Position and orientation determination of a probe with use of the IMU MPU9250 and a ATmega328 microcontroller

Charlotte Treffers
Luc van Wietmarschen

June 15, 2016

TU Delft

Position and orientation determination of a probe with use of the IMU MPU9250 and a ATmega328 microcontroller

by

Charlotte Treffers
Luc van Wietmarschen

June 15, 2016

Students:	Charlotte Treffers	4270134
	Luc van Wietmarschen	4313496
Project duration:	April 18, 2016 – July 1, 2016	
Supervisor:	dr. Marco Spirito,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Abstract

This paper gives an overview of the position and orientation determination system made in context of the final project of the bachelor Electrical Engineering at Delft University of Technology. By combining the measurements of the accelerometers, gyroscopes and magnetometer from the Intertial Measurement Unit MPU9250 in a complementary filter, the orientation of the MPU9250 was determined. The displacement of the MPU9250 was determined by using the trapezoidal integrating method to integrate the acceleration measured with the accelerometers. This displacement was combined with the orientation to determine the position of the MPU9250. With the knowledge of the relative orientation and distance between the MPU9250 and the probe-tip, the orientation and the position of the probe-tip was determined with use of quaternions. Pitch and Roll angles were determined with an accuracy of 2 degrees. The yaw angle determination was incorrect due to the inaccuracy of the magnetometers. The position determined did not drift, but was off due to imprecise filtering and integration. For better results a microcontroller with more programmable space is suggested.

CONTENTS

1. Introduction

A. Tissue Imaging Probe System

This paper will cover the position and orientation system of the *Tissue Imaging Probe System* (TIPS). The TIPS was made in context of the finale project of the bachelor Electrical Engineering at Delft University of Technology. The TIPS will provide a method to detect skin cancer without relying on visual inspection. To do this the permittivity of the skin is measured with use of the TIPS. In short the TIPS works by placing the probe on the skin. Then EM-waves with different frequency are send to the skin and the reflections of these EM-waves are measured. The permittivity will then be determined with use of an algorithm [1]. The visualisation-software combines the permittivity-measurements and the location and orientation of the probe to make a 3D-sketch. The doctor can now easily see if there is a skin cancer cell and where this cell is located. The position and orientation system described in this paper will track the position and orientation of the probe during the permittivity-measurements and will provide this information to the visualisation-software. The system needs to satisfy the requirements set in section 2.

B. State of the art

There exist a lot of possible ways to determine the position of an object, each with its own advantages and disadvantages. There are two ways to give a position, the absolute position or the relative position to a starting point. To determine absolute position an observer is needed for reference, such as a camera or for example in GPS a satellite network.

Absolute position detection is already done in commercially available products, such as the Microsoft Kinect [2], and poses a method with a sufficient accuracy for this project. However, the object needs to be visible to the observing device during measurements. This could be inconvenient for the doctor and patient, especially since the probe-tip needs to be known which will be very close to the body and thus the doctor will easily be between the camera and the probe-tip. Another option would be to make a marker on the far side of the probe where the line of sight of the camera is less blocked by the doctor and patient. However, this would require a fairly large handle to make sure the line of sight is kept and that would not make the probe very user-friendly.

An already used way to determine equipment locations in medical care is trough magnetic resonance imaging [3]. However, this method requires large and expensive machinery that also influences the magnetic field and would thus influence the measurements of the probe. Due to these disadvantages absolute positioning is not used. Relative positioning will also work for the end result and will be less expensive.

To measure relative position one only needs to detect change in position from a starting point. For this product the starting point is set when the instrument is touching the subject and the operator (doctor) presses a button on the instrument. Two ways for determining relative position were found to be of a reasonable price and implementation complexity: optical mice systems and an IMU-system containing accelerometers, gyroscopes and magnetometers.

The use of an optical mouse sensor will give a two dimensional displacement of the probe. There already exist methods for optical mouse position determination [4]. This requires to place a sensor on the subjects surface and thus require the sensor to be as low as the probe-tip. This would make the probe wider at the tip and unsuitable to move across small surfaces or sharp curves. On top of that there would be two optical sensors needed, since the sensor cannot be placed on the tip of the probe and the exact position of the probe can only be guessed to be to the side of the optical mouse. Also when the probe rotates the mouse sensor only sees a change in position, but the position of the probe-tip stays the same. This problem can be solved with using two optical mouse sensors on both sides, but this make the probe even wider.

By using an IMU-system with accelerometers, gyroscopes and magnetometers one can measure the relative position and orientation of an object in all three dimensions [5]. With those parameters known one can calculate any other point on the device attached to this sensor. This will give the possibility to place the IMU away from the probe-tip (although not too far to conserve accuracy). This will allow

the probe to be placed on the subject without any other materials near the probe-tip. Thus this will not influence the measurement and will allow the probe-tip to be moved along sharp curves. On top of that these sensors are small, cheap and easily integrated with other electronics.

Thus the IMU-system containing gyroscopes, accelerometers and magnetometer will be used in this paper as a basis to track the position and orientation. This position-system must satisfy the requirements given in section 2. In section 3 the design based on the position-system will be given. In sections 4 up to and 10 the different elements of this design are explained. In these sections the implementation of these elements are given and tested. Section 11 gives an overview of how those elements are implemented together on the microprocessor. After testing the individual elements the whole position-system is tested and the results are given in section 12. The results of the testing are discussed in section 13. To end with there will be given a conclusion in section 14.

2. Programme of Requirements

The position-system will track the movements and orientation of the probe and return an accurate position and orientation to the visualisation-software.

A. Functional requirements

[1,1] The position-system must give the position of the probe to the visualisation system.
[1,2] The position-system must give an orientation of the probe to the visualisation system.
[1,3] The position-system must work during a measurement performed by a doctor in a hospital or practice.

B. Environmental compatibility

[2,1] The position-system must not influence the surrounding equipment.
[2,2] The position-system must not have any negative side effects on the user and on the patient.

C. System requirements

1) Utilisation features:

[3,1,1] The position-system must give a position in x,y,z in cm to the visualisation-software.
[3,1,2] The position-system must give the position with an accuracy of 1 cm.
[3,1,3] The position-system must give the orientation in a normal vector to the visualisation-software.
[3,1,4] The position-system must give the orientation with an accuracy of 2 degrees.
[3,1,5] The position-system must be insensitive to the rotation of the probe.
[3,1,6] The position-system must not influence the permittivity measurement.
[3,1,7] The position-system must be user-friendly.
[3,1,8] The position and orientation tracking must start/stopped when a button is pressed.

2) Production and putting into use features:

[3,2,1] The position-system must be attached to the probe, but must not limit the movability of the probe.
[3,2,2] Matlab must be pre-installed on the user's premise and must have the needed license.
[3,2,3] The handle must be printable by 3D-printer and made to be assembled by hand.
[3,2,4] The position determination device, including a button, must fit in the handle.
[3,2,5] The operational software must be installed on the microcontroller ATmega328.
[3,2,6] The wiring of the device must be done before delivery.
[3,2,7] The system may not stop sending data within 12 hours of active use.

3) Discarding features:

[3,3,1] It must be possible to dismantle the device by hand.
[3,3,2] The microprocessor must be reprogrammable and reusable.

D. Testing Requirements

[4,1] All subsystems must be able to communicate with each other.
[4,2] All filters will not filter out data that is needed for the accuracy described in requirements [3,1,2] and [3,1,4].
[4,3] All filters will filter out distortions that would otherwise cause inaccuracy greater than described in requirements [3,1,2] and [3,1,4].
[4,4] All offsets needs to be completely removed, the mean after removal must be zero.

E. Development of manufacturing methodologies

[5,1] The position-system must run on a microcontroller ATmega328. The ATmega328 must pass the position of the probe to the visualisation-software, which is written in Matlab.
[5,2] The operational software must be developed in standard ANSI C. The employed compiler must be validated according to the ANSI/ISO/IEC standards.

F. Business strategies, marketing and sales opportunities

[6,1] The product must be delivered and installed within 3 months when ordered.

3. Design overview

A. Overall design

The design of the position-system is based on an *Intertial Measurement Unit* (IMU) with gyroscopes, accelerometers and magnetometers. For this project the MPU9250 from InvenSense is used. The MPU9250 is discussed in section 3-C. The MPU9250 is placed at the side of the probe, a short distance from the probe-tip. The MPU9250 is not placed too close to the tip to meet requirement [3,2,1], which states that the position-system may not limit the freedom of movement of the probe. The MPU9250 and the probe are placed in a handle which also contains a button. The 3D design of the handle is presented in section 3-B.

The button will be used to start and stop the tracking of the position, meeting requirement [3,1,8]. The position of the probe-tip at that start moment will be the reference point to the positioning measurement. In figure 3.1 an overview of the position-system is given. The MPU9250 will be connected to the microcontroller (ATmega328, Arduino Nano, requirement [5,1]) with use of the I^2C-bus. The button will also be connected to the microcontroller. The microcontroller calculates the orientation of the probe and position of the probe-tip. After the calculation the orientation and position will be given to the visualisation-software, meeting requirements [1,1] and [1,2]. Section 3-D gives a more detailed overview of the design.

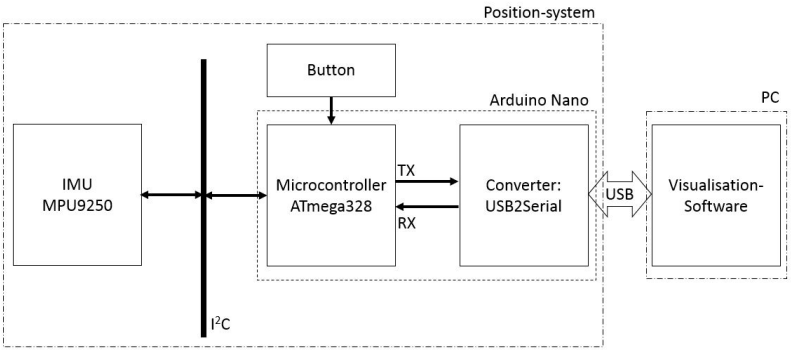
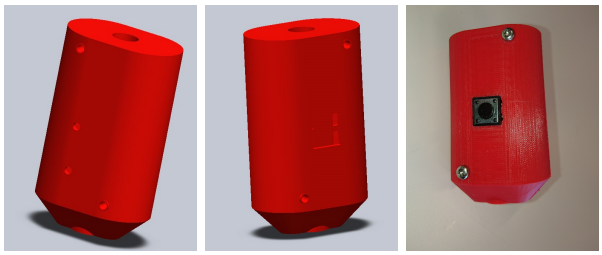


Figure 3.1: Large overview of the position-system

B. 3D model of handle

The MPU9250 and button will be placed in a handle, which is designed using Solidworks. The handle encloses the probe and therefore contains a hole where the probe can pass trough. To attach the MPU9250 to the handle there are two screw holes and to attach the button there is a square hole, this to meet requirement [3,2,4]. The handle was 3D printed (requirement [3,2,3]) in two parts, this is to allow the user to place the components at the inside. The two parts can be put together by two screws, allowing for assembly by hand. The Solidworks model and 3D printed handle are shown in figure 3.2.



(a) Solidworks model back side (b) Solidworks model front side (c) 3D printed handle

Figure 3.2: Solidworks model and 3D print of handle

C. MPU9250

For the position-system based on a IMU-system, containing accelerometer, gyroscopes and magnetometers, the MPU9250 will be used (figure 3.3[1]).

