialog.getOpenFileNames(self, 'Open files sensor 2',
                                                                                 "C://",
                                                                                 'CSV Files (*.csv)')
                self.sensor2Files = 0
                self.sensor2Files = dict()
                for i in range(0, len(list(self.sensor2FilesSelect)[0])):
                    self.sensor2Files[i] = list(self.sensor2FilesSelect)[0][i]
                    self.amount_sensor2Files = len(self.sensor2Files)
                    if self.amount_sensor1Files < 1:
                        print('no files for sensor 2 loaded')
            else:
                self.reloadCancel2 = True

        '''''If there are no files selected yet, this part is run for sensor 2'''
        if self.amount_sensor2Files == 0:
            self.reloadCancel2 = False
            self.sensor2FilesSelect = QtWidgets.QFileDialog.getOpenFileNames(self, 'Open files sensor 2',
                                                                             "C://",
```

```
                                                                    'CSV Files (*.csv)')
            self.sensor2Files = dict()
            ''''' Check if there is a file selected, if non, pass'''
            self.emptycheck2 = not all(self.sensor2FilesSelect)
            if self.emptycheck2 == True:
                pass
            for i in range(0, len(list(self.sensor2FilesSelect)[0])):
                self.sensor2Files[i] = list(self.sensor2FilesSelect)[0][i]
                self.amount_sensor2Files = len(self.sensor2Files)
            if self.amount_sensor2Files < 1:
                print('no files for sensor 2 loaded')

        ''''' Check if the files are loaded and set the amount of files to the amountFiles label'''
        if self.emptycheck1 == False:
            if self.amount_sensor1Files > 0:
                if self.amount_sensor2Files > 0:
                    self.amountFiles.setText(str('Sensor 1 = ') + str(self.amount_sensor1Files) + str(' | ')
    + str('Sensor 2 = ') + str(self.amount_sensor2Files))

        if self.amount_sensor1Files and self.amount_sensor2Files == 0:
            self.amountFiles.setText('No files loaded')

        if self.amount_sensor1Files > 0:
            '''''because genfromTXT in ManualData.py cannot read from dict types,
            the values are taken and put into a list, the ID's are not used'''
            self.sensor1FilesList = list(self.sensor1Files.values())
            ''''' Make use of the selected data'''
            self.sensor1 = data.LoadFiles(self.sensor1FilesList)
        if self.amount_sensor2Files > 0:
            self.sensor2FilesList = list(self.sensor2Files.values())
            self.sensor2 = data.LoadFiles(self.sensor2FilesList)

        if self.amount_sensor1Files > 0:
            ''''' Let the lables display the right values'''
            self.amountTime.setText('time is calculated here')
            self.amountPackets.setText(str(self.sensor1.amount_packets))
            self.time = self.sensor1.amount_packets / self.setFrequency
            self.amountTime.setText(str(time.strftime("%M:%S", time.gmtime(self.time))))
        else:
            if self.amount_sensor2Files > 0:
                self.amountTime.setText('time is calculated here')
                self.amountPackets.setText(str(self.sensor2.amount_packets))
                self.time = self.sensor2.amount_packets / self.setFrequency
                self.amountTime.setText(str(time.strftime("%M:%S", time.gmtime(self.time))))
            else:
                pass

        ''''' Set the selected files into the comboxes above graph1 to select'''
        self.selectDataFileShow1.clear() #clear the combobox first before adding items
        if self.amount_sensor1Files > 0:
            sensor1files = str(self.sensor1FilesList)
            if "Mag" in sensor1files:
                self.selectDataFileShow1.addItem("Gyroscopes", QtCore.QVariant(1))
                self.selectDataFileShow1.addItem("Accelerometers", QtCore.QVariant(2))
                self.selectDataFileShow1.addItem("Magnetometers", QtCore.QVariant(3))
            if "Euler" in sensor1files:
                self.selectDataFileShow1.addItem("Euler", QtCore.QVariant(4))
            self.runGraphics1()
        else:
            pass

        ''''' Set the selected files into the comboxes above graph2 to select'''
        self.selectDataFileShow2.clear() #clear the combobox first before adding items
        if self.amount_sensor2Files > 0:
            sensor2files = str(self.sensor2FilesList)
```

```
182.             if "Mag" in sensor2files:
183.                 self.selectDataFileShow2.addItem("Gyroscopes", QtCore.QVariant(1))
184.                 self.selectDataFileShow2.addItem("Accelerometers", QtCore.QVariant(2))
185.                 self.selectDataFileShow2.addItem("Magnetometers", QtCore.QVariant(3))
186.             if "Euler" in sensor2files:
187.                 self.selectDataFileShow2.addItem("Euler", QtCore.QVariant(4))
188.             self.runGraphics2()
189.
190.     def saveFiles(self):
191.         self.location = QtWidgets.QFileDialog.getSaveFileName(self, 'Save File', '', 'CSV(*.csv)')
192.         print(type(self.location[0]), self.location[0])
193.         print(self.location)
194.         pass
195.
196.     def calibrate(self):
197.         ''''' First check if there are files of two sensors loaded '''
198.         if self.setCalibration == 2 and self.manualCalBox == 2:
199.             if self.amount_sensor1Files == 0 or self.amount_sensor2Files == 0:
200.                 print('Need two IMU files for calibration!')
201.                 return
202.             ''''' collect and initialize data first to put in fuctions, then runfunctions:
203.                 1. set derivative (takes starting and ending value and put these ito new arrays. One gyro, o
    ne accellero and
204.                     one derivative of the gyro.
205.                 2. run calibration (takes the three variables from 1 and put these in the loops for finding
    J and O, these
206.                     values are returned from te function and can be used for calculation.
207.             '''
208.             '''''' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$      1       $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
    $$$$$$$$$$$  '''
209.
210.             ''''' gather the neccesary data to put in new arrays for accelerometer and gyroscope, and gyro d
    erivative'''
211.             print('running data preperation......')
212.             startCalData = int(self.packageEdit1.text())    # get input from text-edit to set as start-
    datafile
213.             stopCalData = int(self.packageEdit2.text())     # get input from text-edit to set as stop-
    datafile
214.             frequency = self.setFrequency                   # get the set frequency
215.             gData1 = self.sensor1.gData             # get complete array of gyroscope data R(1*3) of sensor
    1
216.             gData2 = self.sensor2.gData             # get complete array of gyroscope data R(1*3) of sensor
    2
217.             aData1 = self.sensor1.aData             # get complete array of accellerometer data R(1*3) of se
    nsor 1
218.             aData2 = self.sensor2.aData             # get complete array of accellerometer data R(1*3) of se
    nsor 2
219.             ''''' Run the function to set the new arrays which are used for calibration'''
220.             # prepare the data for calibration in the range set by start/stop:
221.             self.dataSetting = data.Derivative(gData1, gData2, aData1, aData2, startCalData, stopCalData, se
    lf.dt, frequency)
222.
223.             gyro1 = self.dataSetting.gyro1              # take gyroscope data from function and rename for c
    alibration function
224.             gyro2 = self.dataSetting.gyro2              # do the above for sensor 2
225.             gyro1der = self.dataSetting.gyro1der        # take gyro derivative data from function and rename
     for calibration function
226.             gyro2der = self.dataSetting.gyro2der        # do the above for sensor 2
227.             accel1 = self.dataSetting.accel1            # take accellero data from function and rename for c
    alibration function
228.             accel2 = self.dataSetting.accel2            # do the above for sensor 2
229.
230.             '''''' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$      2       $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
    $$$$$$$$$$$  '''
```

```
            print('running calibration......')
            ''''' Set guessing values for the angles and vectors, to start the first alibration look with. The outcoms are optimalized'''
            x = np.array([[20], [60], [25], [55]])     # enter an initial guess for the agnles, used for the spherical coordinates of J
            x = x / 180 * np.pi               # change the angles in radians
            o1 = np.array([0.1, 0.1, 0.1])    # enter an initial guess for the vector o1, which is the distance to the joint
            o2 = np.array([0.1, 0.1, 0.1])    # enter an initial guess for the vector o2, which is the distance to the joint
            # set starting value for the loop to 0, because the calibration takes new variables from 1 which start with 0:
            startCal = 0
            # set the stopping value for the loop to the minimum value found in all the generated gyro, accel and derivative
            # files. This assures there is no array that exceeds the minimum length of any other array.
            stopCal = min([len(gyro1), len(gyro2), len(gyro1der), len(gyro2der), len(accel1), len(accel2)])

            ''''' run the calibration function, it takes variables from above and gives the J and O, witch are joint axis
                coordinates and the joint position coordinates '''
            self.calibration = calb.CalibrationJ(x, gData1, gData2, gyro1, gyro2, startCal, stopCal, frequency, self.dt)
            self.newj1 = self.calibration.newj1     # get the optimized J and set it as a variable witch can be
            self.newj2 = self.calibration.newj2     # used everywhere in the program.
            print('new J1 and J2 are set: ', self.newj1, self.newj2)
            self.calibrationStatus.setText('Succeeded')
        if self.setCalibration == 2 and self.manualCalBox == 0:
            self.calibrationStatus.setText('Check use manual data box!')
            print('automated clibration missing')
        if self.setCalibration == 3 and self.manualCalBox == 2:
            self.calibrationStatus.setText('3D NA')
            print('program 3D kinematics first')
        else:
            return

    def angleCalculation(self):
        if self.amount_sensor1Files == 0 or self.amount_sensor2Files == 0:
            return

        print('calculating the angle..')
        self.angle = []                 #empty the variable
        '''''  ^^^^^^^^^^^^^^^^^^^    RUN ANGLE.PY TO CALCULATE THE ANGLES     ^^^^^^^^^^^^^^^^^^^^^^^^'''

        self.angle = angle.AngleCalculation(0, 0, self.newj1, self.newj2, 0, 0, self.sensor1.gData, self.sensor2.gData,
                                            self.setFrequency)

        if self.sensor1.RM == True:
            self.selectAngleShowCombo.clear()
            self.selectAngleShowCombo.addItem("Euler X", QtCore.QVariant(1))
            self.selectAngleShowCombo.addItem("Euler Y", QtCore.QVariant(2))
            self.selectAngleShowCombo.addItem("Angle R Cos", QtCore.QVariant(5))
            print('comboset')
        if self.sensor1.CIM == True:
            self.selectAngleShowCombo.clear()
            self.selectAngleShowCombo.addItem("Angular rotation for Gyroscope", QtCore.QVariant(3))
            self.selectAngleShowCombo.addItem("Angular velocity", QtCore.QVariant(4))
        else:
            print("no data received for angle calculation")
        self.runGraphics3()

    def port(self):
```

```
284.         print('port function not defined')
285.
286.     ''''' Underneith this line are general functions for the UI, '''
287.
288.     def runGraphics1(self):
289.         if self.reloadCancel1 == False:
290.             self.sensorGraph1.clear()
291.             if self.reload1 == True:
292.                 self.setGraph1Show = 1
293.                 self.data1x = []
294.                 self.data1y = []
295.                 self.data1z = []
296.             if self.amount_sensor1Files == 0:
297.                 print('no files to display graph 1')
298.             else:
299.                 range1axis = [0]
300.                 range1axis = np.arange(1, self.sensor1.amount_packets+1)
301.                 if self.setGraph1Show == 1:
302.                     self.data1x = self.sensor1.gyroX
303.                     self.data1y = self.sensor1.gyroY
304.                     self.data1z = self.sensor1.gyroZ
305.                 if self.setGraph1Show == 2:
306.                     self.data1x = self.sensor1.accelX
307.                     self.data1y = self.sensor1.accelY
308.                     self.data1z = self.sensor1.accelZ
309.                 if self.setGraph1Show == 3:
310.                     self.data1x = self.sensor1.magnetoX
311.                     self.data1y = self.sensor1.magnetoY
312.                     self.data1z = self.sensor1.magnetoZ
313.                 self.sensorGraph1.plot(x=range1axis, y=self.data1x, pen='b')
314.                 self.sensorGraph1.plot(x=range1axis, y=self.data1y, pen='r')
315.                 self.sensorGraph1.plot(x=range1axis, y=self.data1z, pen='g')
316.                 self.sensorGraph1.show()
317.         else:
318.             print('check selected data for graphics 1')
319.
320.     def runGraphics2(self):
321.         if self.reloadCancel1 == False:
322.             self.sensorGraph2.clear()
323.             if self.reload2 == True:
324.                 self.setGraph2Show = 1
325.                 self.data2x = []
326.                 self.data2y = []
327.                 self.data2z = []
328.             if self.amount_sensor2Files == 0:
329.                 print('no files to display graph 2')
330.             else:
331.                 range2axis = [0]
332.                 range2axis = np.arange(1, self.sensor2.amount_packets + 1)
333.                 if self.setGraph2Show == 1:
334.                     self.data2x = self.sensor2.gyroX
335.                     self.data2y = self.sensor2.gyroY
336.                     self.data2z = self.sensor2.gyroZ
337.                 if self.setGraph2Show == 2:
338.                     self.data2x = self.sensor2.accelX
339.                     self.data2y = self.sensor2.accelY
340.                     self.data2z = self.sensor2.accelZ
341.                 if self.setGraph2Show == 3:
342.                     self.data2x = self.sensor2.magnetoX
343.                     self.data2y = self.sensor2.magnetoY
344.                     self.data2z = self.sensor2.magnetoZ
345.                 self.sensorGraph2.plot(x=range2axis, y=self.data2x, pen='b')
346.                 self.sensorGraph2.plot(x=range2axis, y=self.data2y, pen='r')
347.                 self.sensorGraph2.plot(x=range2axis, y=self.data2z, pen='g')
348.                 self.sensorGraph2.show()
```