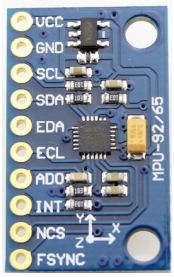


Figure 3.3: MPU9250

The MPU9250 is an IMU that combines the MPU6050, which contains accelerometers and gyroscopes in a single chip [6], and the AK8963, which is a 3-axis digital compass (also known as magnetometers) [7]. The MPU6050 has an onboard *Digital Motion Processor* (DMP). The DMP processes the complex 6-axis MotionFusion algorithms. Those algorithm can calculated the orientation of the MPU9250, but only uses the accelerometers and gyroscopes, to do this. Because it doesn‘t use the magnetometers the yaw angle is not calculated very accurate and therefore the DMP is not used in this project. In section 7 it is explained why the measurements of the magnetometers are needed to calculated the yaw angle. Further the MPU6050 works with a 1024 byte *First In First Out* (FIFO) buffer and the MPU9250 can communicate with the ATmega328 with an I^2C bus at 400kHz. The ATmega328 can read out the measurements of the MPU6050 by using the FIFO buffer and read out the AK8963 by enabling a bypass, this will later be made clear in section 4. In appendix A in figure A.1 the Block Diagram of the MPU6050 given by InvenSense is shown.

The MPU9250 uses 16-bit *analog-to-digital converters* (ADC's) for digitizing the gyroscopes, accelerometers and magnetometers outputs. The gyroscopes of the MPU9250 has an adjustable full scale range. Dependent on the full range scale the sensitivity of the gyroscopes are between 16.4 and 131 $LSB/°/sec$, corresponding with a sensitivity of between 0.00763 and $0.06098°/sec$ (LSB meaning *Least Significant Bit*). The full scale range of the accelerometers are also adjustable and the sensitivity is between 2048 and 16384 LSB/g, corresponding with a sensitivity between 0.00060 and $0.00479m/s^2$. The magnetometer has a full scale range of +/- 4800 μT and a sensitivity of 0.6 $\mu T/LSB$.

There is an ATmega328 library provided by J. Rowberg to readout the MPU9250 with use of the I^2C bus [8]. To use this library it is necessary to use the I2C library as well [9]. The MPU9250 library helps to initialize the I^2C bus and give different functions to read out the FIFO buffer. Also the library provides some functions to help with 3D position calculation, some of those will be used to make the implementation of the calculation of the position and orientation of the probe, described in section 8-D. A list with used libraries can be found in appendix C. More about the I^2C bus and readout of the sensors can be found in section 4.

The reason to select this IMU over others is because of its wide implementation in combination with the ATmega328, meaning that they work together as demanded by requirement [4,1] and there are many examples and libraries that can be used for our needs. The specifications also indicate it is accurate enough with respect to requirements [3,1,2] and [3,1,4]. On top of that it is small enough to fit in the handle as demanded by requirement [3,2,4].

[1]Figure from http://www.hotmcu.com/9dof-imu-module-with-mpu9250-p-172.html

D. System overview

In appendix A in figure A.2 the pin diagram of the position-system is shown. The MPU9250 is connected to the ATmega328 with a level-shifter in between. This is done because the MPU9250 works at a DC voltage of 3.3V and the ATmega328 works at a DC voltage of 5V. The button is also connected to the ATmega328 and uses the pull up resistor of the Arduino Nano.

The calculation of the orientation and position of the probe are done by the ATmega328, as determined by requirement [1,1] and [1,2]. In figure 3.4 an overview of the different elements of the calculation is given. First the values of the MPU9250 are read out with use of the I^2C bus, see section 4. The values of the measurements are noisy, especially from the accelerometers and magnetometers. To prevent the noise from affecting the position and orientation calculation the values of the sensors are first noise filtered, using the noise filter described in section 6. After the noise filtering the filtered values are combined in the complementary filter to gain the orientation of the MPU9250. The complementary filter is described in section 7. Thereafter the noise and high-pass filtered measurements of the accelerometers and the orientation of the MPU9250 are combined to calculate the position and orientation of the probe-tip, see section 8. To calculate the position and orientation of the probe-tip, the distance between the probe-tip and the MPU9250 and relative orientation between the probe-tip and the MPU9250 must be known, section 9 describes an algorithm to determine this distance and orientation. Section 10 describes how the orientation and position of the probe is given to the visualisation-software. In section 11 is described how all the named sub-elements work together in code that will be programmed on the ATmega328.

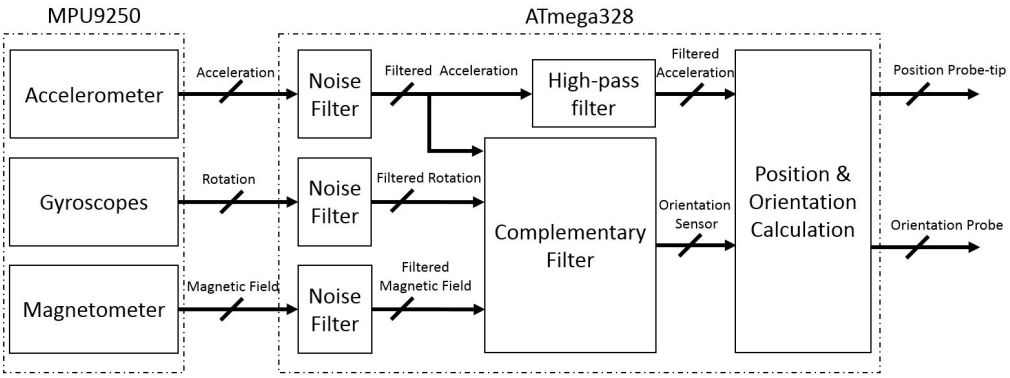


Figure 3.4: System overview

4. I2C AND SENSOR READOUT

A. Theory

The ATmega328 communicates with the MPU9250 with use of the I^2C bus. I^2C is a serial protocol for a two-wire interface [10]. The I^2C communication makes use of the *Serial DAta* (SDA) and *Serial CLock* (SCL) pins of the MPU9250. The SCL provides the serial clock and the SDA provides the serial data. The address of the MPU9250 is set by pin AD0, this address is 0x68 when the pin is connected to ground, which is done in this project. The address of the AK8963 is by default 0x0C. Before the communication starts the SCL and SDA are both high, caused by pull-up resistors. These pull-up resistors are placed to make certain both pins become high when not being defined by the ATmega328, this will prevent communication problems when the ATmega328 is waiting for incoming data.

When the communication between the ATmega328 (master) and the MPU9250 (slave) starts the *start condition* is created. This is done by making SDA low, see figure 4.1[2]. Then the serial clock (SCL) begins to generate clock pulses and the first byte is send by the master. This first byte contains the 7-bit address of the slave and a read/write bite. When the read/write bit is 0 the master will write to the slave and when the bit is 1 the master will read from the slave. The bits are send when the serial clock is high. When the serial clock is low the bit value are allowed to change. The number of bytes sent is not limited but each byte must be followed by an acknowledge bit, see figure 4.1. When the master is done writing/reading then the stop condition is created, see figure 4.1.

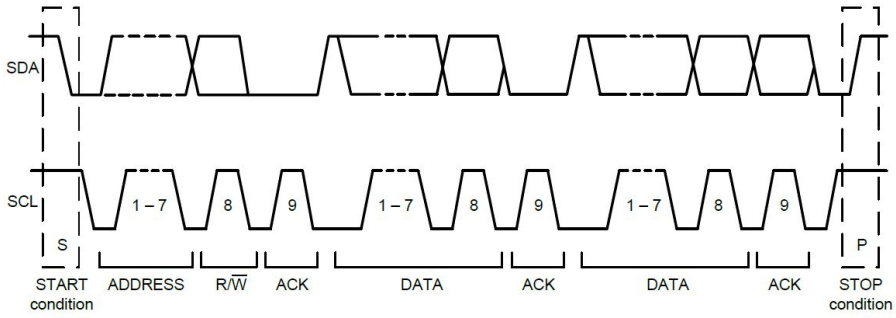


Figure 4.1: I^2C protocol

B. Implementation

The I^2C communication is implemented by using the I2Cdev library [9] and the MPU9250 library from J.Rowberg [8]. With use of especially the functions writeBytes, writeWords, readBytes and readWords from the I2Cdev library the I^2C is initialized and used to read out the MPU9250.

To collect data from the MPU6050 the ATmega328 first indicates which registers are needed to be read out. The MPU6050 then puts these registers bytes in the FIFO buffer to be accessed by the I^2C bus. The ATmega328 then reads out the *First In First Out* (FIFO) buffer and puts the right bytes in the corresponding variables. A bit shift operation is required to gather the data since the measurements are stored as 16-bit values divided into two bytes, the higher byte must be shifted 8 bits to make room for the lower byte in the 16 bit space that is allocated on the ATmega328 for the measurement.

Next is the readout of the magnetometers, located on the AK8963. To interact with the AK8963 the MPU9250 must allow the AK8963 to directly connect to the I^2C bus, this is done by enabling the I^2C bypass on the MPU9250. After this the AK8963 is activated by setting its mode to *single readout mode*. In this mode the AK8963 will do one readout, put that in its readout registers and ends by putting itself into *sleep mode* again. Next time the ATmega328 accesses the MPU9250 the I^2C bypass is automatically disabled, so this needs to be enabled every readout of the AK8963. The decision to use the single readout mode on the AK8963 is also chosen because of the automatic resetting of the I^2C bypass by the MPU9250. The AK8963 also contains a continuous readout mode, but that requires the

[2]Figure from datasheet MPU9250 [10]

I^2C connection to remain open, thus it is not possible to read out both the MPU9250 and AK8963 in continuous mode. If the AK8963 is left in continuous mode while the I^2C connection is terminated it will overflow its own buffer and it becomes unpredictable what value will be read out when accessed. One small advantage to the single readout mode is that the readout frequency of the MPU6050 and AK8963 are synchronised since the AK8963 data is only read when the MPU6050 has new data ready. The exact code implementing this can be found in the MPU9250.h library.

C. Testing

To test if the I^2C communication works, the value of the accelerometer, gyroscopes and magnetometers were read out and printed on the serial monitor of de arduino interface. When reading out the data the MPU9250 was turned in various directions, to check if the readouts were as expected. The output of the accelerometers, gyroscopes and magnetometers corresponded with the expected values. This is as required by requirement [4,1] and therefore it was concluded that the I^2C communication works.

5. Calibration

A. Theory

In this section the accelerometer, gyroscope and magnetometer offsets are explained. Other errors, such as drift and noise, are explained and corrected in section 6 and section 7.

Offset is defined as the DC offset of the measurement. The DC component is the mean value of the waveform. If this mean value is zero, there is no DC offset. All three sensor types use *Micro Electrical Mechanical System* (MEMS) hardware, which is less precise than for example ring lasers. Which are not used because of requirement [3,2,4], since they are too large. This hardware means that even when the device is lying still the accelerometers, gyroscopes and magnetometers still output a value other than zero. Each device has its own reasons to have this offset and sometimes it is even the offset that must be monitored for orientation determination. Each device therefore needs its own offset calculation and correction method.

1) Gyroscope offset: Gyroscopes have the simplest offset form of the three methods. It is simply due to manufacturing and the fact that MEMS hardware is not so precise. It is also easy to measure and correct. The device is locked in place so that it doesn‘t move and should therefore measure zero angular velocity. If the raw data does have a non-zero DC component, it is the offset of the gyroscope. To correct for these offsets these non-zero values must always be subtracted from the raw data. In this way the waveform is centred around the time-axis and all reading correspond to movements (or other forms of noise).

2) Accelerometer offset: For the accelerometers the offset is quite large, because there is always the gravitational field of the earth which is measured by the accelerometers. This gravitational pull is however known, namely the gravitational constant, and by placing the MPU9250 level with upwards facing z-axis this gravitational offset can be easily subtracted. After this and locking the MPU9250 in place, so that there is no movement, there will still be a DC offset noticeable in the raw output as a mean in the output waveform. This offset must then always be subtracted from the raw accelerometer output to get accurate data, even during movements. For orientation determination the accelerometer data is used to measure gravity. In this case the DC offsets must still be subtracted, but the gravity component should be left standing. This is further explained in section 7. In case of the displacement determination the gravity should also be subtracted from the measurements of the accelerometers, to gain the real acceleration of the MPU9250, this is explained in section 8.

3) Magnetometer offset: Similarly to the accelerometers the magnetometers always sense a, smaller but still noticeable, offset. The magnetometers will only be used to determine orientation and for that they need to sense the magnetic field of the earth. This earth magnetic field has a approximate strength of $40\mu T$ in The Netherlands. Most electrical devices contain components that interact with this field. This poses a problem when trying to precisely measure the earth magnetic field. These disturbances are divided into two groups, the hard iron losses and the soft iron losses [11]. The hard iron losses are due to the chips permanently magnetized components and soft iron losses are due to the geomagnetic field inducing currents onto normally unmagnetized components. The hard iron losses are easy to be calculated and corrected, because they are constant regardless of orientation and position of the device. One easy way to determine this offset is to measure the raw data from the magnetometers when the device is lying level on the surface before and after a rotation of 180 degrees. The hard iron losses are turned with the device, whereas the earth magnetic field stays in the same reference frame. Thus one first measures the magnetometer offset and the earth magnetic field in positive direction and after the turns measures the magnetometer offset with the negatively oriented earth magnetic field. The sum of the two reading is twice the magnetometer offset. This is illustrated in figure 5.1 and the following equations:

$$A = Device field + Earth field \tag{5.1a}$$

$$B = Device field - Earth field \tag{5.1b}$$

$$A + B = 2 * Device field \tag{5.1c}$$

Here the device field is the hard iron offset that must always be subtracted from the raw data to measure the magnetic field apart from the device. Doing this for each axis gives the corresponding hard iron offsets.

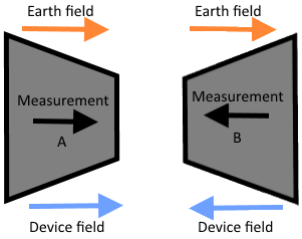


Figure 5.1: Illustration of hard iron losses determination

The soft iron offsets are harder to detect and correct for because they vary with the orientation of the device. The magnetic field of the earth is enhanced or blocked out by different types of metal in the chip. These metals rotate along with the chip, but the earth magnetic field does not, resulting in different offsets in different orientations. These soft iron losses result in a scaling error in each magnetometers data respectively to each other. One method to measure and correct for this error is to rotate the device around all axes and plotting the measurements of the magnetometers on each corresponding axis creating a ellipsoid of the measured magnetic field. This ellipsoid is due to the different scaling of the magnetic field in the chip, where a certain orientation induces a stronger field than others. To correct the error, the difference between the ellipsoid and a perfect sphere must be calculated. This gives scaling for all three axes and thus for the magnetometers. This scale must always be applied to the raw data to compensate for the soft iron losses. To calculate this scale the ellipsoid must first be aligned with the magnetometer axes. This is done by finding the major axis of the ellipsoid and aligning it with one of the axes using a quaternion rotation. After this the scaling can be reverse calculated from the difference between the ellipsoid and a perfect sphere. This scaling is the ratio between the major and minor axis for each magnetometer axis. After all these scales are found they must be rotated back into the actual measurement frame with the reverse quaternion rotation that was done to align the major axis with one of the magnetometer axes. To get the corrected data from the raw data, the raw data must be rotated using the previously found quaternion, then scaled using the found scales and rotated back into the measurement frame. A more compact way of doing this is using a matrix that contains the quaternion rotation and scales, such that only a multiplication of the raw data with this matrix needs to be done to get the corrected data. The calibrated data is then given by the following equation [12]:

$$\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} C_1 & C_2 & C_3 \\ C_4 & C_5 & C_6 \\ C_7 & C_8 & C_9 \end{bmatrix} \begin{bmatrix} M_{rawx} - B_x \\ M_{rawy} - B_y \\ M_{rawz} - B_z \end{bmatrix} \tag{5.2}$$

In this equation the M matrix represents the offset corrected data, the C matrix is the soft iron losses correction matrix, the B matrix represents the biases induced by hard iron losses and the M_{raw} are the raw magnetometer values.

B. Implementation

The calibration is implemented as a procedure performed apart from the normal operation of the device. During this procedure the device is placed and moved in specific orientations and patterns. The raw data from these orientations or patterns is then put into Matlab where simple calculations are done.

1) Gyroscope offset measurement: To measure gyroscope offset the device is locked in one single orientation, it does not matter in what orientation since the gyroscope does not measure any external forces. 1000 samples are then taken from all three gyroscopes to prevent noise errors. The mean of these samples represents the DC offset. During normal operations this offset is always subtracted from the raw data to give the corrected data.

2) Accelerometer offset measurement: The accelerometer offset is measured similarly to that of the gyroscopes. The device is locked on a level surface with the z-axis pointing upwards. 1000 samples of the raw outputs from each accelerometer are taken. An integer value representing the gravitational pull of the earth (in this case 16384) is subtracted from all z-axis samples. Hereafter the mean of the data is calculated in Matlab. This mean is the DC offset and will be subtracted from the raw data during normal operations.

3) Magnetometer offset measurement: The magnetometers have the most complex procedure for offset calculation and correction. For the hard iron losses the device is locked in place on a level surface and 1000 samples are taken. Then the device is rotated exactly 180 degrees around the z-axis and another 1000 samples are taken. These 2000 samples are then used to calculate the offset for the x- and y-axis using equation 5.1. For the z-axis the device is put exactly at a 90 degrees angle with the level surface and 1000 samples are taken. The device is than turned 180 degrees on the surface and again 1000 samples are taken. The same method for the x- and y-axis then determines the hard iron offset for the z-axis. These hard iron offsets are always subtracted from the raw data during normal operations.

To measure the soft iron losses a third party program is used, called MagMaster. This program uses specific orientations of the device. First the device is put with upwards facing x-axis, the second measurement is with the device turned exactly 180 degrees around the x-axis. This is then repeated when the x-axis is facing downward and after that also performed with the other axes. These sampled data is then put into calculations to determine the transformation matrix. This matrix multiplication is done after the hard iron offsets are subtracted and give the corrected values during normal operations. The hard iron losses can also be calculated using this program and yielded similar results as calibration by hand

C. Results

The results of gyroscope offset calibration are given in figures 5.2 and 5.3. In the first figure the uncorrected values are given, it can be seen that these values are not centred around value zero. The corrected data in figure 5.3 shows the data around the zero value and thus without DC-offset and with this requirement [4,4] is met.

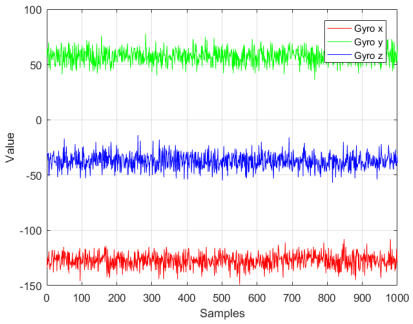


Figure 5.2: Raw gyroscope data

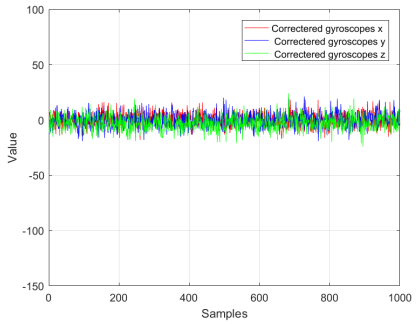


Figure 5.3: Gyroscope data corrected for offset

The accelerometer calibration results are given in figures 5.4 and 5.5. These results are similar to that of the gyroscopes.

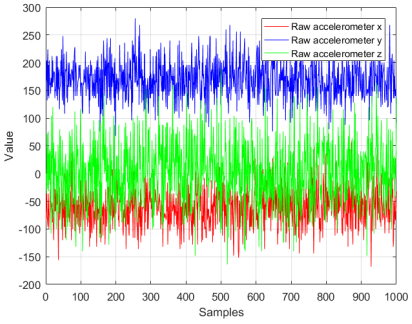


Figure 5.4: Raw accelerometer data

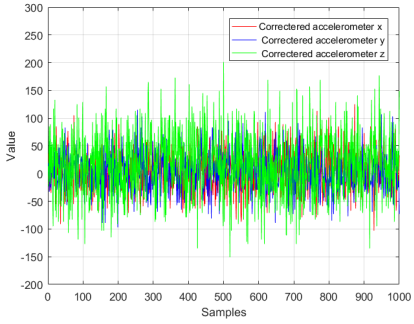


Figure 5.5: Accelerometer data corrected for offset

The magnetometers are harder to represent, since they are not calibrated around a zero axis versus time. They are calibrated to form circular data when two magnetometers axes are plotted and the device is rotated around the remaining axis. In figures 5.6, 5.7 and 5.8 the raw magnetometer data when the device is rotated is given. It can be seen that the circles do not circle around the center of the plane, these are the hard iron losses. Also it can be seen that these circles are slightly elliptic, this is due to the soft iron losses. The corrected data are given in figures 5.9, 5.10 and 5.11. The circles are almost centred around the origin and far less elliptical. Unfortunately this calibration does not meet requirement [4,4], since the offset are not completely removed.

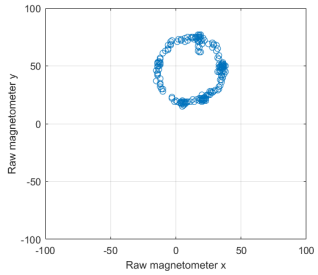


Figure 5.6: Raw magnetometer X vs Y data

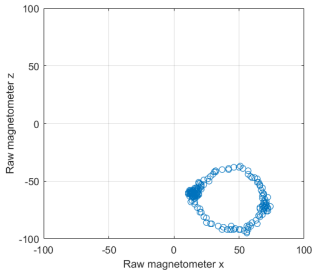


Figure 5.7: Raw magnetometer X vs Z data

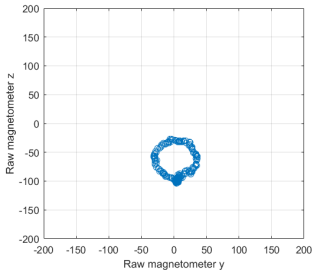


Figure 5.8: Raw magnetometer Y vs Z data

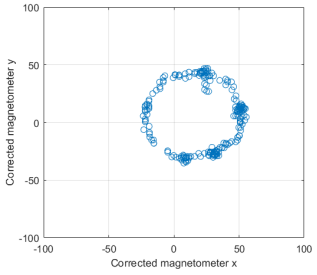


Figure 5.9: Corrected magnetometer X, Y data

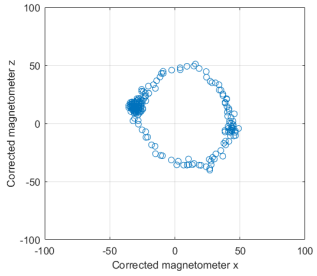


Figure 5.10: Corrected magnetometer X, Z data

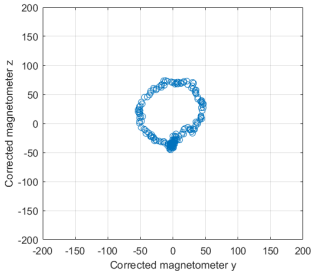


Figure 5.11: Corrected magnetometer Y, Z data

6. Noise filter

A. Theory

The measurements of especially the accelerometers and magnetometers are very noisy. To prevent the noise from having influence on the position-calculation the measurements are filtered. The reason to not use a normal low-pass filter is that a low-pass filter is slow. In some cases of significant change in value the system must response fast and therefore the low-pass filter is not used in all cases.