```
349.         else:
350.             print('check selected data for graphics 2')
351.
352.     def runGraphics3(self):
353.         self.angleGraph.clear()
354.         range3axis = []
355.         self.data3a = []
356.         self.data3y = []
357.         self.data3z = []
358.         if self.amount_sensor2Files == 0:
359.             print('no files to display graph 3')
360.         else:
361.             print('now running the graphics script 3....')
362.             if self.setGraph3Show == 1:
363.                 self.data3a = self.angle.angleXrm1D
364.                 self.data3y = self.angle.angleXrm2D
365.                 self.data3z = self.angle.angleXrm1D + self.angle.angleXrm2D
366.             if self.setGraph3Show == 2:
367.                 self.data3a = self.angle.angleYrm1D
368.                 self.data3y = self.angle.angleYrm2D
369.                 self.data3z = self.angle.angleYrm1D + self.angle.angleYrm2D
370.             if self.setGraph3Show == 3:
371.                 range3axis = np.arange(1, len(self.angle.angleG) + 1)
372.                 self.data3a = self.angle.angleG
373.             if self.setGraph3Show == 4:
374.                 range3axis = np.arange(1, len(self.angle.angleV) + 1)
375.                 self.data3a = self.angle.angleV
376.             if self.setGraph3Show == 5:
377.                 range3axis = np.arange(1, len(self.angle.angleR) + 1)
378.                 self.data3a = self.angle.angleRxD
379.                 # self.data3y = self.angle.angleRinvD
380.                 # self.data3z = self.angle.angleRzD
381.             self.angleGraph.plot(x=range3axis, y=self.data3a, pen='y')
382.             # self.angleGraph.plot(x=range3axis, y=self.data3y, pen='r')
383.             # self.angleGraph.plot(x=range3axis, y=self.data3z, pen='g')
384.             self.angleGraph.show()
385.
386.     def graphHandle1(self):     # function that is performed when the values of the graphics-
    combobox are changed
387.         self.setGraph1Show = self.selectDataFileShow1.currentData()
388.         print('Selection graph-type 1', self.setGraph1Show)
389.         self.runGraphics1()
390.
391.     def graphHandle2(self):
392.         self.setGraph2Show = self.selectDataFileShow2.currentData()
393.         print('Selection graph-type 2', self.setGraph2Show)
394.         self.runGraphics2()
395.
396.     def graphHandle3(self):
397.         self.setGraph3Show = self.selectAngleShowCombo.currentData()
398.         print('Selection graph-type 3', self.setGraph3Show)
399.         self.runGraphics3()
400.
401. if __name__ == "__main__":
402.     app = QtWidgets.QApplication(sys.argv)
403.     application = MainWindow()
404.     application.show()
405.     app.exec_()
```

4.3 ThesisApp.py

```
__author__ = 'Arno van der Zwet'
__copyright__ = "Copyright 2017, Arno van der Zwet"
__title__ = 'Thesis Human Kinetic Technology'

# This file is created for
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'IMU.ui'
#
# Created by: PyQt5 UI code generator 5.8
#
# WARNING! All changes made in this file will be lost!

import sys
from PyQt5 import QtCore, QtGui, QtWidgets
from pyqtgraph import PlotWidget


class Ui_MainWindow(object):

    ''''' This is the main window class, this class is used and run in the main program. It will load all th
e elements
    of the programm and place these in the User Interface (UI). The basic structure and layout of this UI wa
s made
    with the help of QtDesigner, part of QtCreator. The output of the designer are .ui files, but were conve
rted
    in to windows command prompt using the 'pyuic' command (pyuic5 -x "ThesisApp.ui" -
o "ThesisApp.py). The outcome
    is part 1 and 3 of this python file, saved as .py file. '''

    '''''\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\       PART 1          \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ '''

    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(990, 801)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.gridLayout = QtWidgets.QGridLayout(self.centralwidget)
        self.gridLayout.setObjectName("gridLayout")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout()
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.gridLayout.addLayout(self.verticalLayout_4, 2, 5, 1, 1)
        self.line = QtWidgets.QFrame(self.centralwidget)
        self.line.setFrameShape(QtWidgets.QFrame.VLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.gridLayout.addWidget(self.line, 2, 1, 1, 1)
        self.groupBox_4 = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox_4.setMaximumSize(QtCore.QSize(16777215, 300))
        self.groupBox_4.setObjectName("groupBox_4")
        self.horizontalLayout_5 = QtWidgets.QHBoxLayout(self.groupBox_4)
        self.horizontalLayout_5.setObjectName("horizontalLayout_5")
        self.verticalLayout_9 = QtWidgets.QVBoxLayout()
        self.verticalLayout_9.setObjectName("verticalLayout_9")
        self.formLayout_7 = QtWidgets.QFormLayout()
        self.formLayout_7.setSizeConstraint(QtWidgets.QLayout.SetFixedSize)
        self.formLayout_7.setFieldGrowthPolicy(QtWidgets.QFormLayout.AllNonFixedFieldsGrow)
        self.formLayout_7.setRowWrapPolicy(QtWidgets.QFormLayout.DontWrapRows)
        self.formLayout_7.setLabelAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)
        self.formLayout_7.setFormAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|QtCore.Qt.AlignTop)
        self.formLayout_7.setContentsMargins(-1, 0, -1, -1)
        self.formLayout_7.setObjectName("formLayout_7")
```

```
58.         self.angleLabel = QtWidgets.QLabel(self.groupBox_4)
59.         self.angleLabel.setObjectName("angleLabel")
60.         self.formLayout_7.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.angleLabel)
61.         self.selectAngleShowCombo = QtWidgets.QComboBox(self.groupBox_4)
62.         self.selectAngleShowCombo.setMaximumSize(QtCore.QSize(16777215, 50))
63.         self.selectAngleShowCombo.setObjectName("selectAngleShowCombo")
64.         self.formLayout_7.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.selectAngleShowCombo)
65.         self.verticalLayout_9.addLayout(self.formLayout_7)
66.         self.angleGraph = PlotWidget(self.groupBox_4)
67.         self.angleGraph.setObjectName("angleGraph")
68.         self.verticalLayout_9.addWidget(self.angleGraph)
69.         self.verticalLayout_9.setStretch(1, 10)
70.         self.horizontalLayout_5.addLayout(self.verticalLayout_9)
71.         self.gridLayout.addWidget(self.groupBox_4, 4, 1, 1, 1)
72.         self.horizontalLayout_10 = QtWidgets.QHBoxLayout()
73.         self.horizontalLayout_10.setObjectName("horizontalLayout_10")
74.         self.verticalLayout = QtWidgets.QVBoxLayout()
75.         self.verticalLayout.setObjectName("verticalLayout")
76.         self.groupBox1 = QtWidgets.QGroupBox(self.centralwidget)
77.         self.groupBox1.setMaximumSize(QtCore.QSize(400, 16777215))
78.         self.groupBox1.setObjectName("groupBox1")
79.         self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.groupBox1)
80.         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
81.         self.groupBox_6 = QtWidgets.QGroupBox(self.groupBox1)
82.         self.groupBox_6.setObjectName("groupBox_6")
83.         self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.groupBox_6)
84.         self.verticalLayout_2.setObjectName("verticalLayout_2")
85.         self.openFileButton = QtWidgets.QPushButton(self.groupBox_6)
86.         self.openFileButton.setObjectName("openFileButton")
87.         self.verticalLayout_2.addWidget(self.openFileButton)
88.         self.openPortButton = QtWidgets.QPushButton(self.groupBox_6)
89.         self.openPortButton.setObjectName("openPortButton")
90.         self.verticalLayout_2.addWidget(self.openPortButton)
91.         spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expa
    nding)
92.         self.verticalLayout_2.addItem(spacerItem)
93.         self.horizontalLayout_2.addWidget(self.groupBox_6)
94.         self.groupBox = QtWidgets.QGroupBox(self.groupBox1)
95.         self.groupBox.setObjectName("groupBox")
96.         self.verticalLayout_7 = QtWidgets.QVBoxLayout(self.groupBox)
97.         self.verticalLayout_7.setObjectName("verticalLayout_7")
98.         self.manualCheckBox = QtWidgets.QCheckBox(self.groupBox)
99.         self.manualCheckBox.setObjectName("manualCheckBox")
100.        self.verticalLayout_7.addWidget(self.manualCheckBox)
101.        self.packageEdit1 = QtWidgets.QLineEdit(self.groupBox)
102.        self.packageEdit1.setMaximumSize(QtCore.QSize(100, 16777215))
103.        self.packageEdit1.setObjectName("packageEdit1")
104.        self.verticalLayout_7.addWidget(self.packageEdit1)
105.        self.packageEdit2 = QtWidgets.QLineEdit(self.groupBox)
106.        self.packageEdit2.setMaximumSize(QtCore.QSize(100, 16777215))
107.        self.packageEdit2.setObjectName("packageEdit2")
108.        self.verticalLayout_7.addWidget(self.packageEdit2)
109.        self.calibrationComboBox = QtWidgets.QComboBox(self.groupBox)
110.        self.calibrationComboBox.setMaximumSize(QtCore.QSize(100, 16777215))
111.        self.calibrationComboBox.setObjectName("calibrationComboBox")
112.        self.verticalLayout_7.addWidget(self.calibrationComboBox)
113.        self.calibrateButton = QtWidgets.QPushButton(self.groupBox)
114.        self.calibrateButton.setMinimumSize(QtCore.QSize(100, 23))
115.        self.calibrateButton.setMaximumSize(QtCore.QSize(100, 16777215))
116.        self.calibrateButton.setObjectName("calibrateButton")
117.        self.verticalLayout_7.addWidget(self.calibrateButton)
118.        spacerItem1 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Exp
    anding)
119.        self.verticalLayout_7.addItem(spacerItem1)
120.        self.horizontalLayout_2.addWidget(self.groupBox)
```

```
121.         self.groupBox_5 = QtWidgets.QGroupBox(self.groupBox1)
122.         self.groupBox_5.setObjectName("groupBox_5")
123.         self.verticalLayout_8 = QtWidgets.QVBoxLayout(self.groupBox_5)
124.         self.verticalLayout_8.setObjectName("verticalLayout_8")
125.         self.label_3 = QtWidgets.QLabel(self.groupBox_5)
126.         self.label_3.setMaximumSize(QtCore.QSize(100, 20))
127.         self.label_3.setObjectName("label_3")
128.         self.verticalLayout_8.addWidget(self.label_3)
129.         self.frequencyComboBox = QtWidgets.QComboBox(self.groupBox_5)
130.         self.frequencyComboBox.setObjectName("frequencyComboBox")
131.         self.verticalLayout_8.addWidget(self.frequencyComboBox)
132.         self.pushButton = QtWidgets.QPushButton(self.groupBox_5)
133.         self.pushButton.setObjectName("pushButton")
134.         self.verticalLayout_8.addWidget(self.pushButton)
135.         spacerItem2 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Exp
    anding)
136.         self.verticalLayout_8.addItem(spacerItem2)
137.         self.angleButton = QtWidgets.QPushButton(self.groupBox_5)
138.         self.angleButton.setObjectName("angleButton")
139.         self.verticalLayout_8.addWidget(self.angleButton)
140.         spacerItem3 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Exp
    anding)
141.         self.verticalLayout_8.addItem(spacerItem3)
142.         self.horizontalLayout_2.addWidget(self.groupBox_5)
143.         self.verticalLayout.addWidget(self.groupBox1)
144.         self.groupBox_2 = QtWidgets.QGroupBox(self.centralwidget)
145.         self.groupBox_2.setMaximumSize(QtCore.QSize(400, 16777215))
146.         self.groupBox_2.setObjectName("groupBox_2")
147.         self.horizontalLayout_6 = QtWidgets.QHBoxLayout(self.groupBox_2)
148.         self.horizontalLayout_6.setObjectName("horizontalLayout_6")
149.         self.horizontalLayout_12 = QtWidgets.QHBoxLayout()
150.         self.horizontalLayout_12.setObjectName("horizontalLayout_12")
151.         self.verticalLayout_12 = QtWidgets.QVBoxLayout()
152.         self.verticalLayout_12.setObjectName("verticalLayout_12")
153.         self.label_NumberOfFiles = QtWidgets.QLabel(self.groupBox_2)
154.         self.label_NumberOfFiles.setMaximumSize(QtCore.QSize(200, 16777215))
155.         self.label_NumberOfFiles.setObjectName("label_NumberOfFiles")
156.         self.verticalLayout_12.addWidget(self.label_NumberOfFiles, 0, QtCore.Qt.AlignLeft)
157.         self.label_MeasureTime = QtWidgets.QLabel(self.groupBox_2)
158.         self.label_MeasureTime.setMaximumSize(QtCore.QSize(200, 16777215))
159.         self.label_MeasureTime.setObjectName("label_MeasureTime")
160.         self.verticalLayout_12.addWidget(self.label_MeasureTime)
161.         self.label_amountPackets = QtWidgets.QLabel(self.groupBox_2)
162.         self.label_amountPackets.setMaximumSize(QtCore.QSize(200, 16777215))
163.         self.label_amountPackets.setObjectName("label_amountPackets")
164.         self.verticalLayout_12.addWidget(self.label_amountPackets)
165.         self.label_calibration = QtWidgets.QLabel(self.groupBox_2)
166.         self.label_calibration.setMaximumSize(QtCore.QSize(200, 16777215))
167.         self.label_calibration.setObjectName("label_calibration")
168.         self.verticalLayout_12.addWidget(self.label_calibration)
169.         self.label_angle = QtWidgets.QLabel(self.groupBox_2)
170.         self.label_angle.setMaximumSize(QtCore.QSize(200, 16777215))
171.         self.label_angle.setObjectName("label_angle")
172.         self.verticalLayout_12.addWidget(self.label_angle)
173.         spacerItem4 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Exp
    anding)
174.         self.verticalLayout_12.addItem(spacerItem4)
175.         self.horizontalLayout_12.addLayout(self.verticalLayout_12)
176.         self.verticalLayout_13 = QtWidgets.QVBoxLayout()
177.         self.verticalLayout_13.setObjectName("verticalLayout_13")
178.         self.amountFiles = QtWidgets.QLabel(self.groupBox_2)
179.         self.amountFiles.setMaximumSize(QtCore.QSize(200, 16777215))
180.         self.amountFiles.setObjectName("amountFiles")
181.         self.verticalLayout_13.addWidget(self.amountFiles, 0, QtCore.Qt.AlignLeft)
182.         self.amountTime = QtWidgets.QLabel(self.groupBox_2)
```