The noise filter make use of the following if-statements:

```
if (Value < CompareValue - NP | Value > CompareValue + NP)
{
  FilteredValue = Value;
}
else
{
  if (Value < CompareValue - NPS | Value > CompareValue + NPS)
  {
    FilteredValue = (Value + PreviousCorrect)/2;
  }
  else
  {
    FilteredValue = (1 - NPL)*PreviousCorrect + NPL*Value;
  }
}
```

The if-statement uses 6 parameters. *Value, CompareValue, PreviousCorrect, NoiseParameter* (NP), *NoiseParameterSen* (NPS) and *NoiseParameterLow* (NPL). The difference between *Value* and *CompareValue* is first compared with the value of *NoiseParameter*. If the difference between the two values is more than the *NoiseParameter* then the *FilteredValue*, the outcome of the filter, becomes *Value*. The difference between *Value* and *CompareValue* is then so significant that there is a real change and in that case the *Value* is not altered before passing it to *FilteredValue*. If the difference is less than the *NoiseParameter* the difference between *Value* and *CompareValue* is compared with *NoiseParameterSen*. If the difference is more than *NoiseParameterSen* the *FilteredValue* becomes the average of *Value* and *PreviousCorrect*. Then difference between *Value* and *CompareValue* is too big to ignore but too small to change the *FilteredValue* to *Value* and therefore the average is taken. If the difference is less than *NoiseParamaterSen* the value is low-pass filtered, with use of the parameter *NoiseParameterLow*.

B. Implementation

1) Filtering measurements accelerometers: The measurements of the accelerometers are very noisy, so to prevented the system from this noise the measurements of the accelerometer are filtered. The parameter *Value* is the output of the accelerometer. *CompareValue* is in this case the previous outcome of the filter. So the previous outcome of the filter is compared with the value of the current measurement. If the difference is more than the NP, the *FilteredValue* becomes the value of the measurement. If the difference is less than the NP the difference is compared with NPS. When the difference is less than NPS the *FilteredValue* is low-pass filtered with as reference the previous outcome of the filter, so *PreviousCorrect* is the previous outcome of the noise filter. A NPL of 0.025 is used in this case, which correspond with a cut-off frequency of $F_c = \frac{NPL}{(1-NPL)*2\pi*\Delta T} = 0.27Hz$, where ΔT is the time between the samples, in this case about 0.015 seconds, in section 11-B the timer of the system is explained. The calculated cut-off frequency is quiet low but this low-pass filter is only applied when a strong low-pass filter is needed. When the difference is more than NPS the *FilteredValue* becomes the average of *Value* and the previous outcome of the noise filter. To determine NP and NPS a measurement with a stationary accelerometer was done, the output data was observed using Matlab and the parameters were determined. In figure 5.5 in section 5-C it is clearly visible that the accelerometers are very noise and so it was determined to make NP 200 which corresponds which a acceleration of $0.119m/s^2$ and to make NPS 150 which corresponds which a acceleration of $0.0898m/s^2$. The filtering doesn't influence the velocity calculation because it filters the negative and positive acceleration.

2) Filtering measurements magnetometers: The magnetometers are like the accelerometers also a bit noisy. The magnetic field of the earth is constant enough for our accuracy, but the environment causes small changes in the measured magnetic field and this is visible in the measurement of the magnetometers. Therefore the magnetometers are also filtered using the noise filter. Like the noise filter of the accelerometers the *Value* is the value of the measurement, in this case from the magnetometers. The *CompareValue* and the *PreviousCorrect* are the previous outcome of the filter. The noise of the magnetometers were measured when staying stationary and observed using Matlab. The NP and NPS were set as (3,3,3), for the x, y and z magnetometers. NPL was set at 0.025 and thus a cut-off frequency of 0.27 Hz.

3) Filtering measurements gyroscopes: The gyroscope tend to drift, to prevent this from impacting the orientation calculation during a rotation the complementary filter is used, see 7. To prevent the gyroscope from drifting when stationary the noise filter is used. The parameter *Value* is the output from the gyroscopes. This is compared with *CompareValue* which is in this case a *NullVector*. *PreviousCorrect* and NPL are in this case also zero. Also there is no need to make use of the average function of the noise filter so NP and NPS are chosen to be the same value. Which means that when the difference between the gyroscope output and zero is more than NP and NPS, the *FilteredValue* becomes the outcome of the gyroscope. When the difference is less than NPS and NP the *FilteredValue* simply becomes zero because *PreviousCorrect* and NPL are zero. When analysing the output of the gyroscope with use of Matlab it became clear that when stationary the drift/noise gives a maximum deviation of 30. The NP and NPS becomes then (30,30,30). This corresponds which a rotation of $0.267°$/sec. Maybe this sound like much, but remember that the gyroscope is only filtered when the output of the gyroscope is between -30 and 30. When the MPU9250 is really rotating the rotation is much faster than $0.267°$/sec and the output of the gyroscope is then not filtered by the noise filter.

In table I all the parameters of the noise filter are shown.

Table I: Overview parameters used in the noise filter

	Value	CompareValue	PreviousCorrect	NP (x, y, z)	NPS (x, y, z)	NPL
Accelerometers	Value measurement	Prev. outcome filter	Prev. outcome filter	(200,200,200)	(150,150,150)	0.025
Magnetometers	Value measurement	Prev. outcome filter	Prev. outcome filter	(3,3,3)	(3,3,3)	0.025
Gyroscopes	Value measurement	'0'	'0'	(30,30,30)	(30,30,30)	0

C. Results

The noise filter was tested during movement of the MPU9250 and during a stationary state. This is done to observe if the noise filter works for all the different ways it was implemented and demanded by requirements [4,2] and [4,3].

1) Filtered accelerometer and magnetometer: The filtering of the accelerometer and magnetometer were done in the same way, but with different parameters. In figure 6.1 the figure taken with Matlab is showing the data of the accelerometer in x when the accelerometer was in stationary state. The working of the low-pass filter can be clearly seen. The filter was also tested during movement, the result of this is shown in figure 6.2. It shows the working of the low-pass filter when not changing orientation to the gravity and it shows that it follows the acceleration when it changes it orientation to the gravity. In figure 6.3 the working of the filter is shown on the data of magnetometer x during a movement. Also here the working of the low-pass filter becomes clear and the following of the magnetic field when moving.

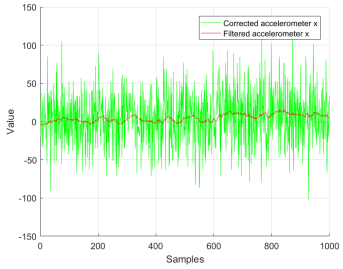


Figure 6.1: Filtering accel. x during stationary state

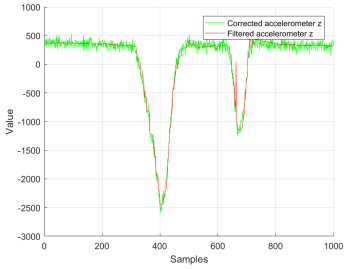


Figure 6.2: Filtering accel. x during a movement

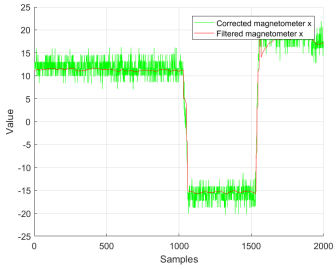


Figure 6.3: Filtering magneto. x during a movement

2) Filtered gyroscopes: The filtering of the gyroscopes was tested during stationary state and during a movement. The results of the stationary state are shown in figures 6.4, 6.5 and 6.6. As becomes clear from the figures the gyroscopes have a little offset and are noisy. The noise filter filters the gyroscopes when the measured value is between -30 and 30. The gyroscopes were also tested during a movement. A very slow rotation gives a much greater measured value than 30 and the output of the filter were, as designed, the same as the measured value.

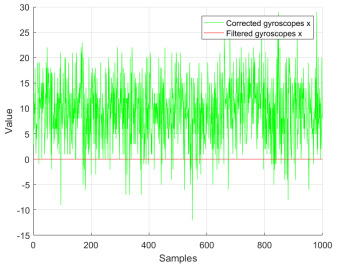


Figure 6.4: Filtering gyro. x during stationary state

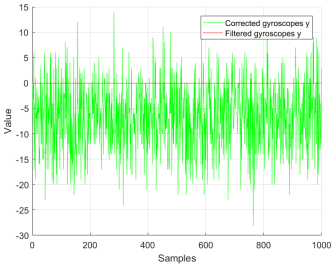


Figure 6.5: Filtering gyro. y during stationary state

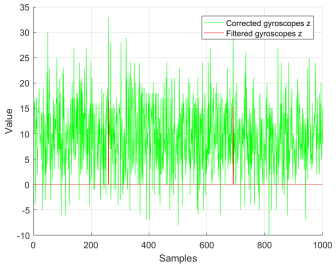


Figure 6.6: Filtering gyro. z during stationary state

7. Complementary filter

A. Theory

In order to calculate the orientation and position of the probe the angle/orientation in which the MPU9250 is standing must be known. It is possible to calculate the relative orientation of the sensor by integrating the value of the gyroscopes, but the gyroscopes tend to drift over time. The accelerometers and magnetometers can be used to determine an orientation with respect to the gravity and the earth magnetic field but they are very noisy. The complementary filter is used to compensate the gyro drift by using the accelerometers and magnetometers measurements. Where measurements of the gyroscopes are reliable on the short term and unreliable after a period of time, due to drift, the accelerometers and magnetometer are unreliable on the short term, due to noise, but they are reliable on the long term. Therefore combining the reliability of the gyroscopes on short term with the reliability of the accelerometers and magnetometer on long term with use of a complementary filter gives an accurate orientation estimation.

Another filter that can be used is the Kalman filter [13]. The Kalman filter calculates the orientation with the knowledge of the noise of the system and measurements. Unfortunately it hard to determine these noise parameters and the Kalman filters requires a lot of computations, making it slow. It is therefore hard to implement this filter on a microcontroller. On the other hand the complementary filter is simple to implement and requires no knowledge of the noise. The complementary filter gives the same or even better results than the Kalman filter [14]. Also the Madgwick filter [15] could be used to estimate the orientation. The Madgwick filter produces an orientation quaternion by using the measurement data from all sensors and two filter parameters. Calculations done by the Madgwick are more intense than the complimentary filter, but are less intense than the calculation done by the Kalman filter. The Madgwick filter achieves more accuracy than the Kalman filter and complementary filter. Nevertheless the orientation determination is done by using the complementary filter. The Madgwick and Kalman filter requires to many calculations, this can‘t be done on an ATmega328 when there are also other calculation that must be done on the ATmega328, as stated in requirement [5,1].

The orientation will be determined in the angles yaw, pitch and roll. Yaw (ψ) gives the rotation around the z-axis in world frame. Pitch (θ) gives the elevation with the horizontal plane and will in our case be the rotation around the y-axis of the MPU9250. Roll (ϕ) gives the rotation around the x-axis of the MPU9250, see figure 7.1[3]. With those angles a quaternion will be constructed, which will be used for calculation the orientation of the probe, see section 8.

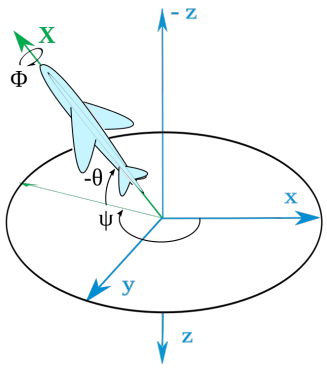


Figure 7.1: yaw ψ, pitch θ and roll ϕ

First pitch and roll are calculated with the use of the complementary filter, which combines the measurement of the accelerometers and the gyroscopes. The complementary filter applies a low-pass filter on the measurements of the accelerometers and applies a high-pass filter on the measurements

[3]Figure from https://en.wikipedia.org/wiki/Euler_angles

of the gyroscopes. It combines the angular velocity measured by the gyroscopes with the orientation estimated by the accelerometers. The measurement of the accelerometers are already low-pass filtered by the noise filter when change in acceleration is not too high, but when making a real movement the measurements are not filtered by the noise filter but by the complementary filter. The same is true for the gyroscopes, when not rotating the drift is filtered by the noise filter, when rotating the drift is filtered by the complementary filter.