```
        self.amountTime.setMaximumSize(QtCore.QSize(200, 16777215))
        self.amountTime.setObjectName("amountTime")
        self.verticalLayout_13.addWidget(self.amountTime)
        self.amountPackets = QtWidgets.QLabel(self.groupBox_2)
        self.amountPackets.setMaximumSize(QtCore.QSize(200, 16777215))
        self.amountPackets.setObjectName("amountPackets")
        self.verticalLayout_13.addWidget(self.amountPackets)
        self.calibrationStatus = QtWidgets.QLabel(self.groupBox_2)
        self.calibrationStatus.setMaximumSize(QtCore.QSize(200, 16777215))
        self.calibrationStatus.setObjectName("calibrationStatus")
        self.verticalLayout_13.addWidget(self.calibrationStatus)
        self.amountAngle = QtWidgets.QLabel(self.groupBox_2)
        self.amountAngle.setMaximumSize(QtCore.QSize(200, 16777215))
        self.amountAngle.setObjectName("amountAngle")
        self.verticalLayout_13.addWidget(self.amountAngle)
        spacerItem5 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Exp
    anding)
        self.verticalLayout_13.addItem(spacerItem5)
        self.horizontalLayout_12.addLayout(self.verticalLayout_13)
        self.listLoadedFiles = QtWidgets.QListWidget(self.groupBox_2)
        self.listLoadedFiles.setMaximumSize(QtCore.QSize(300, 16777215))
        self.listLoadedFiles.setObjectName("listLoadedFiles")
        self.horizontalLayout_12.addWidget(self.listLoadedFiles, 0, QtCore.Qt.AlignLeft)
        self.horizontalLayout_6.addLayout(self.horizontalLayout_12)
        self.verticalLayout.addWidget(self.groupBox_2)
        self.verticalLayout.setStretch(0, 5)
        self.horizontalLayout_10.addLayout(self.verticalLayout)
        self.groupBox_3 = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox_3.setObjectName("groupBox_3")
        self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.groupBox_3)
        self.verticalLayout_6.setObjectName("verticalLayout_6")
        self.formLayout_4 = QtWidgets.QFormLayout()
        self.formLayout_4.setObjectName("formLayout_4")
        self.sensorDataLabel1 = QtWidgets.QLabel(self.groupBox_3)
        self.sensorDataLabel1.setObjectName("sensorDataLabel1")
        self.formLayout_4.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.sensorDataLabel1)
        self.selectDataFileShow1 = QtWidgets.QComboBox(self.groupBox_3)
        self.selectDataFileShow1.setObjectName("selectDataFileShow1")
        self.formLayout_4.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.selectDataFileShow1)
        self.verticalLayout_6.addLayout(self.formLayout_4)
        self.sensorGraph1 = PlotWidget(self.groupBox_3)
        self.sensorGraph1.setObjectName("sensorGraph1")
        self.verticalLayout_6.addWidget(self.sensorGraph1)
        self.formLayout_3 = QtWidgets.QFormLayout()
        self.formLayout_3.setObjectName("formLayout_3")
        self.sensorDataLabel2 = QtWidgets.QLabel(self.groupBox_3)
        self.sensorDataLabel2.setObjectName("sensorDataLabel2")
        self.formLayout_3.setWidget(1, QtWidgets.QFormLayout.LabelRole, self.sensorDataLabel2)
        self.selectDataFileShow2 = QtWidgets.QComboBox(self.groupBox_3)
        self.selectDataFileShow2.setObjectName("selectDataFileShow2")
        self.formLayout_3.setWidget(1, QtWidgets.QFormLayout.FieldRole, self.selectDataFileShow2)
        self.verticalLayout_6.addLayout(self.formLayout_3)
        self.sensorGraph2 = PlotWidget(self.groupBox_3)
        self.sensorGraph2.setEnabled(True)
        self.sensorGraph2.setObjectName("sensorGraph2")
        self.verticalLayout_6.addWidget(self.sensorGraph2)
        self.sensorGraph2.raise_()
        self.sensorGraph1.raise_()
        self.horizontalLayout_10.addWidget(self.groupBox_3)
        self.horizontalLayout_10.setStretch(1, 50)
        self.gridLayout.addLayout(self.horizontalLayout_10, 3, 1, 1, 1)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 990, 21))
        self.menubar.setObjectName("menubar")
```

```
247.         self.menuOpen_File = QtWidgets.QMenu(self.menubar)
248.         self.menuOpen_File.setObjectName("menuOpen_File")
249.         self.menuCalibration = QtWidgets.QMenu(self.menubar)
250.         self.menuCalibration.setObjectName("menuCalibration")
251.         self.menuCalibration_2 = QtWidgets.QMenu(self.menuCalibration)
252.         self.menuCalibration_2.setObjectName("menuCalibration_2")
253.         self.menuView = QtWidgets.QMenu(self.menubar)
254.         self.menuView.setObjectName("menuView")
255.         self.menuSensor1 = QtWidgets.QMenu(self.menuView)
256.         self.menuSensor1.setObjectName("menuSensor1")
257.         self.menuSensor2 = QtWidgets.QMenu(self.menuView)
258.         self.menuSensor2.setObjectName("menuSensor2")
259.         MainWindow.setMenuBar(self.menubar)
260.         self.statusbar = QtWidgets.QStatusBar(MainWindow)
261.         self.statusbar.setObjectName("statusbar")
262.         MainWindow.setStatusBar(self.statusbar)
263.         self.actionOpen_File = QtWidgets.QAction(MainWindow)
264.         self.actionOpen_File.setObjectName("actionOpen_File")
265.         self.actionExit = QtWidgets.QAction(MainWindow)
266.         self.actionExit.setObjectName("actionExit")
267.         self.actionSave = QtWidgets.QAction(MainWindow)
268.         self.actionSave.setObjectName("actionSave")
269.         self.actionExit_2 = QtWidgets.QAction(MainWindow)
270.         self.actionExit_2.setObjectName("actionExit_2")
271.         self.actionManual = QtWidgets.QAction(MainWindow)
272.         self.actionManual.setObjectName("actionManual")
273.         self.actionAutomatic = QtWidgets.QAction(MainWindow)
274.         self.actionAutomatic.setObjectName("actionAutomatic")
275.         self.actionGyroscope_data = QtWidgets.QAction(MainWindow)
276.         self.actionGyroscope_data.setObjectName("actionGyroscope_data")
277.         self.actionAccelerometer_data = QtWidgets.QAction(MainWindow)
278.         self.actionAccelerometer_data.setObjectName("actionAccelerometer_data")
279.         self.actionMagnetometer_data = QtWidgets.QAction(MainWindow)
280.         self.actionMagnetometer_data.setObjectName("actionMagnetometer_data")
281.         self.actionExit_3 = QtWidgets.QAction(MainWindow)
282.         self.actionExit_3.setObjectName("actionExit_3")
283.         self.actionAccelerometer = QtWidgets.QAction(MainWindow)
284.         self.actionAccelerometer.setObjectName("actionAccelerometer")
285.         self.actionGyroscope = QtWidgets.QAction(MainWindow)
286.         self.actionGyroscope.setObjectName("actionGyroscope")
287.         self.actionMagnetometer = QtWidgets.QAction(MainWindow)
288.         self.actionMagnetometer.setObjectName("actionMagnetometer")
289.         self.actionClear_cashe = QtWidgets.QAction(MainWindow)
290.         self.actionClear_cashe.setObjectName("actionClear_cashe")
291.         self.actionConnection = QtWidgets.QAction(MainWindow)
292.         self.actionConnection.setObjectName("actionConnection")
293.         self.actionQuit = QtWidgets.QAction(MainWindow)
294.         self.actionQuit.setObjectName("actionQuit")
295.         self.menuOpen_File.addAction(self.actionOpen_File)
296.         self.menuOpen_File.addAction(self.actionSave)
297.         self.menuOpen_File.addSeparator()
298.         self.menuOpen_File.addAction(self.actionQuit)
299.         self.menuCalibration_2.addAction(self.actionManual)
300.         self.menuCalibration_2.addAction(self.actionAutomatic)
301.         self.menuCalibration.addAction(self.actionConnection)
302.         self.menuCalibration.addAction(self.menuCalibration_2.menuAction())
303.         self.menuCalibration.addSeparator()
304.         self.menuCalibration.addAction(self.actionClear_cashe)
305.         self.menuSensor1.addAction(self.actionGyroscope_data)
306.         self.menuSensor1.addAction(self.actionAccelerometer_data)
307.         self.menuSensor1.addAction(self.actionMagnetometer_data)
308.         self.menuSensor2.addAction(self.actionGyroscope)
309.         self.menuSensor2.addAction(self.actionAccelerometer)
310.         self.menuSensor2.addAction(self.actionMagnetometer)
311.         self.menuView.addAction(self.menuSensor1.menuAction())
```

```
312.         self.menuView.addAction(self.menuSensor2.menuAction())
313.         self.menubar.addAction(self.menuOpen_File.menuAction())
314.         self.menubar.addAction(self.menuCalibration.menuAction())
315.         self.menubar.addAction(self.menuView.menuAction())
316.
317.         self.retranslateUi(MainWindow)
318.         QtCore.QMetaObject.connectSlotsByName(MainWindow)
319.
320.         '''''\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\       PART 2          \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    '''
321.         '''''######## here are general button activities, do not delete, but copy when the layout is renewed
    ##########'''
322.         #self.openPortButton.clicked.connect(MainWindow.closeApplication)
323.         self.actionQuit.triggered.connect(MainWindow.closeApplication)
324.         self.selectDataFileShow2.activated['QString'].connect(MainWindow.update)
325.         self.packageEdit1.setText("start number")
326.         self.packageEdit2.setText("end number")
327.         #add STANDARD items to comboBox, variable items are set in Main
328.         self.calibrationComboBox.addItem("2D Kinematics", QtCore.QVariant(2))
329.         self.calibrationComboBox.addItem("3D Kinematics", QtCore.QVariant(3))
330.         self.frequencyComboBox.addItem("1 Hz", QtCore.QVariant(1))
331.         self.frequencyComboBox.addItem("2 Hz", QtCore.QVariant(2))
332.         self.frequencyComboBox.addItem("4 Hz", QtCore.QVariant(4))
333.         self.frequencyComboBox.addItem("8 Hz", QtCore.QVariant(8))
334.         self.frequencyComboBox.addItem("16 Hz", QtCore.QVariant(16))
335.         self.frequencyComboBox.addItem("32 Hz", QtCore.QVariant(32))
336.         self.frequencyComboBox.addItem("64 Hz", QtCore.QVariant(64))
337.         self.frequencyComboBox.addItem("128 Hz", QtCore.QVariant(128))
338.         self.frequencyComboBox.addItem("256 Hz", QtCore.QVariant(256))
339.         self.frequencyComboBox.addItem("512 Hz", QtCore.QVariant(512))
340.
341.         # at initialize, get the QVariant (data) from the combobox and put it in setCalibration
342.         self.setCalibration = self.calibrationComboBox.currentData()
343.         self.setFrequency = self.frequencyComboBox.currentData()
344.
345.         #if checkbox is changed, run ....Handle which gets the string and data
346.         self.calibrationComboBox.currentIndexChanged[str].connect(self.calibrationHandle)
347.         self.frequencyComboBox.currentIndexChanged[str].connect(self.frequencyHandle)
348.         self.selectDataFileShow1.currentIndexChanged[str].connect(self.graphHandle1)
349.         self.selectDataFileShow2.currentIndexChanged[str].connect(self.graphHandle2)
350.         self.selectAngleShowCombo.currentIndexChanged[str].connect(self.graphHandle3)
351.         self.manualCheckBox.stateChanged.connect(self.manualHandle)
352.
353.     '''''\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\       PART 3          \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ '''

354.     ''''' @@@@@@@@@@@@@@@@@@    Here the strings and titles are set in the elements    @@@@@@@@@@@@@@@@@@@@@@
    @@'''
355.     def retranslateUi(self, MainWindow):
356.         _translate = QtCore.QCoreApplication.translate
357.         MainWindow.setWindowTitle(_translate("MainWindow", "Thesis IMU 2017 1.0"))#copy in new layout
358.         self.groupBox_4.setTitle(_translate("MainWindow", "Angle"))
359.         self.angleLabel.setText(_translate("MainWindow", "Select Angle"))
360.         self.groupBox1.setTitle(_translate("MainWindow", "Data Tools"))
361.         self.groupBox_6.setTitle(_translate("MainWindow", "Start"))
362.         self.openFileButton.setText(_translate("MainWindow", "Open File"))
363.         self.openPortButton.setText(_translate("MainWindow", "Open Port"))
364.         self.groupBox.setTitle(_translate("MainWindow", "Calibration"))
365.         self.manualCheckBox.setText(_translate("MainWindow", "Use manual data"))
366.         self.calibrateButton.setText(_translate("MainWindow", "Calibrate"))
367.         self.groupBox_5.setTitle(_translate("MainWindow", "Data"))
368.         self.label_3.setText(_translate("MainWindow", "Frequency"))
369.         self.pushButton.setText(_translate("MainWindow", "PushButton"))
370.         self.angleButton.setText(_translate("MainWindow", "Calculate Angel"))
371.         self.groupBox_2.setTitle(_translate("MainWindow", "Data information"))
372.         self.label_NumberOfFiles.setText(_translate("MainWindow", "Files loaded"))
```

```
373.         self.label_MeasureTime.setText(_translate("MainWindow", "Measuring time (sec):"))
374.         self.label_amountPackets.setText(_translate("MainWindow", "Number of packets:"))
375.         self.label_calibration.setText(_translate("MainWindow", "Calibration"))
376.         self.label_angle.setText(_translate("MainWindow", "Angle"))
377.         self.amountFiles.setText(_translate("MainWindow", "No files selected"))
378.         self.amountTime.setText(_translate("MainWindow", "......"))
379.         self.amountPackets.setText(_translate("MainWindow", "......"))
380.         self.calibrationStatus.setText(_translate("MainWindow", "not done yet"))
381.         self.amountAngle.setText(_translate("MainWindow", "......"))
382.         self.groupBox_3.setTitle(_translate("MainWindow", "Measured data"))
383.         self.sensorDataLabel1.setText(_translate("MainWindow", "Data Sensor 1"))
384.         self.sensorDataLabel2.setText(_translate("MainWindow", "Data Sensor 2"))
385.         self.menuOpen_File.setTitle(_translate("MainWindow", "File"))
386.         self.menuCalibration.setTitle(_translate("MainWindow", "Options"))
387.         self.menuCalibration_2.setTitle(_translate("MainWindow", "Calibration"))
388.         self.menuView.setTitle(_translate("MainWindow", "View"))
389.         self.menuSensor1.setTitle(_translate("MainWindow", "Data sensor 1"))
390.         self.menuSensor2.setTitle(_translate("MainWindow", "Data sensor 2"))
391.         self.actionOpen_File.setText(_translate("MainWindow", "Open"))
392.         self.actionExit.setText(_translate("MainWindow", "Exit"))
393.         self.actionSave.setText(_translate("MainWindow", "Save"))
394.         self.actionExit_2.setText(_translate("MainWindow", "Exit"))
395.         self.actionManual.setText(_translate("MainWindow", "Manual"))
396.         self.actionAutomatic.setText(_translate("MainWindow", "Automatic"))
397.         self.actionGyroscope_data.setText(_translate("MainWindow", "Gyroscope"))
398.         self.actionAccelerometer_data.setText(_translate("MainWindow", "Accelerometer"))
399.         self.actionMagnetometer_data.setText(_translate("MainWindow", "Magnetometer"))
400.         self.actionExit_3.setText(_translate("MainWindow", "Quit"))
401.         self.actionAccelerometer.setText(_translate("MainWindow", "Accelerometer"))
402.         self.actionGyroscope.setText(_translate("MainWindow", "Gyroscope"))
403.         self.actionMagnetometer.setText(_translate("MainWindow", "Magnetometer"))
404.         self.actionClear_cashe.setText(_translate("MainWindow", "Clear cache"))
405.         self.actionConnection.setText(_translate("MainWindow", "Connection"))
406.         self.actionQuit.setText(_translate("MainWindow", "Quit"))
407.         self.actionQuit.setWhatsThis(_translate("MainWindow", "Quit the program"))
408.
409.     '''''\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\       PART 4          \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ '''

410.     ''''' @@@@@@@@@@@@@@@@@@@@      Here some extra functionality is created for the UI     @@@@@@@@@@@@@@@@@@@'
     ''
411.     # method for closing the app by sys exit, afer clicked yes in message box
412.     def closeApplication(self):
413.         choise = QtWidgets.QMessageBox.question(self, 'Quit Application', "Are you sure you want to quit?",

414.                                                 QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No)
415.         if choise == QtWidgets.QMessageBox.Yes:
416.             sys.exit()
417.         else:
418.             pass
419.
420.     # method for closing the app by sys exit, afer clicked yes in message box
421.     def calibrationHandle(self, text):
422.         self.setCalibration = self.calibrationComboBox.currentData()
423.         print('Calibration is set to: %s' % text)
424.         print('calibration value', self.setCalibration)
425.
426.     def frequencyHandle(self, text):
427.         self.setFrequency = self.frequencyComboBox.currentData()
428.         self.setDataJump = self.setFrequency/10
429.         print('frequency: ', self.setFrequency)
430.
431.     def manualHandle(self):
432.         self.manualCalBox = self.manualCheckBox.checkState()
433.         self.packageEdit1.setText('')
434.         self.packageEdit2.setText('')
```

4.4 Data.py

```
__author__ = 'Arno van der Zwet'
__copyright__ = "Copyright 2017, Arno van der Zwet"
__title__ = 'Thesis Human Kinetic Technology'
'''''
created on: feb 2017
file name: Data.py
purpose: load and manipulate csv files for main.py
'''

#import the nessacary modules and functions, this may also conttain funtions from pythonfiles that are made for this project

import numpy as np

'''''########################################  FILE LOADING   ########################################
#'''


class LoadFiles:

    def __init__(self, filenames): #init is used to set variables for the class, it takes the arguments that are after self
        nofilecheck = not all(filenames)
        if nofilecheck == True:
            return
        self.filenames = filenames
        self.amount_files = len(self.filenames)                 #read how many files are selected
        print("files in manualData", self.amount_files)
        '''''Check which file contains each strings is which and put them into a variable'''
        # first set the strings of what the list should be checked of
        self.txtCIM = "CalInertialAndMag"
        self.txtIO = "DigitalIO"
        self.txtEA = "EulerAngles"
        self.txtQ = "Quaternion"
        self.txtRM = "RotationMatrix"
        self.amount_packets = []

        if self.amount_files == 0:                  #if only one file is selected, generate nly one file.
            print("no files were selected")
            self.CIM = []
            self.IO = []
            self.EA = []
            self.Q = []
            self.RM = []
        else:
            # Take the list of sensor one and search in the text for each strings set above.
            # Every 'if' statement sets loads the data if it contains the text
            for text in self.filenames:
                if self.txtCIM in text:
                    self.dataCIM = np.genfromtxt(text, delimiter=",", skip_header=1)
                    self.CIM = True
                #if self.txtIO in text:
                    #self.dataIO = np.genfromtxt(text, delimiter=",", skip_header=1)
                    #self.IO = True
                if self.txtEA in text:
                    self.dataEA = np.genfromtxt(text, delimiter=",", skip_header=1)
                    self.EA = True
                if self.txtEA not in text:
                    self.EA = False
                if self.txtQ in text:
                    self.dataQ = np.genfromtxt(text, delimiter=",", skip_header=1)
                    self.Q = True
```

```
                if self.txtQ not in text:
                    self.Q = False
                if self.txtRM in text:
                    self.dataRM = np.genfromtxt(text, delimiter=",", skip_header=1)
                    self.RM = True
                if self.txtRM not in text:
                    self.RM = False
                self.amount_packets = len(self.dataCIM)
        print('amount of packets', self.amount_packets)
        if self.RM == True:
            self.rmData = self.dataRM [0:, range(1, 10)]
        else:
            pass
        if self.CIM == True:
            self.gData = self.dataCIM[0:, range(1, 4)]
            self.aData = self.dataCIM[0:, range(4, 7)]

            self.gyroX = self.dataCIM[0:, 1]
            self.gyroY = self.dataCIM[0:, 2]
            self.gyroZ = self.dataCIM[0:, 3]
            self.accelX = self.dataCIM[0:, 4]
            self.accelY = self.dataCIM[0:, 5]
            self.accelZ = self.dataCIM[0:, 6]
            self.magnetoX = self.dataCIM[0:, 7]
            self.magnetoY = self.dataCIM[0:, 8]
            self.magnetoZ = self.dataCIM[0:, 9]
            self.gyro = self.dataCIM[1:, range(1, 4)]
            self.accel = self.dataCIM[1:, [4, 5, 6]]
        else:
            pass


class Derivative:
    def __init__(self, gData1, gData2, aData1, aData2, start, stop, dt, frequency):
        print('running derivative...')
        self.gData1 = gData1
        self.gData2 = gData2
        self.aData1 = aData1
        self.aData2 = aData2
        (print('data loaded'))
        length = [len(self.gData1), len(self.gData2)]
        print(min(length))
        self.dt = dt
        self.frequency = frequency
        self.dt2 = self.dt/self.frequency
        self.start = start - 1
        self.stop = stop - 1
        self.gyro1 = dict()
        self.gyro2 = dict()
        self.gyro1der = dict()
        self.gyro2der = dict()
        self.accel1 = dict()
        self.accel2 = dict()
        for u in range(self.start, self.stop, dt):
            self.gyro1[u] = np.append([self.gData1[u]], [])
            self.gyro2[u] = np.append([self.gData2[u]], [])
            self.gyro1der[u] = np.append([(((self.gData1[u - 2 * self.dt]) - (8 * self.gData1[u - self.dt]) +
    (
                8 * self.gData1[u + self.dt]) - (self.gData1[u + 2 * self.dt])) / (12 * self.dt2)], [])
            self.gyro2der[u] = np.append([(((self.gData2[u - 2 * self.dt]) - (8 * self.gData2[u - self.dt]) +
    (
                8 * self.gData2[u + self.dt]) - (self.gData2[u + 2 * self.dt])) / (12 * self.dt2)], [])
            self.accel1[u] = np.append([self.aData1[u]], [])
            self.accel2[u] = np.append([self.aData2[u]], [])
        self.gyro1      = list(self.gyro1.values())
```

```
123.         self.gyro2      = list(self.gyro2.values())
124.         self.gyro1der   = list(self.gyro1der.values())
125.         self.gyro2der   = list(self.gyro2der.values())
126.         self.accel1     = list(self.accel1.values())
127.         self.accel2     = list(self.accel2.values())
128.         print('derivative set')
```

4.5 Calibration.py

```
__author__ = 'Arno van der Zwet'
__copyright__ = "Copyright 2017, Arno van der Zwet"
__title__ = 'Thesis Human Kinetic Technology'