The gyroscopes measures the angular velocity in degree per sec (dps). By integrating the angular velocity the angle of the MPU9250 can be determined. This angular velocity is given in 16 bits and with a full rate scale of -/+ 250 degrees. Therefore the LSB/deg/sec is $\frac{2^{16}}{500} = 131$, in which LSB means the *Least Significant Bit*. The measured value of the gyroscopes must therefore by divided by this LSB value in order to get the angular velocity in degrees per second. The angular velocity is then given by equation 7.1.

$$angularvelocity = value\_gyro/131 \tag{7.1}$$

The accelerometers measure acceleration and therefore also the gravity of the earth. By determining the ratio of the different amount of gravity measured by the individual accelerometers the orientation with respect to the gravity can be calculated. This is done by equations 7.2 and 7.3 [16]. The variables *ax, ay* and *az* represent the acceleration measured by the individual accelerometers for the x-, y- and z-axis of the chip respectively. Equation 7.2 is computed with the *arctan2* computer math function, which makes the output of the *arctan* go from (-π, π] where the normal function *arctan* goes from (-$\frac{\pi}{2}$, $\frac{\pi}{2}$). This results in a roll angle that can go from -180 degrees to 180 degrees.

$$aRoll = \arctan(\frac{ay}{\sqrt{ax^2 + az^2}}) \tag{7.2}$$

$$aPitch = \arctan(\frac{-ax}{\sqrt{ay^2 + az^2}}) \tag{7.3}$$

The angles roll and pitch can now be calculated with the complementary filter equations 7.4 and 7.5. In these equations the dT is the time between two iterations and *roll/pitch* on the right side of the equation is the result of the roll/pitch calculation of the previous iteration. The *aRoll* and *aPitch* estimated with the accelerometers needs to be converted to degree since they are calculated in radians. α is the filter parameter and must be between 0 and 1. The lower the α the more the roll/pitch listens to the accelerometers. Because the gyroscopes are more reliable on the short term and the accelerometer and magnetometer more on the long term a high α is preferred.

$$roll = \alpha * (angularvelocity[x] * dT + roll) + (1 - \alpha) * aRoll * \frac{180}{\pi} \tag{7.4}$$

$$pitch = \alpha * (angularvelocity[y] * dT + pitch) + (1 - \alpha) * aPitch * \frac{180}{\pi} \tag{7.5}$$

The yaw rotation cannot be calculated with only the use of the accelerometer. This is because the z-axis is defined in the direction of the gravity. A rotation around this axis does not change the readout of the accelerometer because the angle with respect to the gravitational field of the earth stays the same. To calculate a yaw rotation the magnetometers are used. The magnetometers measure the earth magnetic field, which is perpendicular to the gravity, and can therefore determine rotation in the horizontal plane. Because the magnetometers are align with the axis of the sensor the measurements of the magnetometer needs to be corrected for any pitch or roll to get the yaw rotation. The yaw rotation is calculated by equations 7.6, 7.7 and 7.8, where *mx, my* and *mz* are the normalized value of the magnetometers and *roll* and *pitch* are in radian [17]. The yaw equation 7.8 is computed by the same *arctan2* computer math function as *aRoll*, to let the *yaw* go from -180 degrees to 180 degrees.

$$magx = mz * sin(roll) - my * cos(roll) \tag{7.6}$$

$$magy = mx * cos(pitch) + my * sin(pitch) * sin(roll) + mz * sin(pitch) * cos(roll) \tag{7.7}$$

$$yaw = \arctan(magx/magy) * rad2deg \tag{7.8}$$

So with the use of the gyroscopes and the accelerometers the complementary filter calculates the pitch and roll angles. Due to the fact that the yaw angle can‘t be determined by the accelerometer the yaw angle is calculated with the use of the measurements of the magnetometers.

B. Implementation

The complementary filter was implemented as a function in the code. The function uses the given equations in section 7-A and has as input the calibrated and filtered values of the accelerometers, gyroscopes and magnetometers. The gyroscope inputs were also corrected with the LSBvalue and the magnetometer and accelerometer inputs were normalized. The parameter α was set at 0.9. This was determined by looking to the results of different values of α.

C. Testing

The complementary filter was first tested during a horizontal stationary state, see figure 7.2. Roll and pitch are around zero which is as expected. They are not completely zero, because the chip was not perfectly aligned with the gravity. The yaw is hovering around some value that indicates the earth magnetics field direction, this is as expected because the set up was not aligned with a compass. Then a roll movement was made, first a movement from horizontal state to a roll of about 90 degrees, around samples 120 to 180 in figure 7.3, then back to horizontal (samples 150 to 200) and then a movement to a roll of about -90 degrees and back (samples 350 to 500), see figure 7.3. The accelerometers and gyroscopes are working perfectly together to get the right angles without drift or noise. Then a movement in pitch was tested. First a movement from horizontal to about -90 degrees pitch and then back to horizontal (samples 40 to 220) and then a movement to about 90 degrees pitch and back (samples 350 to 500), see figure 7.4.

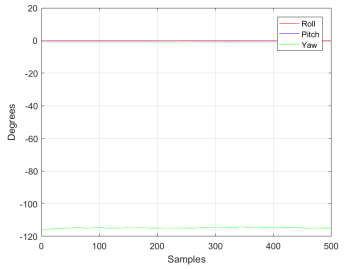


Figure 7.2: Horizontal stationary state

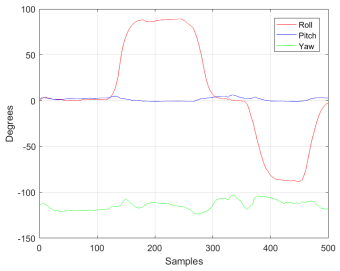


Figure 7.3: A roll movement

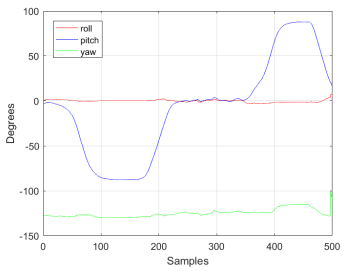


Figure 7.4: A pitch movement

To measure the accuracy of the roll and pitch the MPU9250 was rotated with an angle of 90 degrees in roll, figure 7.5, and pitch, figure 7.6. The mean of the samples 200 to 300 of the roll movement was 88.7 degrees, this is between the 2 degrees accuracy needed by requirement [3,1,4]. All the samples between 200 and 300 had the right accuracy. The mean of the samples 200 to 300 of the pitch movement was 88.4 degrees, also this meets requirement [3,1,4]. Some of the values of the samples between 200 and 300 are given in appendix B in table III.

A problem occurred when testing the yaw. When testing the yaw under the same condition as where it was calibrated the yaw seemed to work fine. But after some testing it was discovered that a small change in position between the MPU9250 and the wires of the device changes the magnetic field significantly and also electronic devices influenced the measurement too much to do a good estimation of yaw rotation. It was concluded that in the world of today, which lot of electronic devices, it is hard to measure only changes in the earth magnetic field since the earth magnetic field is not very strong. From now on the yaw rotation is set on 0 degrees and the MPU9250 is only rotated in roll and pitch.

To make sure that the problem really is the measurements of the magnetometer, the equation for calculating the yaw was tested with the use of Matlab. Different value of normalized magnetometer values by different angle of pitch and roll were tested. The outcome of the testing were as expected. In

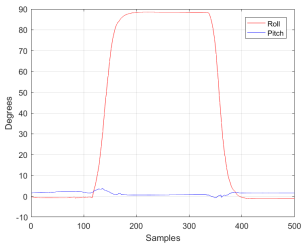


Figure 7.5: A roll movement of 90 degrees

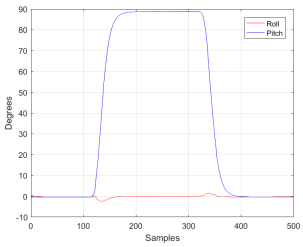


Figure 7.6: A pitch movement of 90 degrees

table IV in appendix B the results are shown. The Matlab-code *testyaw.m* can be found on GitHub as indicated in appendix D.

D. Discussion

In section 7-C it is shown that the determination of the roll and pitch angles works fine and that the determination of the yaw angle works fine in theory, but that there is a problem with the determination of the yaw with the real measurements. It occurred that nowadays an one time calibration is not enough to corrected for the changing magnetic field produced by electronic devices. To corrected for the changing magnetic fields it is an option to calibrate before every measuring session, this can be done by using an algorithm [18]. The algorithm asks the user to move/rotate the magnetometer in all direction and determines the calibration values from the measurements done during those movements. This algorithm works fine when the surrounding area doesn't have a varying magnetic field, but when moving an electronic device a little bit closer the calibration becomes incorrect. To eliminate the influence of the magnetic field of the wires of the MPU9250, the wires can also be shielded. This eliminates the influence of the magnetic field of the wires, but does't eliminated the influences of other surrounded magnetic fields.

Another option to determine the yaw rotation is by reading out the quaternion of the dmp of the MPU9250. A problem with the dmp is that it only uses the measurements of the accelerometer and gyroscope. InvenSense doesn't give any information on how the dmp calculates the yaw rotation and after reading out the quaternion and calculating the yaw from this quaternion, it became clear that the yaw calculated with the dmp didn't give a very precise rotation. Therefore the decision was made to let the user only rotate in roll and pitch and to set the yaw rotation to zero for now.

For future developments of the orientation and position system it would be preferable to implement the Madgwick filter. The Madgwick filter needs more calculations to estimated the quaternion of the rotation but is more accurate than the complementary filter. When implementing the Madgwick filter another microcontroller is needed to do the calculation, the ATmega328 cannot do the Madgwick filter calculations and also calculate the position and orientation of the probe.

8. Position and orientation calculation

A. Theory on orientation

The design given in section 3 indicates that the position of the probe tip and the MPU9250 are different. In this section a mathematical model for this system of two points is given which will describe the location and orientation of the probe tip as function of the MPU9250 position and orientation change. For this the acceleration measured by the accelerometers is converted to displacement and the orientation of the MPU9250, calculated by the complementary filter (section 7), is transformed to a quaternion (a sort of vector). With this displacement and quaternion the position and orientation of the probe is then calculated. This position and orientation can then by fed to the visualisation-software. In this section the main mathematical theory about the method that is used to trace the orientation of the probe. Subsection 8-B goes into details on the theory behind displacement tracking of the probe by the use of accelerometers.

1) Reference frame: First the reference frame and starting position of the device. The MPU9250 will not be placed precisely on the tip of the probe, the point that must be tracked. So an offset vector from the MPU9250 to the probe-tip is needed. An overview of the system is given in figure 8.1. In this reference frame the x,y,z axes are defined by the right hand rule and three rotations parameters defined as the rotation over either the x, y or z axis are also defined by the right hand rule around those axes. The chip itself calculates its orientation using pitch, roll and yaw as described in section 7-A.

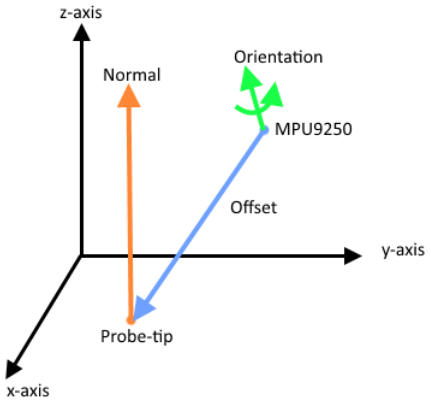


Figure 8.1: Reference frame of probe tip and positioning sensor model

In figure 8.1 Probe-tip and MPU9250 are respectively the position vector of the probe-tip and the MPU9250. Offset is a vector from the MPU9250 to the Probe-tip, the normal is the normal vector of the probe and last is the orientation quaternion given by the chip, calculated from the pitch, roll and yaw. During operations the position and orientation of the MPU9250 are calculated and from this given position, the known offset vector and the orientation of the normal to the MPU9250, the position of the tip and the normal are calculated. At the start of operations the position of the MPU9250 is taken as the zero point and the position of the tip is calculated, with as reference the position and orientation of the MPU9250. In section 9 the algorithm to determine the offset vector and the orientation of the normal with respect to the MPU9250 are explained.

2) Quaternions: To determine the orientation of the normal quaternions are used. Quaternions are a number system that extends the complex number. A quaternion is made off 4 numbers, one real and three complex ones called [i,j,k], these parameters are defined as follows:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{8.1}$$

It is important to note that quaternion multiplication is non commutative and most other operations are not as simple as they seem. Luckily, for this project quaternions are only needed for their unique

rotational properties as vectors and will thus described as vectors from this point on. However, the math behind the operations remains completely different from normal vector math.

Quaternions [19] are defined as three dimensional vectors that also track their spin, making it a 4 parameter object. These parameters are: [w,x,y,z], wherein the w is the rotation around its own axis in respect to the reference frame in radians and x,y,z are the familiar coordinates to indicate a vector in three dimensions. The rotation (w) is important to track the orientation of objects. In most mechanical models rotation of a vector is tracked by the amount of rotation around the reference frame axes, so an x rotation being a rotation of the vector around the x-axis. A full three dimensional rotation can then be given by three rotations around axes, for example a rotation around the x-axis, z-axis and x-axis again. In this project the order is unknown as the rotation is measured only as start orientation and final orientation, whether the first rotation was around the x-axis or not is not known, but does matter. For example a rotation around the x-axis does not affect the vector if the vector is located along the x-axis and thus a rotation around the z-axis afterwards yields a different result if the order is inverted. This problem is known as Gimbal-Lock [20]. The quaternion does not have this problem. This makes the quaternion very suited for calculations with our device. Normally the euler angles will suffice if the device is not rotated more than 180 degrees over one axis, however, the device will track along three dimensional objects that could have all sorts of shapes. It is therefore needed that the device can track its orientation under all conditions and shapes, as demanded in requirement [1,2]. This is why quaternion representation of the orientation is chosen over euler angles.

Next are some basic quaternion operations that are needed to calculate orientation. First to construct a quaternion from angles the following expressions can be used:

$$w = cos(pitch/2) * cos(yaw/2) * cos(roll/2) + sin(pitch/2) * sin(yaw/2) * sin(roll/2) \tag{8.2a}$$
$$x = sin(pitch/2) * cos(yaw/2) * cos(roll/2) - cos(pitch/2) * sin(yaw/2) * sin(roll/2) \tag{8.2b}$$
$$y = cos(pitch/2) * sin(yaw/2) * cos(roll/2) + sin(pitch/2) * cos(yaw/2) * sin(roll/2) \tag{8.2c}$$
$$z = cos(pitch/2) * cos(yaw/2) * sin(roll/2) - sin(pitch/2) * sin(yaw/2) * cos(roll/2) \tag{8.2d}$$

In this equation the roll, pitch and yaw are the angles calculated in section 7. The w, x, y and z are the quaternion parameters.

In quaternion maths a vector is rotated around a quaternion and not around the reference frame. The x,y,z of the quaternion indicate the axis around which the vector will be rotated and the w is the amount of rotation around this axis. A rotation around a reference frame axis can now be done without ever gimbal locking, since the rotation quaternion will never be on the same axis as the vector in question. Furthermore there exist no order of rotations any more because only one rotation is done. Rotation and orientation quaternions are unit vectors and for this project only those two types are used. To rotate a vector v around a quaternion q the following calculation is used:

$$v_{out} = q * v_{in} * conj(q) \tag{8.3a}$$
$$conj(q) = [w, -x, -y, -z] \tag{8.3b}$$

With v_{in} the input vector, v_{out} the output vector and $conj(q)$ the conjugate of quaternion q. The multiplication here is a quaternion multiplication, only done with quaternions, so the vector must be temporary stored in a quaternion. To make the vector a (temporary) quaternion the x,y,z are copied and the w is left 0. Afterwards the x,y,z of the quaternion are restored in the new vector and the w is abandoned, since the vector does not need to remember its rotation around its own axis. Quaternion multiplication of quaternion p(w,x,y,z) and quaternion q(w,x,y,z) is as follows per parameter:

$$w = p.w * q.w - p.x * q.x - p.y * q.y - p.z * q.z \tag{8.4a}$$
$$x = p.w * q.x + p.x * q.w + p.y * q.z - p.z * q.y \tag{8.4b}$$
$$y = p.w * q.y - p.x * q.z + p.y * q.w + p.z * q.x \tag{8.4c}$$
$$z = p.w * q.z + p.x * q.y - p.y * q.x + p.z * q.w \tag{8.4d}$$

In this equation the dots indicate the quaternion (p or q) followed by the parameter (w, x, y or z). To rotate a quaternion around another quaternion they are multiplied (this is non commutative). Thus rotating quaternion p around quaternion q is:

$$p_{out} = q * p_{in} \tag{8.5}$$

In which p_{in} is the input quaternion and p_{out} is rotated quaternion. Note that the order of multiplication matters. To calculate the rotation quaternion q that rotates quaternion p_{in} to quaternion p_{out} the inverse multiplication must be used:

$$q = p_{out} * inv(p_{in}) \tag{8.6a}$$

$$inv(p_in) = \frac{conj(p_{in})}{abs(p_{in})} \tag{8.6b}$$

With $abs(p_{in})$ the absolute of p_{in}, which is calculated the same way as for normal vectors. Since this project only uses unit quaternions $abs(p_{in})$ will always be one and can be removed from the equation. These equations give the basics of quaternion maths that will be needed to track the orientation of the device.

3) Calculations on model: With the knowledge of quaternions the mathematical model for the device can be made. If the orientation of the MPU9250 changes (a rotation) all other object will rotate along with the orientation quaternion. This is because the system is rigid and all objects are connected without shifting parts. This means that given the previous and current orientation of the MPU9250 the rotation quaternion that rotated the previous to current orientation can be calculated. The calculations described in section 7 will give this previous and current orientation. Calculating this rotation vector is then done using equation 8.6. Because the entire device turns in the same way, this rotation quaternion can be used to rotate all other vectors and orientations of the device, such that they match the new orientation of the entire device. Using equation 8.3 the new orientation of the normal and offset vectors are calculated. Now that all necessary rotations are done the system can be translated.

B. Theory on converting acceleration to displacement

To determine the translation of the MPU9250 the change in place (displacement) must be know. This displacement can be found using the accelerometers and the orientation found by the complementary filter. The acceleration of the MPU9250 must still be converted into actual displacement in meters. This subsection describes the maths behind this derivation. The MPU9250 provides the acceleration of the chip in the x,y,z direction of the sensor frame. Knowing the orientation of the MPU9250 and the gravitational constant the acceleration due to the gravity of the earth can be cancelled out. The orientation of the device indicate how much each axis is pointing to the earth with respect to each other. Multiplying the gravitational constant with these respects gives the offset for each accelerometer axis. Subtracting this amount from the raw data then gives gravity free acceleration. The remaining acceleration needs to be integrated down to distance. The acceleration is integrated down to a velocity vector that tracks the velocity during the entire operation. This velocity vector is then integrated one more time to give the position shift during one iteration. This is done using the following equations:

$$\Delta v = dT \cdot a \tag{8.7a}$$

$$\Delta s = dT \cdot \Delta v \tag{8.7b}$$

Here v is the velocity vector, a the acceleration vector, s the distance shift vector and dT is the time interval. This theory works for continuous data that is used after measurement. However the device requires real time displacement updates and uses discrete input values. Therefore integration is done using trapezoidal integration [21]. Trapezoidal integration is a discrete method that uses the current and previous measurement to determine the integrand as follows:

$$y(n) = y(n-1) + \frac{1}{2} \cdot dT \cdot [x(n-1) + x(n)], n = 1, 2, 3... \tag{8.8}$$

In this equation $y(n)$ is the integrated output, $y(n-1)$ is the previous output, dT is the time interval of measurements, $x(n-1)$ is the previous input and $x(n)$ is the current input. This method is more accurate than the standard rectangular integration and only requires one previous value to be saved. A visualization of the difference between rectangular integration and trapezoidal integration is given in figure 8.2. This integration method is then applied first to the acceleration and second to the velocity to give the displacement. Using the trapezoidal integration method will work for noise-less data, which

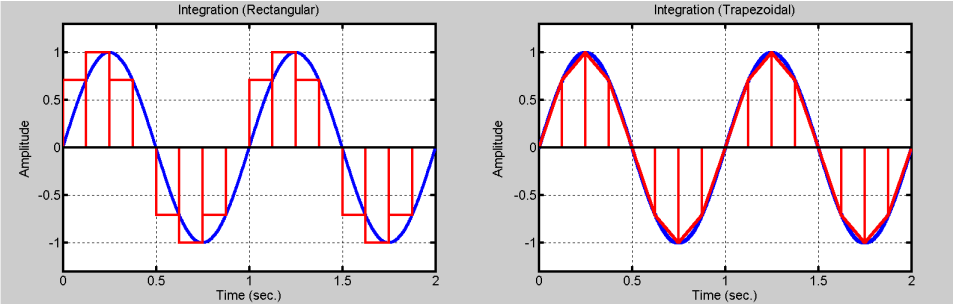


Figure 8.2: Integration using rectangular and trapezoidal methods [21]

does not come from MEMS accelerometers. The greatest thread are DC components, because integrating a constant gives a slope and the second integration will give a exponential function, quickly making the output data unusable. To prevent this a high-pass filter must be applied to the input data, to correct this even better a high-pass filter should be implemented between every integration step as well. Two of the most common digital filtering methods are the Infinite Impulse Response filtering (IIR) and Fast Fourier Transform filtering (FFT), unfortunately these cannot be applied real time and were therefore not used. The Finite Impulse Response filtering (FIR) method can be used for its real time application and the delay of the filter can be easily altered. The filtering method is described by the following non-recursive difference equation:

$$y(n) = a_0x(n) + a_1x(n-1) + a_2x(n-2) + ... + a_{N-1}x(n-(N-1)) + a_Nx(n-N) \tag{8.9}$$

In this equation $y(n)$ is the filtered output, a_n are the filter parameters that give weight to each element, $x(n)$ are the inputs and N is the length of the filter. Making N large will make the filter use more samples and thus become slower to change, a stronger high-pass filter. This will also make the response slower. In this implementation the device must react real time to changes, but the sampling rate is 66Hz, higher than the movement changes the doctor will make with the device. Thus the filter can be applied, simply with a relatively low N. This filter would induce too many calculations and delay on the results and, after testing, was chosen not to be used. The equation is left here to indicate the possibility to use this in feature research where a bigger microcontroller can be used.

The final implementation took a different route, in stead of filtering with a high-pass filter that responses strongly to change a slow low-pass filter is applied to determine the drift and then subtract the found drift from the input data. In this way the overshoots by the high-pass filter do not occur, but drift is still countered. The equation for a simple low-pass filter is as follows:

$$u(n) = (1-\alpha)\cdot u(n-1) + \alpha \cdot x(n-1) \tag{8.10a}$$

$$y(n) = x(n) - u(n) \tag{8.10b}$$

In equation 8.10a $u(n)$ is the filter coefficient that indicates the drift of the input signal, α is the filter constant, $x(n)$ is the raw input and $y(n)$ is the output. What differs this technique from the high-pass filter is the fact that not the previous outputs, but the previous filter coefficient is looped. This means that the drift solely depends on the input and thus there occur no overshoots due to high frequent changes in the output. The filter constant is determined as follows:

$$\alpha = \frac{2\cdot\pi\cdot f_{cut}\cdot dT}{2\cdot\pi\cdot f_{cut}\cdot dT + 1} \tag{8.11}$$

Where f_{cut} is the cut-off frequency and dT is the time constant, thus $\frac{1}{f} = 0.015s$ (for the sampling rate of 66Hz). For this method of filtering in order to get displacement from acceleration a cut-off frequency of 0.7Hz is taken [21]. This gives a constant of 0.0619. Very close to zero (the minimum) as expected, because only drift must be detected by this filter.