#import the nessacary modules and functions, this may also conttain funtions from pythonfiles that are made for this project
import numpy as np
import math as mt


class CalibrationJ:

    def __init__(self, x, gData1, gData2, gyro1, gyro2, start, stop, frequence, dt):
        self.x = x
        self.gyro1 = gyro1
        self.gyro2 = gyro2
        self.start = start
        self.stop = stop
        self.frequence = frequence
        self.dt = dt
        self.dtx = self.dt/frequence
        self.gData1 = gData1
        self.gData2 = gData2
        self.defineJ()                      # run this method while initialize

    def defineJ(self):
        iter = 0
        self.meanError = [0]  # create variable for vstack, 0 because it does not work if variable is empty

        print("Initial values used for optimization: ", self.x, self.start, self.stop, self.dt)
        for p in range(0, 30):
            np.seterr(all='ignore') # this function makes sure no error appears when the value given in np.cos nears 0
            '''''Before the loop is run, first initialate the vales for the function:'''
            # from x (the spherical coordinates):
            phi1 = self.x[0]
            theta1 = self.x[1]
            phi2 = self.x[2]
            theta2 = self.x[3]
            # create empty arrays to fill:
            errorVector = [0]                               # empty the errorvector
            self.jacobian = np.array([0, 0, 0, 0])          # empty the jacobian matrix
            self.pseudoInverse = []                         # empty the pseudoInverse of the Jacobian
            ''''' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  step 1  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'''
            '''''                 step 1: calculate J1 and J2, and take one row from G1 and G2
  '''
            self.j1 = np.array([mt.cos(phi1)*mt.cos(theta1), mt.cos(phi1)*mt.sin(theta1), mt.sin(phi1)])
            self.j2 = np.array([mt.cos(phi2)*mt.cos(theta2), mt.cos(phi2)*mt.sin(theta2), mt.sin(phi2)])

            for h in range(0, self.stop):
                g1 = self.gyro1[h]
                g2 = self.gyro2[h]
                g1x = self.gyro1[h][0]
                g1y = self.gyro1[h][1]
                g1z = self.gyro1[h][2]
                g2x = self.gyro2[h][0]
                g2y = self.gyro2[h][1]
                g2z = self.gyro2[h][2]
                ''''' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  step 2  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'''
```

```
                ''''' Step 2: Calculate the errorVector '''
                errorVector = np.vstack((errorVector, np.array([np.linalg.norm(np.cross(
                    self.j1, g1)) - np.linalg.norm(np.cross(self.j2, g2))])))
                ''''' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  step 3  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'''
                ''''' step 3.1: Calculate the Jacobian '''
                jacob = np.array([(2 * abs(g1x * np.sin(phi1) - g1z * np.cos(phi1) * np.cos(theta1)) * np.sign(
                    g1x * np.sin(phi1) - g1z * np.cos(phi1) * np.cos(theta1)) * (g1x * np.cos(phi1) + g1z * np.cos(
                    theta1) * np.sin(phi1)) + 2 * abs(g1y * np.sin(phi1) - g1z * np.cos(phi1) * np.sin(theta1)) * np.sign(
                    g1y * np.sin(phi1) - g1z * np.cos(phi1) * np.sin(theta1)) * (g1y * np.cos(phi1) + g1z * np.sin(
                    phi1) * np.sin(theta1)) - 2 * abs(g1y * np.cos(phi1) * np.cos(theta1) - g1x * np.cos(phi1) * np.sin(
                    theta1)) * np.sign(g1y * np.cos(phi1) * np.cos(theta1) - g1x * np.cos(phi1) * np.sin(theta1)) * (
                    g1y * np.cos(theta1) * np.sin(phi1) - g1x * np.sin(phi1) * np.sin(theta1))) / (2 * (abs(g1y * np.cos(
                    phi1) * np.cos(theta1) - g1x * np.cos(phi1) * np.sin(theta1)) ** 2 + abs(g1x * np.sin(
                    phi1) - g1z * np.cos(phi1) * np.cos(theta1)) ** 2 + abs(g1y * np.sin(phi1) - g1z * np.cos(
                    phi1) * np.sin(theta1)) ** 2) ** (1 / 2)),
                    -
(2 * abs(g1y * np.cos(phi1) * np.cos(theta1) - g1x * np.cos(phi1) * np.sin(theta1)) * np.sign(
                    g1y * np.cos(phi1) * np.cos(theta1) - g1x * np.cos(phi1) * np.sin(theta1)) * (
                    g1x * np.cos(phi1) * np.cos(theta1) + g1y * np.cos(phi1) * np.sin(theta1)) - 2 * g1z * abs(
                    g1x * np.sin(phi1) - g1z * np.cos(phi1) * np.cos(theta1)) * np.sign(g1x * np.sin(
                    phi1) - g1z * np.cos(phi1) * np.cos(theta1)) * np.cos(phi1) * np.sin(theta1) + 2 * g1z * abs(
                    g1y * np.sin(phi1) - g1z * np.cos(phi1) * np.sin(theta1)) * np.sign(g1y * np.sin(
                    phi1) - g1z * np.cos(phi1) * np.sin(theta1)) * np.cos(phi1) * np.cos(theta1)) / (2 * (abs(
                    g1y * np.cos(phi1) * np.cos(theta1) - g1x * np.cos(phi1) * np.sin(theta1)) ** 2 + abs(
                    g1x * np.sin(phi1) - g1z * np.cos(phi1) * np.cos(theta1)) ** 2 + abs(g1y * np.sin(
                    phi1) - g1z * np.cos(phi1) * np.sin(theta1)) ** 2) ** (1 / 2)),
                    -
(2 * abs(g2x * np.sin(phi2) - g2z * np.cos(phi2) * np.cos(theta2)) * np.sign(g2x * np.sin(
                    phi2) - g2z * np.cos(phi2) * np.cos(theta2)) * (g2x * np.cos(phi2) + g2z * np.cos(theta2) * np.sin(
                    phi2)) + 2 * abs(g2y * np.sin(phi2) - g2z * np.cos(phi2) * np.sin(theta2)) * np.sign(g2y * np.sin(
                    phi2) - g2z * np.cos(phi2) * np.sin(theta2)) * (g2y * np.cos(phi2) + g2z * np.sin(phi2) * np.sin(
                    theta2)) - 2 * abs(g2y * np.cos(phi2) * np.cos(theta2) - g2x * np.cos(phi2) * np.sin(
                    theta2)) * np.sign(g2y * np.cos(phi2) * np.cos(theta2) - g2x * np.cos(phi2) * np.sin(theta2)) * (
                    g2y * np.cos(theta2) * np.sin(phi2) - g2x * np.sin(phi2) * np.sin(theta2))) / (2 * (abs(
                    g2y * np.cos(phi2) * np.cos(theta2) - g2x * np.cos(phi2) * np.sin(theta2)) ** 2 + abs(
                    g2x * np.sin(phi2) - g2z * np.cos(phi2) * np.cos(theta2)) ** 2 + abs(g2y * np.sin(
                    phi2) - g2z * np.cos(phi2) * np.sin(theta2)) ** 2) ** (1 / 2)),
                    (2 * abs(g2y * np.cos(phi2) * np.cos(theta2) - g2x * np.cos(phi2) * np.sin(theta2)) * np.sign(
                    g2y * np.cos(phi2) * np.cos(theta2) - g2x * np.cos(phi2) * np.sin(theta2)) * (g2x * np.cos(
                    phi2) * np.cos(theta2) + g2y * np.cos(phi2) * np.sin(theta2)) - 2 * g2z * abs(g2x * np.sin(
                    phi2) - g2z * np.cos(phi2) * np.cos(theta2)) * np.sign(g2x * np.sin(phi2) - g2z * np.cos(
                    phi2) * np.cos(theta2)) * np.cos(phi2) * np.sin(theta2) + 2 * g2z * abs(g2y * np.sin(
```

```
                    phi2) - g2z * np.cos(phi2) * np.sin(theta2)) * np.sign(g2y * np.sin(phi2) - g2z * np.cos(
                    phi2) * np.sin(theta2)) * np.cos(phi2) * np.cos(theta2)) / (2 * (abs(g2y * np.cos(phi2) * np.cos(
                    theta2) - g2x * np.cos(phi2) * np.sin(theta2)) ** 2 + abs(g2x * np.sin(phi2) - g2z * np.cos(
                    phi2) * np.cos(theta2)) ** 2 + abs(g2y * np.sin(phi2) - g2z * np.cos(phi2) * np.sin(
                    theta2)) ** 2) ** (1 / 2))]])
                jacob1 = jacob[0, 0]        # take 1st value from the array jacob
                jacob2 = jacob[1, 0]        # take 2nd value from the array jacob
                jacob3 = jacob[2, 0]        # take 3rd value from the array jacob
                jacob4 = jacob[3, 0]        # take 4th value from the array jacob
                # organize and stack(add) the above values into the jacobian matrix
                self.jacobian = np.vstack((self.jacobian, np.array([jacob1, jacob2, jacob3, jacob4])))
            '''''end of the loop, the arrays created are used under this line'''

            errorVector = errorVector[1:, :]        # throw away the first column because it contains zero's
            self.jacobian = self.jacobian[1:, :]    # throw away the first column because it contains zero's
            '''''                 step 3.2:   Calculate the pseudo inverse of the jacobian                   '''
            self.pseudoInverse = np.linalg.pinv(self.jacobian)
            ''''' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  step 4  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'''
            '''''                     step 4: Update x by x = x - mppi*errorJ                             '''
            dot = np.dot(self.pseudoInverse, errorVector)
            self.new_x = self.x - dot
            self.x = self.new_x
            ''''' For optimalization of the loop: '''
            iter = iter+1                           # raise amount of iterations with 1 after each loop
            self.meanError = np.vstack((self.meanError, abs(np.mean(errorVector))))   # set errorvalue to absolute, this assures the loop
                                                            # ends even if the error is negative
        else:
            self.meanError = self.meanError[1:, :]
            print('Number of iterations', iter)
            print('Final error: ', self.meanError)
            self.newj1 = self.j1
            self.newj2 = self.j2
            print('norm J1, J2 for unitvector', np.linalg.norm(self.j1), np.linalg.norm(self.j2))
            print('length errorvector:', len(errorVector))
            return

    def checkSign(self):
        print('çhecking signs.....')
        self.sign1 = np.dot(self.gData1[3300, 0:], self.newj1)
        self.sign2 = np.dot(self.gData2[3300, 0:], self.newj2)
        print(self.sign1, self.sign2)
```

4.6 Angle.py

```
__author__ = 'Arno van der Zwet'
__copyright__ = "Copyright 2017, Arno van der Zwet"
__title__ = 'Thesis Human Kinetic Technology'

import numpy as np
import scipy.integrate as scip


class AngleCalculation:

    def __init__(self, newO1, newO2, newJ1, newJ2, aData1, aData2, gData1, gData2, frequency):
        self.newO1 = newO1
        self.newO1 = newO2
        self.newJ1 = newJ1
        self.newJ2 = newJ2
        self.aData1 = aData1
        self.aData2 = aData2
        self.gData1 = gData1
        self.gData2 = gData2
        self.C = [1, 0, 0]
        self.rotMat = []
        self.frequency = frequency
        self.angleG()