C. Uncertainties

There exist a number of uncertainties that come from this mathematical model. First of the acceleration is tracked at intervals, so not all changes in acceleration are recorded and thus not all changes in position are calculated. Due to the fact that the system should not make too sudden motions compared to the sampling rate of 66Hz this error should be minor. The next problem is that the measurements of the accelerometers are corrected for the gravity they measure, this is done with the use of the calculated orientation in section 7. A wrong calculated orientation will give a wrong correction of the measured gravity, which will result in an acceleration measured, which is actually gravity. The last problem is the order of rotating and translating. The order in which this is done matters, first translating and then rotating yields different positions. In this implementation first rotating and then translating is chosen. This is because the orientation must be known before subtracting the gravity factor from the accelerometer data, so the displacement only makes sense after calculating the orientation. Also due to the high sampling rate of 66Hz and the supposed calm motion of the doctors hand this error will be very minor.

D. Implementation

The model described above is implemented on the microcontroller and written in standard ANSI C. The base code can be found, including helper headers to implement the quaternion math, on GitHub as indicated in appendix D (*TIPSMPU9250.ino*). The readout values of the MPU9250 (accelerometers, gyroscopes and magnetometers) will be processed to orientation and acceleration using the filters described in sections 6 and 7. Thereafter the quaternion is determined using equation 8.2 and the translation is calculated using equation 8.8 after filtering using equation 8.9. With the high-pass filter applied to acceleration before integrating, velocity before the second integration and finally also to the displacement after the second integration. These give the final orientation and position of the probe. An overview of the program is given in section 11.

E. Testing

The mathematical theories proofs [22] have long been found and not made in this paper. In section 7 it was found that the orientation of the yaw was incorrect and thus testing displacement using this orientation would give wrong results. What can and will be tested here is the tracking of position by the accelerometers. In order to test this the orientation tracking is turned off, so that the wrong orientation will not be reflected in the readout accelerations. This does mean that the gravity needs to be subtracted from the raw data by hand. To do this the device is put level with the ground, so that gravity is only seen in the z-direction and can then be easily subtracted. The device is then moved over one of the three axes for 10cm. To see the effects of the high-pass filter both the unfiltered as the filtered data is extracted during testing.

To see how much the outcomes vary over multiple measurements the displacement can be summed to give a distance travelled during the measurement. This distance is then compared to the actual distance travelled and the differences give the accuracy. For this the device is moved 10 centimetres 15 times.

To test the mathematical model a simulation is done using Matlab and the formulas described in the theory subsection. This will solely test if the rotation and displacement merge together to give a position in the world frame. To do this there will be two manual inputs: displacement measured in the x,y,z directions of the chip and rotations with respect to the chips axes. The outputs will describe the location of the probe tip and probe sensor assuming they are connected by a vector that is (0,1,0) from sensor to tip for reading simplicity. This means the sensor starts in the origin and the probe-tip will be at (0,1,0). Now four operations are done on the model, two translations and two rotations. These operations are further explained in the results. The five resulting positions the device will undertake are visualised in a 3D figure.

F. Results

First the results for moving in the x direction measured with the x accelerometer are given. The acceleration is given in figure 8.3, the velocity in figure 8.4 and the displacement in figure 8.5. Next the same is done for the y measurements in figures 8.6, 8.7 and 8.8 and for the z measurements in figures 8.9, 8.10 and 8.11. The red lines indicate the unfiltered values and the blue lines indicate the filtered values. The unfiltered value of the velocity and the displacement are the integrated value of the filtered value of respectively acceleration and velocity. The movement was started around sample 120 for each axis and stopped around sample 150.

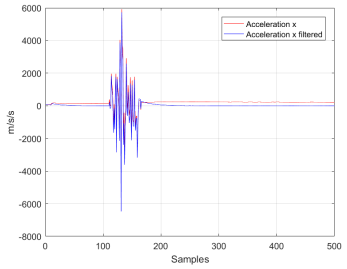


Figure 8.3: Acceleration x

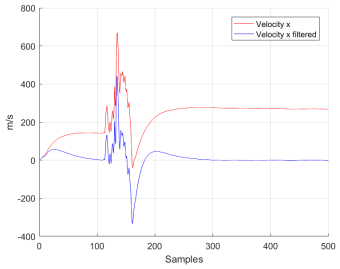


Figure 8.4: Velocity x

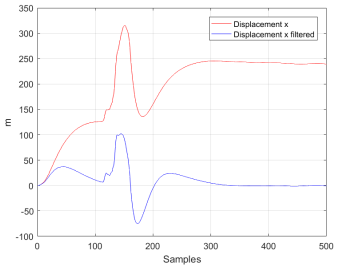


Figure 8.5: Displacement x

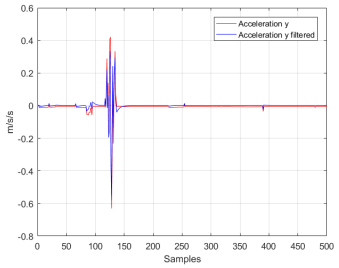


Figure 8.6: Acceleration y

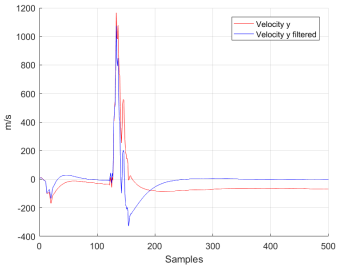


Figure 8.7: Velocity y

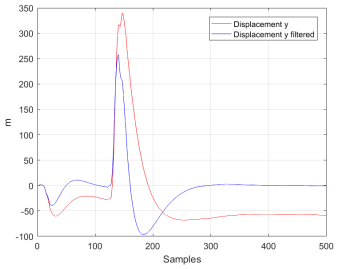


Figure 8.8: Displacement y

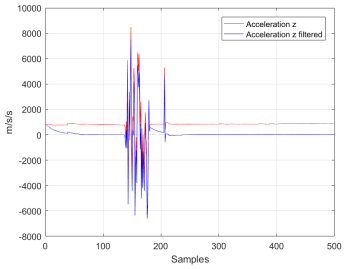


Figure 8.9: Acceleration z

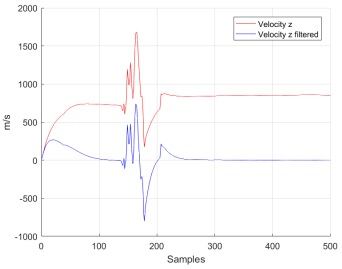


Figure 8.10: Velocity z

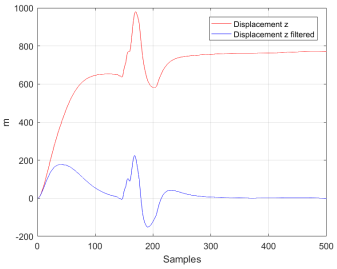


Figure 8.11: Displacement z

What can be clearly seen from these figures is that the unfiltered values drift heavily. The filter corrects this well as can be seen by the fact that the filtered values all originate and come back to zero after a movement was made. However, there is one problem that occurs with the high-pass filtering. The filtering induces a delay in the velocity and displacement that induces uncertainties. This is because the change in acceleration is very sharp and this sharpness is lost due to integrating. These results mean that the filter does not suffice requirement [4,2], but does suffice requirement [4,3].

The accuracy test gave a mean value travelled of 9.86 centimetres, so only 1,4 mm off target. The read out summed displacements are given in table II. The first row of the table indicates the measurement

number, the second row the acquired sum and the third row the error of the measurement with respect to the 10cm that was actually moved. As can be seen the error is greater than the precision said in requirement [3,1,2].

Table II: Displacement test results

Measurement Required	Sum (cm) 10	Error (cm) +- 0.5	Measurement	Sum (cm) 10	Error (cm) +- 0.5
1	8.2600	1.7400	9	15.5500	-5.5500
2	7.2400	2.7600	10	12.6500	-2.6500
3	19.2800	-9.2800	11	10.0600	-0.0600
4	8.8200	1.1800	12	2.6900	7.3100
5	6.3900	3.6100	13	15.7200	-5.7200
6	10.1000	-0.1000	14	8.0400	1.9600
7	4.1800	5.8200	15	9.9500	0.0500
8	8.9800	1.0200			

The testing of the theoretical orientation and displacement model results are given in figure 8.12. Five vectors are given of which each has a separate colour and a X to indicate the sensor and a O to indicate the tip. The staring position (red) has the device lying on the xy-plane with the sensor in the origin and the tip at (0,1,0). At this point the sensor itself is aligned with the reference frame. The first operation is a 90 degrees roll (around the x-axis of the sensor) of the device, this will place the probe upright without moving the sensor point. The result of this is the second orientation (blue), where the sensor is still at the origin and the tip is in (0,0,1). Next is a displacement of 1 in the positive x direction of the sensor. Because the x-axis of the chip and the world frame align at this moment the result is easily recognised, the sensor ends in (1,0,0) and the tip in (1,0,1). This is the third position (green). Now a pitch change of negative 45 degrees is performed, placing the tip at $(\frac{1}{\sqrt{2}},0,1-\frac{1}{\sqrt{2}})$ and leaving the sensor at (1,0,0). This is the fourth position (magenta). The last operation is a displacement of 1 in the negative y direction. This is not aligned with the world frame, so this tests if the model correctly combines displacement with orientation. The result is that the sensor is moved to $(1+\frac{1}{\sqrt{2}},0,-\frac{1}{\sqrt{2}})$ and the tip moves to (1,0,0). This is the expected end position (black). The mathematical models works correctly and combines displacement and orientation well.

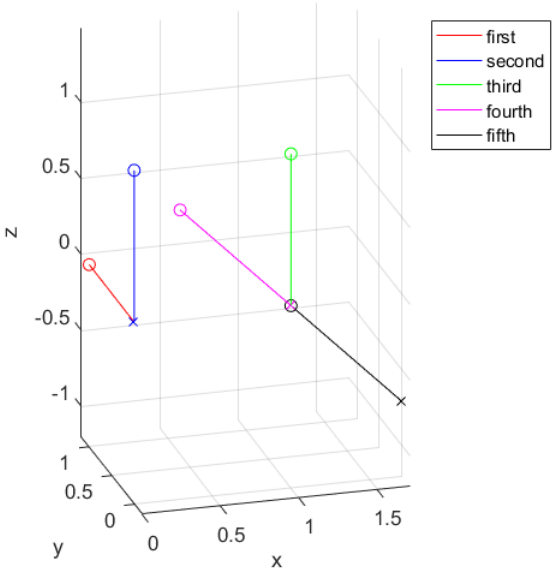


Figure 8.12: Results of four operations on the mathematical model

G. Discussion

The results suggest the use of a better high-pass filter. This filter will have to cancel drift the same way the current filter does, those results are good. But the main improvement needs to be in the filtering on movements, currently this is done incorrectly and overshoots at reverse movements after a normal movement cancel out the total movement, making the result unusable. A filter with more filter coefficients as described in subsection 8-B would be a good first guess. Implementing this filter will cause more delay in the results and will require more programmable space than is available on the ATmeaga328.

9. Determination of distance and orientation between MPU9250 and probe-tip

A. Theory

In section 8 it was explained how to calculate the position and orientation of the MPU9250 and also the position of the probe-tip and the orientation of the probe. To calculate this position and orientation, the distance between the MPU9250 and the probe-tip and the orientation between those two must be known. In this section the estimation of the distance and the orientation from MPU9250 to the probe-tip will be explained.