    def angleG(self):
        length = [len(self.gData1), len(self.gData2)]
        stop = min(length)
        self.angleV = []
        print('checkpoint2')
        for i in range(0, stop-1):
            g11 = self.gData1[i]
            g12 = self.gData1[i+1]
            g21 = self.gData2[i]
            g22 = self.gData1[i+1]
            G1 = g12-g11
            G2 = g22-g21
            self.angleV = np.append(self.angleV, [(np.dot(G1, self.newJ1)) - (np.dot(G2, self.newJ2))])
        self.angleGdeg = np.rad2deg(self.angleV)
        self.angleG = scip.cumtrapz(self.angleV, dx=(1/self.frequency))
        self.angleG2deg = np.trapz(self.angleV, dx=(1/self.frequency))
```

Appendix 5, Hypothesis calibration tests

Table 7, Overview of the input data for the Calibration tests and the Hypothesis. The first five tests are different measurements, the sixth and seventh test will be executed with the IMU-data from measurement 5.

Input					**Hypothesis**					
Test number	Filename	StartValueData Synchronization	Calibration dataset	Initial value for X (phi1, theta1, phi2, theta2)	J1	J2	Iterations for J	Sign	Angle min	Angle max
1	1,1	**968**		0,1 ; 0,5 ; 0,2 ; 0,5	0,000	0,000	< 20	+	± 90	± 180
	1,2	**1150**			0,000	0,000				
					1,000	-1,000				
2	2,1	**2056**		0,1 ; 0,5 ; 0,2 ; 0,5	0,000	0,000	< 20	-	± 90	± 180
	2,2	**2240**			0,000	0,000				
					1,000	1,000				
3	3,1	**1700**		0,1 ; 0,5 ; 0,2 ; 0,5	-	-	< 20	+	± 90	± 180
	3,2	**1897**			-	-				
					-	-				
4	4,1	**1296**		0,1 ; 0,5 ; 0,2 ; 0,5	-	-	< 20	+	± 90	± 180
	4,2	**1457**			-	-				
					-	-				
5	5,1	**1828**		0,1 ; 0,5 ; 0,2 ; 0,5	-	-	< 20	+	± 90	± 180
	5,2	**1663**			-	-				
					-	-				
6	5,1	**1828**	Same as test 5	0,4 ; 0,2 ; 0,1 ; 0,7	Same as test 5		< 20	+	± 90	± 180
	5,2	**1663**								
7	5,1	**1828**	859	0,4 ; 0,2 ; 0,1 ; 0,7	Same as test 5		< 20	+	± 90	± 180
	5,2	**1663**	5							
			886							

Appendix 6, Results

6.1 Results calibration tests

Table 8, Result overview from the calibration test. The used data-points for calibration were selected randomly using the ginput() function. The amount of data is the total amount of gyroscope data that is run through the optimization loop 30 times.

Result Calibration								
Testnr	Data used calibration (start, step, stop)	Amount of data	Iterations for J	Sign	Determined J1	Determined J2	Deviation of predicted J from hypothesis	
1	425	205	11	-	-0,040	-0,043	-0,040	-0,043
	5				0,060	-0,047	0,060	-0,047
	1449				0,997	-0,998	-0,003	0,002
2	433	236	13	+	0,012	0,000	0,012	0,000
	5				0,006	0,017	0,006	0,017
	1609				1,000	1,000	0,000	0,000
3	445	255	16	-	0,123	0,123		
	5				-0,099	0,078		
	1719				-0,987	0,989		
4	1478	229	18	-	-0,136	-0,283		
	5				0,102	0,219		
	2622				0,985	-0,934		
5	423	222	15	-	0,255	0,309		
	5				0,167	0,283		
	1530				-0,952	0,908		
6	423	222	18	-	0,255	0,309	0,000	0,000
	5				0,167	0,283	0,000	0,000
	1530				-0,952	0,908	0,000	0,000
7	859	6	25	-	0,235	0,227	-0,020	-0,081
	5				0,359	0,294	0,192	0,011
	886				-0,903	0,928	0,049	0,020
	Averages (test 1-6):	228	15					

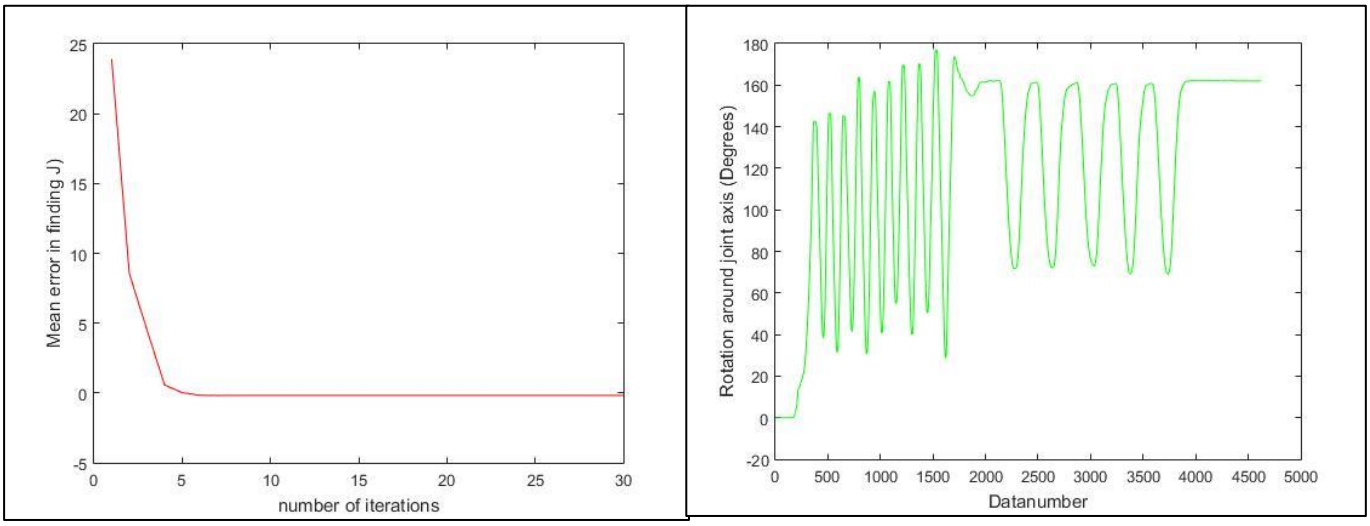

Figure 19, Results Test 1, L: Progress of the mean error during 30 optimization iterations. R: Angular rotations of the total measurement.

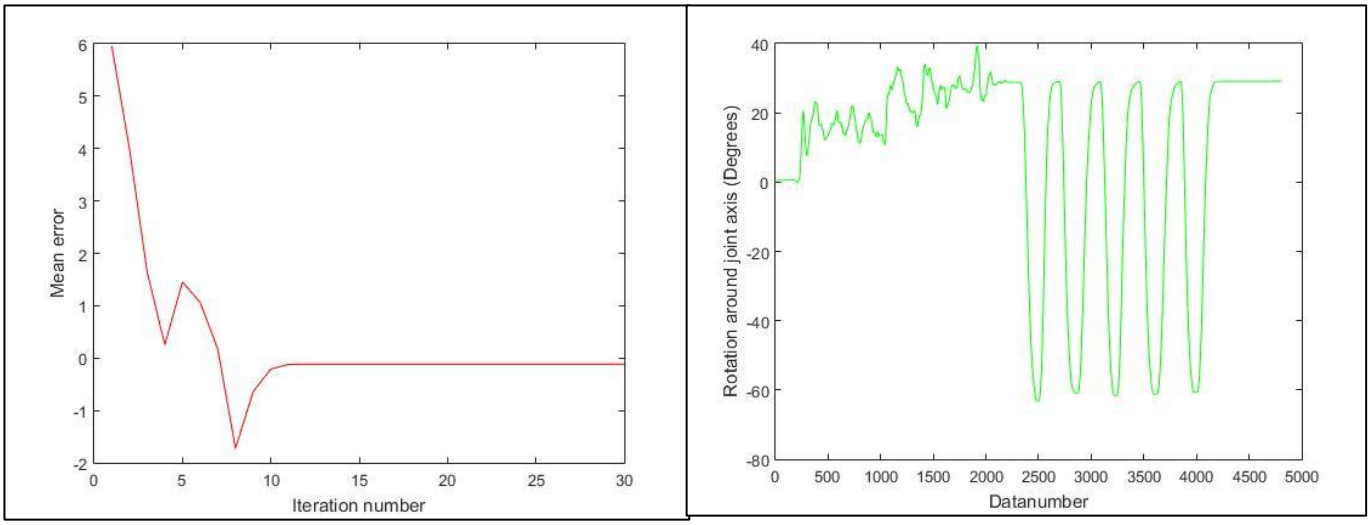

*Figure 22, **Results Test 2**, L: Progress of the mean error during 30 optimization iterations. R: Angular rotations of the total measurement, in here the difference in rotations when 2 segments rotate(0..2100) and if one segments rotates (2100 ..4900).*

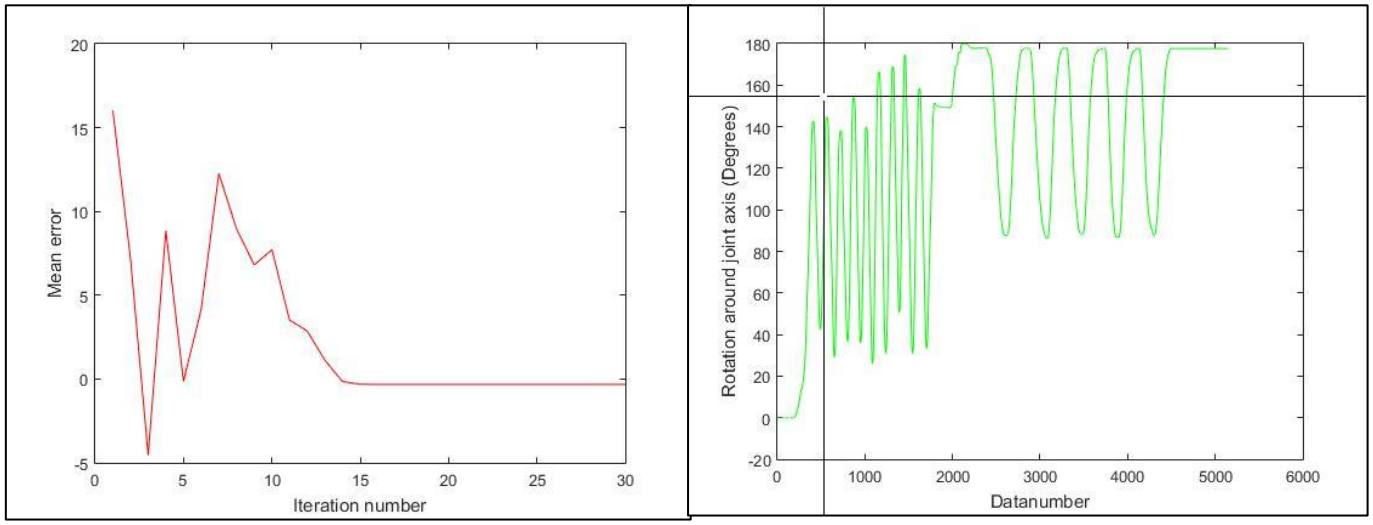

*Figure 21, **Results Test 3**, L: Progress of the mean error during 30 optimization iterations. R: Angular rotations of the total measurement, the cross reflects the function 'ginput()' which allows to select coordinates in the graph.*

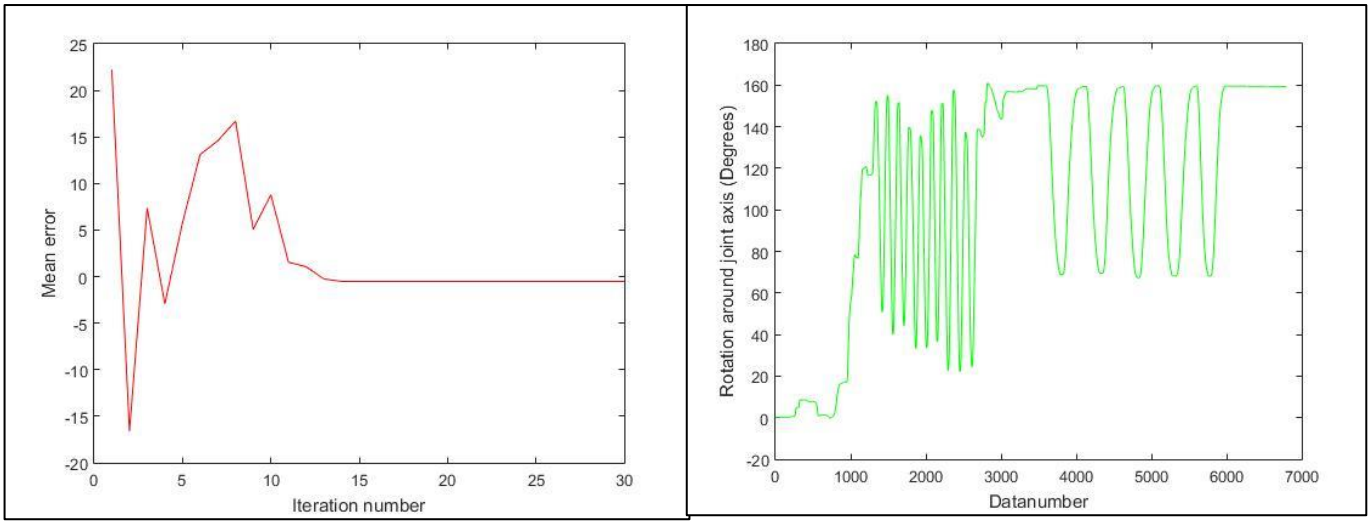

*Figure 20, **Results Test 4**, L: Progress of the mean error during 30 optimization iterations. R: Angular rotations of the total measurement.*

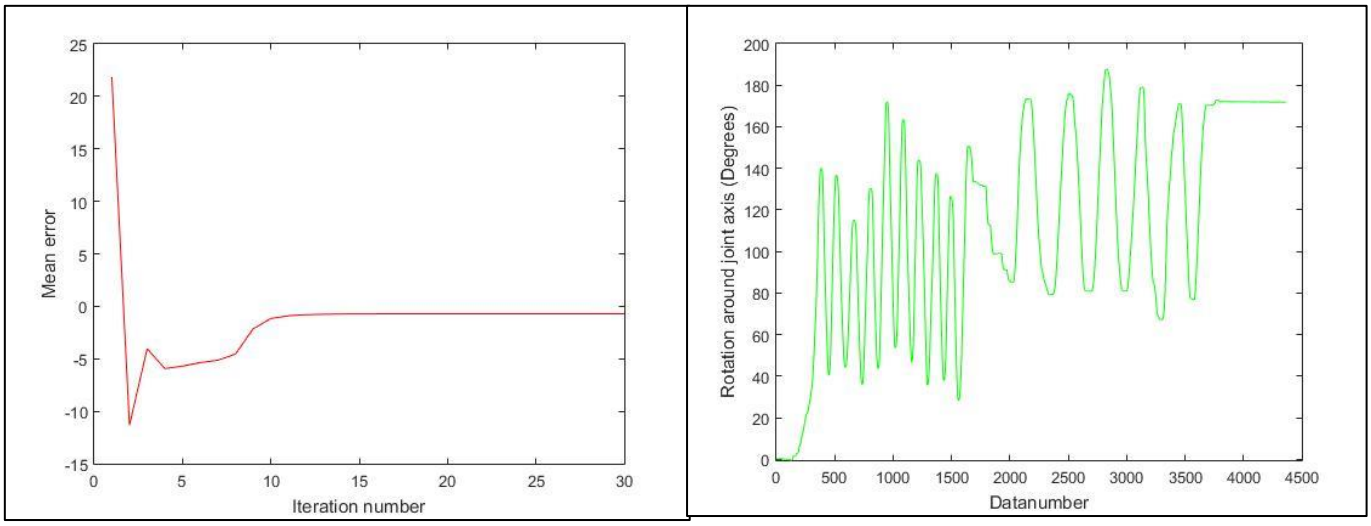

*Figure 25, **Results Test 5**, L: Progress of the mean error during 30 optimization iterations. R: Angular rotations of the total measurement.*

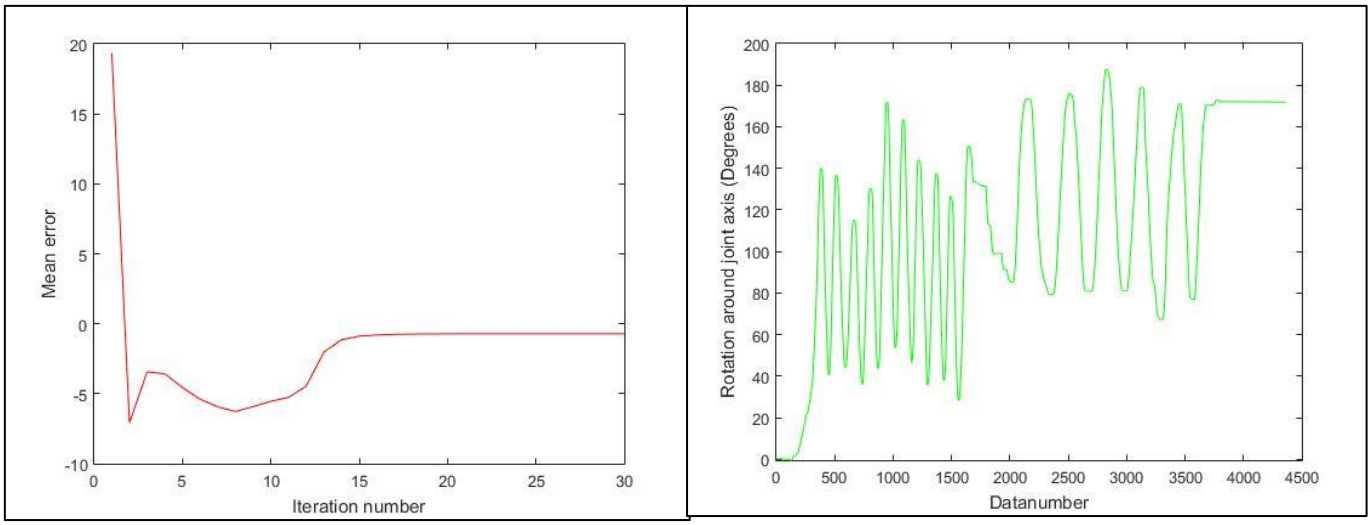

*Figure 24, **Results Test 6**, L: Progress of the mean error during 30 optimization iterations. R: Angular rotations of the total measurement. Different start-angles for J^1 and J^2 show a similar result as test 5, only it takes a bit longer to optimize.*

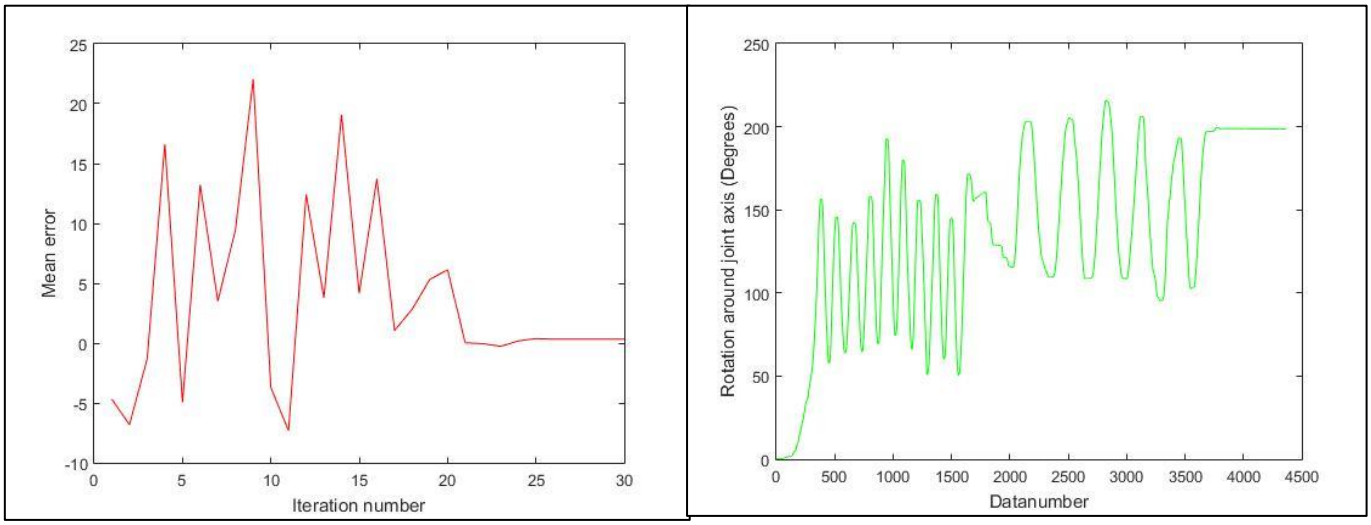

*Figure 23, **Results Test 7**, L: Progress of the mean error during 30 optimization iterations. R: Angular rotations of the total measurement. Less data-points for optimization show in the left graph that the optimization becomes unstable*

6.2 Results angular rotation test

Table 9, Overview of the Angular rotation test results.

Info IMU		
File(s)	LoggedData K1_	LoggedData K2_
Selected calibration data	1092	2109
Amount calibration data	204	
Initial X values	0,4 ; 0,2 ; 0,1 ; 0,7	
Iterations for J	12	
Determined J1, J2	-0,296	-0,011
	-0,113	0,011
	0,948	1,000
Synchronize point angle	2603	
Startangle	168,5	
Angles calculated for RMSE	512	

Info video	
File(s)	KinoveaTrajectory.txt
Amount tracking frames	511
Angle amount for RMSE	394
Synchronize point angle	610

Measurement		
	Video	**IMU**
Maximum angle	183,3	183,5
Minimum angle	33,3	33,1
Mean angle	108,6	108,8
Average angle deviation (deg)	0,18	
Number used for RMSE	392	
Sum squared deviation	1117	
RMSE	1,69	

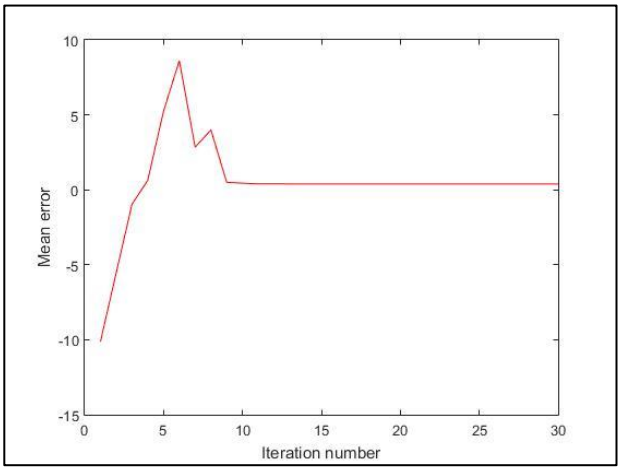

Figure 26, Plot of mean error during the 30 iterations of the optimization process. J1 & J2 are found within 10 iterations.

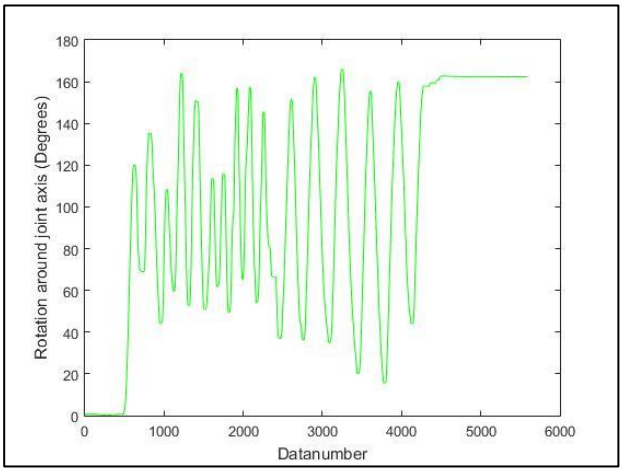

Figure 27, Plot of the angular displacement of the whole angular rotation measurement.

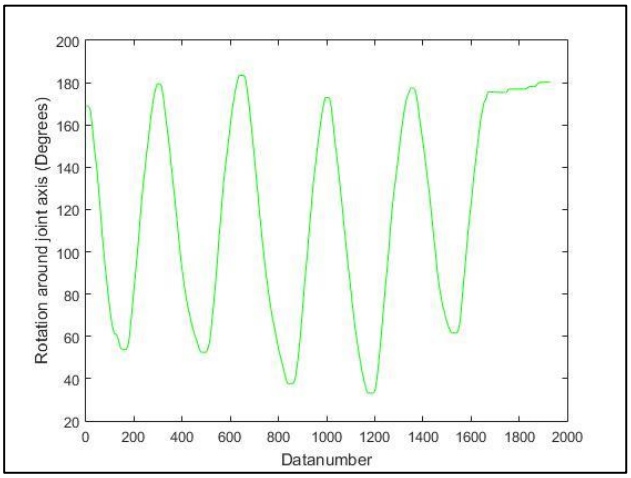

Figure 28, Plot of the angular rotation during the selected measurement data

6.3 Results bike test

Table 10, overview of the bike test results.

Info IMU		
File(s)	LoggedData BikeSb_	LoggedData BikeOb_
Selected calibration data	9952	11091
Amount calibration data	228	
Initial X values	0,4 ; 0,2 ; 0,1 ; 0,7	
Iterations for J	15	
Determined J1, J2	-0,382	-0,015
	0,293	-0,998
	0,877	-0,059
Synchronize point angle	9865	
Startangle	155,2	
Angles calculated for RMSE	272	

Info video		
File(s)	KinoveaTrajectory.txt	
	2222	2732
Amount tracking frames	511	
Angle amount for RMSE	272	
Synchronize point angle	91	

Measurement		
	Video	IMU
Maximum angle	157,7	157,3
Minimum angle	81,4	76,1
Mean angle	116,1	112,7
Average angle deviation	-3,38	
Number used for RMSE	272	
Sum squared deviation	5033	
RMSE	4,30	

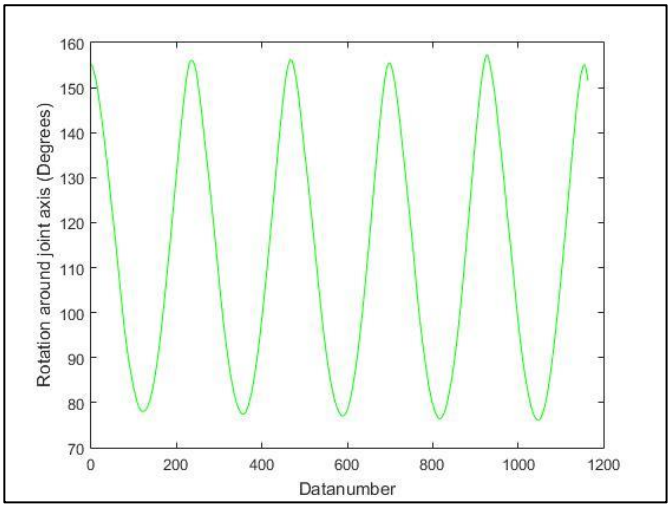

Figure 31, Plot of the angular rotation during the selected measurement data

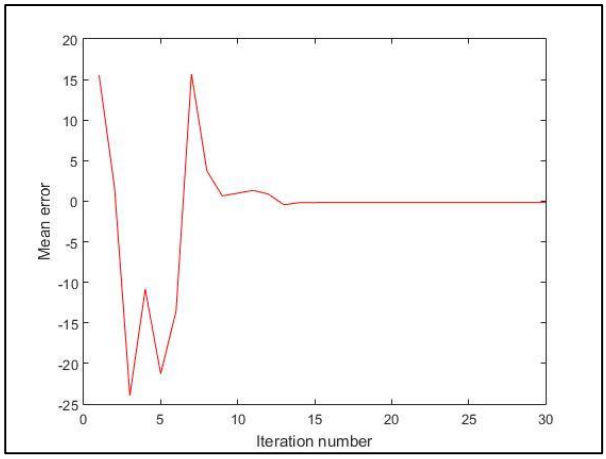

Figure 29, Plot of mean error during the 30 iterations of the optimization process. J1 & J2 are found within 15 iterations.

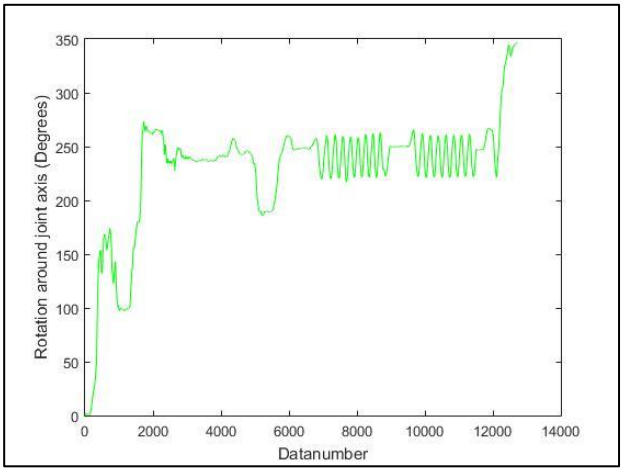

Figure 30, Plot of the angular displacement of the whole bike measurement. The angles are not correct due to a wrong sign.

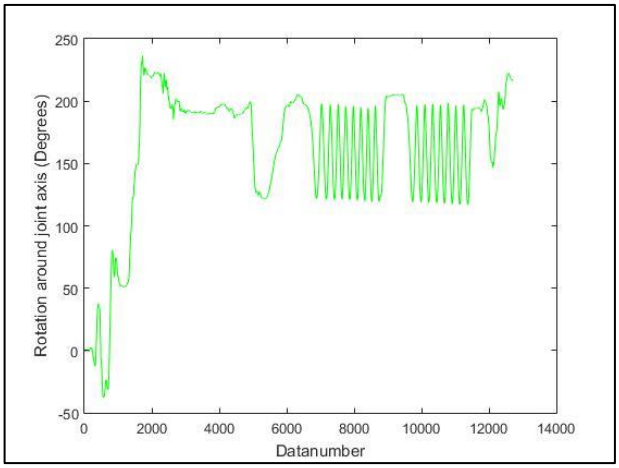

Figure 32, Plot of the right angular displacement of the whole measurement, after the sign was inverted

Appendix 7, Projectplanvoorstel

2017

Validation of IMU-based knee angle measurement during treadmill cycling.

JUNE 2017

A.P. VAN DER ZWET

1. Subject

Validation of IMU-based knee angle measurement during treadmill cycling.

Author

Arno van der Zwet

Student number

12104582

Date

Monday 12th September 2016

University & Study

The Hague University of Applied Sciences, Human Kinetic Technology

Work field

Sensors/sport innovation

Extern project

No

Reader

Docent supervisor: Daphne Wezenberg

Email: d.wezenberg@hhs.nl

E-mail

arnovanderzwet@gmail.com

Total ECTS up till period 12

58 ECTS

Table of content

2. Introduction

Internal Measuring Unit (IMU) based measurements in sports and daily activities are trending. These the latest developments in IMU allow the measurement devices to measure the least amount of movement in 3D of the human body by using three accelerometers, combined with three gyroscopes and a magnetometer.

Within every sport where sports gear is needed, companies are developing their gear to the highest standards. As the equipment evolves, so does the market of custom made sports gear which in cycling means getting a bike fitting. During a bike fitting the position of the cyclist on the bike is measured and changed by for instance adjusting the saddle height, width of the handlebar, length of the stem, or even inserting special shoe-inlays. A bike fitting is done to find the ideal bike position, which results in a greater power output on the pedals with the same physiological effort (heartrate or maximal muscle activity) and with less injuries, this is done by measuring the angles of the legs, arms, and torso, sometimes combined with physiological data like power output and heartrate. No standards in positioning are found because the position on the bike depends on the anatomical build of each individual athlete (muscle length and bone structure). All bike fitting-providers mostly use the same method: the movements of the cyclist are recoded by an 2D video capturing system or an 3D optical motion capturing system (Bioracer Belgium, 2016) (retul, 2016), while the cyclists ride their bikes that are fixed into a trainer (figure 1). This bike fixation is done to hold the athlete centered to the motion capturing system for higher accuracy, but this fixation of the bike is unnatural compared to a ride on the road, where the bike can move freely underneath the body. Another downside of the methods in bike fitting is that the power output during a bike fitting lasts only as long as the bike fitting takes, which mostly is 1,5 hours maximum. As cycling is an endurance sport, cyclists sometimes ride their bikes for even more than four hours, where exhaustion of muscles can appear, which may result in a different position on the bike. For instance when the adductor muscles of the upper legs get tired, the knees can move to a more lateral position and the load-distribution within the knee-joint changes to medial direction. The unnatural bike fixation together with the limitation of long-time muscle exhaustion, makes a new method of measuring the cyclists position on the open road beneficial.

Figure 33, Tacx trainer
source: http://www.wigglestatic.com/product-media/5360075251/T2600_Tacx_Blue_Motion_trainer_back_1207.jpg?w=2000&h=2000&a=7

For developing a method that can be used to measure body movements during cycling on an public road, it is necessary to first validate an internal measuring system that can be worn by the cyclist. An internal measuring system contains Internal Measuring Units (IMU's) that can be used for recording accelerations (accelerometers) and angular movements (gyroscopes) in all three planes x, y and z. Although no studies have been found which contain knee angle measurements during cycling, some studies with other activities like walking and jumping (Glen Cooper, 2009; S.A.A.N. Bolink, 2016; J. Favre, 2009) show that angles can be determined using IMU's and have a great correlation with 3D optical motion capturing systems like Optitrack. However one study in walking (Glen Cooper, 2009) shows that the correlation of the determined angles gets less, as the speed of the activity rises. Because the knee makes the biggest angular movement and acceleration during cycling, the knee movement is chosen for this study. The goal of this study is therefore answer the main question: Can IMU's be used to measure the absolute angular changes, and movements of the knee in all 3 planes during treadmill cycling, and at what cadence will measurements still be valid?

Optitrack is determined as golden standard for motion analysis (Farrokh F. Mohammadzadeh, 2015) and will be therefore suitable for validating the angle measurement and knee movements, done by the IMU's during cycling.

3. Method

Within this study five healthy subjects, aged between 18 and 65 years old, which are in possession of a bike with cadence sensor, will participate in a measurement just like a bike fitting. Because Optitrack will be used as golden standard for validation the bike needs be fixed onto a Tacx Flow trainer and will thereby reduce the lateral movements of the bike and test subjects, which also makes the determination of the absolute knee angle in the sagittal plain easier. The bike will be placed within the range of the Optitrack system for synchronized measurements with the IMU's. Each test subject will ride their bikes for a total of five minutes, with the first minute as warm-up. Because this study requires different kind of speeds for the knee to move, the test subjects will ride their bikes with four different cadence speeds (50, 70, 90, and 110 RPM), each lasting for one minute. This allows the determined absolute knee angle to be validated at different kind of cadences. The resistance during this study is not significant as long as the test subject can reach the maximum cadence, the Tacx Flow will be set on the standard slope of 0%.

Figure 34, yellow stars show the location of the x-imu's on the left leg

The IMU's that will be used for this study are four x-IMU's (x-IO) which have a dimension of 57x38x21 mm, and can measure accelerations and rotations in three planes x, y and z. The x-IMU's will be fixed onto four different places on the left leg (figure 2). On the lower leg two x-imu's are placed on 1/4 ad 3/4 of the length on a virtual line from the lateral Malleolus of the ankle to the caput Fibulae. The two remaining x-imu's are placed on 1/4 and 3/4 of the length on a virtual line from the lateral epicondyle of the knee to the trochanter major.

The angle can then be determined within a Python program by the position of the four x-imu's, and by Euler decomposition which uses the place and direction of two x-imu's (one on the lower leg, one on the upper leg). Because the angles will be determined by using gyroscopic and accelerometer data in planes x, y, and z, Python also will be used to filter out the lateral and vertical accelerations produced by the high cadence circular movement of the fixed foot, which should result into a clear angular change of the knee. The absolute knee angles calculated with each method will be compared to the Optitrack angles and be validated using SPSS.

There are two sub-questions to be answered during this study:

- What is the difference in angle determination between different kind of cadences?
- How many IMU's are necessary for a valid knee-angle measurement and where on the leg do they need to be fixed?

Hopefully the average correlation value of the validation will be > 0.8 which means that the angular determination by the x-IMU's are valid, but expected is that the lower cadences (50 & 70 RPM) show a bigger correlation than the higher cadences (90 & 110 RPM) as in the previous study on walking speeds (Glen Cooper, 2009).

4. Planning

At the start of the project the first chapter (introduction) of the report is almost finished. The upcoming 14 weeks after the presentation of the project plan will be mainly consist out of programming in Python and adjusting the method. After that it's testing the code, executing tests with the test subjects and writing the report after analyzing the test results. The general overview of the planning can be found in Apendix II and will be a guideline for the project to finish on time. Beside this project there is no other schoolwork which may endanger the planning.

Appendix I – Reference List

Bioracer Belgium. (2016). *bioracermotion.com*. Retrieved from www.bioracermotion.com: http://bioracermotion.com/?page_id=7

Farrokh F. Mohammadzadeh, S. L. (2015). Feasibility of a Wearable, Sensor-based Motion Tracking System. *Procedia Manufacturing, Volume 3*, 192-199. Retrieved 5 2016

Glen Cooper, I. S. (2009). Inertial sensor-based knee flexion/extensio angle estimation. *Journall of Biomechanics 42*, 2678-2685.

J. Favre, R. A. (2009). Functional calibration procedure for 3D knee joint angle description using inertial sensors. *Journal of Biomechanics, Volume 42*, 2330-2335.

retul. (2016, 5). *www.retul.com - Vantage 3D Motion Capture System*. Retrieved from www.retul.com: https://www.retul.com/retul-products/vantage-motion-capture-system/

S.A.A.N. Bolink, H. N. (2016). Validity of an inertial measurement unit to assess pelvic orientation angles during gait, sit–stand transfers and step-up transfers: Comparison with an optoelectronic motion capture system. *Medical Engineering & Physics, Volume 38*, 225-231.

Appendix II – Planning

Week		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Planning	**< 38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**1**	**2**
Concept project plan	< 12-sep																
Final project plan	12-sep																
Presentation project plan	19-sep																
Programming phase																	
General python code																	
X-imu read code																	
Trial testing X-imu																	
Angle filter code																	
Real-time data stream																	
Method phase																	
Fine-tuning method																	
Trial measurement																	
Testing phase																	
Testing subjects																	
Analyzing results																	
General																	
Meeting with the supervisor																	
Writing report																	
Hand in thesis															28-dec		
Defense preparation																	
Defense																	

Appendix III - Personal learning goals

Working according to a plan:

During my last internship I was working with a strict schedule, but found that really difficult. During execution of my project I will have to do every task that is planned for that specific day or week. The planning from appendix II will be the weekly guideline but I am going to make planning's for each day as well.

Writing an article:

In the past I have only written one introduction for an article during an internship. But performing a study on my own and writing an article about that subject will be a first for me. This project is therefore the ideal way to practice my writing skills, as well as documentation which need to be flawless.

Programming:

During the Human Kinetic Technology study I have encountered several programming languages like Matlab and C. But in our future work field there is another language called Python, that is also usable for data-management. Writing a python program for the x-imu data will not only increase my knowledge about Python itself, but also improve my general program skills.

Appendix 8, Evaluation personal learning goals

Working according to a plan:
During my last internship I was working with a strict schedule, but found that really difficult. During execution of my project I will have to do every task that is planned for that specific day or week. The planning from appendix II will be the weekly guideline but I am going to make planning's for each day as well.

Evaluation:
Working according to plan is clearly something I have to improve, due to the delay of this project. Working and studying will go together, but only if there are strict planning's or day-/week-goals. Evaluating this goal makes me eager to set new ones for myself in the future, regarding project planning.

Writing an article:
In the past I have only written one introduction for an article during an internship. But performing a study on my own and writing an article about that subject will be a first for me. This project is therefore the ideal way to practice my writing skills, as well as documentation which need to be flawless.

Evaluation:
This thesis was an excellent way to train my writing skills. Because my native langue is Dutch, writing the thesis in English made is extra challenging. Also the methodology of the algorithm was a good way to practice my writing skills. Explaining the method is what I found extremely difficult.

Programming:

During the Human Kinetic Technology study I have encountered several programming languages like Matlab and C. But in our future work field there is another language called Python, that is also usable for data-management. Writing a python program for the x-imu data will not only increase my knowledge about Python itself, but also improve my general program skills.

Evaluation:

During this study I was able to master Python and its syntax. After writing the algorithm in python and Matlab, I have now much more experience in how to set up programs for research.