1) Distance estimation: The distance between the MPU9250 and probe-tip cannot simply be measured by hand, because the MPU9250 is in the handle and it is not known where exactly the MPU9250 measures. Also an estimation done with the output of the MPU9250 will be more accurate, since the final measurements also will be done by the MPU9250. To estimate the distance between the MPU9250 and the probe-tip, the probe-tip will be fixed on one point. The probe will then be moved around, but without moving the probe-tip. The positions of the MPU9250 will be on a sphere around the probe-tip, since the distance between the MPU9250 and the probe-tip stays the same during the movement. The calculated position of the MPU9250 will be read by Matlab and used to calculated the centre point and radius of this sphere.

The centre and radius of a sphere can be calculated when four points lying on the sphere are given [23]. Those four point must be different and must not lie in the same plane. The centre of the sphere (x,y,z) can then be determined by calculating the determinant given in equation 9.1, where x_1 is the x value of point 1 etcetera. This determinant must be zero.

$$\begin{vmatrix} x^2+y^2+z^2 & x & y & z & 1 \\ x_1^2+y_1^2+z_1^2 & x_1 & y_1 & z_1 & 1 \\ x_2^2+y_2^2+z_2^2 & x_2 & y_2 & z_2 & 1 \\ x_3^2+y_3^2+z_3^2 & x_3 & y_3 & z_3 & 1 \\ x_4^2+y_4^2+z_4^2 & x_4 & y_4 & z_4 & 1 \end{vmatrix} = 0 \tag{9.1}$$

Stephen R. Schmitt [23] gives a method to rewrite this determinant into the different cofactors M_{11}, M_{12}, M_{13}, M_{14} and M_{15}, which then gives equation 9.2. By using the general equation of a sphere this equation can be rewriting to equations 9.3, 9.4, 9.5 and 9.6. Those equations give the centre of the sphere and the radius (r) of the sphere. This radius is the distance between the MPU9250 and the probe-tip.

$$(x^2+y^2+z^2)M_{11} - x*M_{12} + y*M_{13} - z*M_{14} + M_{15} = 0 \tag{9.2}$$

$$x = 0.5 * \frac{M_{12}}{M_{11}} \tag{9.3}$$

$$y = -0.5 * \frac{M_{13}}{M_{11}} \tag{9.4}$$

$$z = 0.5 * \frac{M_{14}}{M_{11}} \tag{9.5}$$

$$r^2 = x^2+y^2+z^2 - \frac{M_{15}}{M_{11}} \tag{9.6}$$

2) Orientation estimation: To estimate the orientation between the MPU9250 and the probe-tip the probe is set in a vertical position. The probe is then rotated around the z-axis, thus staying vertical. The MPU9250 then makes a circle around the probe. Matlab is used to read out the position and orientation of the MPU9250 on this circle. With three points the radius and the centre of the circle can be determined. This can be done by making a line between the first two points (A and B in figure 9.1[4]) and between between point 2 (B) and 3 (C) . The crosspoint of the perpendicular bisectors of those two lines is the centre of the circle. Calculating the distance between the one of the point and the centre will give the

[4]Figure from http://xahlee.info/SpecialPlaneCurves_dir/Circle_dir/circle.html

radius of the circle. This works in 2D, in 3D you must first define the plane where the points are on and then calculate the circle on this plane.

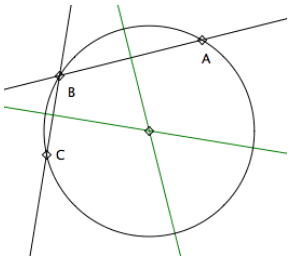


Figure 9.1: Calculation of a midpoint of a circle

In figure 9.2 an overview of the frame used to calculate the orientation is given. In this frame vector *Radius* $\underline{R}$ is the vector pointing from the centre of the calculated circle to the position of the MPU9250. The length and direction of this vector is known since the positions from the centre of the circle and the MPU9250 are known. The vector *Normal* $\underline{N}$ points from the centre of the circle to the probe-tip and is the normal of the probe in inverse. The direction of this vector is known, since it makes a 90 degree angle with the plane of the circle and is defined to be oriented in the direction of the negative z-axis. The length of this vector is unknown. The vector *Offset* $\underline{O}$ is the vector from the MPU9250 to the probe-tip. The length of this vector is determine in the previous subsection. The direction of this vector is unknown.

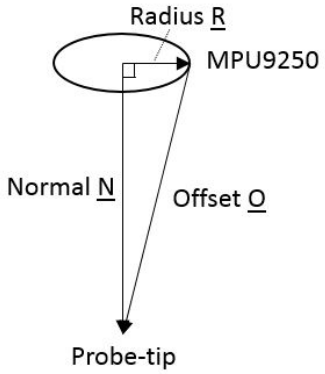


Figure 9.2: Frame for calculation of the orientaion between the MPU9250 and the probe-tip

To determine the length of vector *Normal* the Pythagorean theorem is used, this can be done because the vectors make a right-angle triangle. The length of *Normal* is then given by equation 9.7.

$$\|N\| = \sqrt{\|O\|^2 - \|R\|^2} \tag{9.7}$$

With the length and direction of the *Normal* and the position of the centre of the calculated circle, the position of the probe-tip can be calculated. Now that the position of the MPU9250 and the probe-tip is known the vector *Offset* $\underline{O}$ can be calculated. Now that the vectors $\underline{O}$ and $\underline{N}$ are known with respect to the reference frame, the orientation of the *Normal* and *Offset* with respect to the MPU9250 can be calculated. For this the orientation of the MPU9250 must be taken into account. The orientation is determined by rotation the vectors around the MPU9250 with use of quaternions, described in section 8-A2. The vector is rotated until the orientation of the MPU9250 is (0,0,0), then the direction of the vectors *Normal* and *Offset* is the orientation of the probe and normal with respect to the MPU9250.

B. Implementation

1) Distance estimation: The implementation of method given in section 9-A1 was implemented in Matlab. During the movement with the probe the position of the MPU9250 is read out by Matlab, more on the readout with Matlab in section 10. During the measurement multiple positions were taken and combined in sets of 4 points. Each of these sets was used to calculate a radius. The average of those radius was taken to be the estimated distance between the MPU2950 and the sensor. The Matlab-codes *radius.m* and *CalibrationTipDistance.m* can be found on GitHub as indicated in appendix D.

2) Orientation estimation: The implementation of the orientation determination was also done in Matlab. The position and orientation of the probe were read out when the probe was rotating in a circle around the probe. To calculate the radius and the centre of the circle the function circlefit3D [24] was used. This function gives the centre and radius of the circle. After this the calculations described in section 9-A2 were implemented. The Matlab-code *CalibrationTipOrientation.m* can be found on GitHub as indicated in appendix D.

C. Testing

Due to the fact that orientation determination doesn't work completely, the algorithm to determine the distance and the orientation between the probe-tip and MPU9250 couldn't be tested using the position-system, but it was tested with the use of Matlab.

To test the distance algorithm points laying on a sphere around point (0,0,0) were given as an input to the algorithm. The blue dots in figure 9.3 indicate the given points. The algorithm calculated the centre of the sphere correltly in (0,0,0) (red dot in figure 9.3). The radius in this case was $\sqrt{3}$, which was correct since the given points were laying on a sphere with radius $\sqrt{3}$.

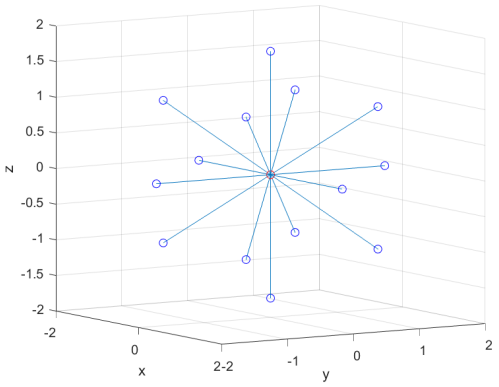


Figure 9.3: Results from distance algorithm; data input (blue dots), calculated centre (red dot) and radius (lines)

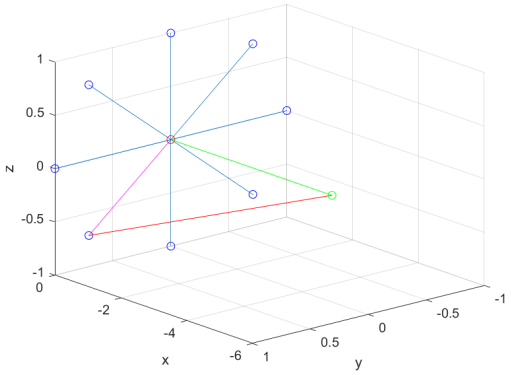


Figure 9.4: Results from orientation algorithm; data input (blue dots), calculated centre(red dot), radius (blue lines), *Radius* R (magenta line), *Normal* N (green line) and *Offset* O (red line)

The orientation algorithm was also tested using Matlab, this was done by giving: 8 points on a circle and the distance between the probe-tip and the MPU9250. With the 8 points the centre of the circle and the radius was determined. Also the orientation of the *Normal* (green line) and the vector *Radius* were determined, see figure 9.4. The length of *Normal* was determined with the length of *Radius* (magenta line) and the given length of *Offset*. Then the position of the probe-tip (green dot) and the vector *Offset* (red line) were calculated. This is all done correctly. The last step of the algorithm, the rotating of the vectors until the orientation of the MPU9250 is (0,0,0), is not done in this code. The testing of rotating with quaternions is already done in section 8-E.

10. Output

A. Theory

The purpose of the position-system described in this paper is to provide the visualisation-software with the position of the probe-tip and the orientation of the probe. Requirement [3,1,1] states that the position must be given in x,y,z and requirement [3,1,3] states that the orientation must be given as a normal vector. The calculation of the position and orientation are described in section 8. Those calculations are done on the microcontroller ATmega328, after the calculations the data needs to be read out with Matlab according to requirement [5,1]. To read out the data from the ATmega328 to Matlab a serial communication is used.

B. Implementation

To implement the serial communication between the ATmega328 and Matlab, Matlab first opens the serial comport and also the ATmega328 opens the serial communication. Then every time Matlab sends a request to read out data from the ATmega328 Matlab waits till the ATmega328 gets to a *Serial.println();* line and then Matlab receive this data written by that line.

When the serial communication is opened, the ATmega328 doesn‘t start right away with sending the position and orientation. It starts with sending this data when the button is pressed and the tracking is started. To let Matlab know that the data is coming, the ATmega328 sends a StartFlag, which is in our case done by the line: *serial.println(F("startStream"));* . After starting the serial communication Matlab read out all the data the ATmega328 sends, but only start to pass through the data after he reads the *startStream*. The data is then passed through to the visualisation-software. The ATmega328 sends his information with beginning to say which variable he sends, for example position x, followed by the value of that variable. This is to prevent a data skew, when one of the send data is not received correctly by Matlab. After using the serial communication the serial comport needs to be closed. The output Matlab script was written with use of the files Main.Script, setupSerial.m and readTemp.m provided by gianluca88 [25].

The Matlab codes *readout.m* and *getArduinoData.m* written to read out data from the ATmega328 can be found on GitHub as indicated in appendix D. The *readOut.m* is the main function from where the communication is started, data is acquired and the communication is stopped. The first part of the code sets up the communication and waits for the start signal coming from the ATmega328. After the signal the readout loop is entered where data is read out into *position* and *normal*. This loop uses the second Matlab file *getArduinoData.m*. In this function the readouts from the ATmega328 are checked for errors and then returned to the main program. There are three things that need to be checked during data acquisition. First is detecting the start of an updated position and normal. The ATmega328 always starts its message of position and normal by sending the x-position, so if this variable is seen the most up to date message is send. If any other value is send first, the reading out happens in the middle of the message and the data will not correspond.

The next thing that is checked for is skew data. If something goes wrong in the communications data might end up in the wrong space in the message, to prevent this from happening every incoming value is counted and if there are more received variables of one kind than of any other a data skew happened. If data skew is detected the measured value is invalid and this is reported to the main program.

The last thing is the detection of a stop signal by the ATmega328, given by the string *stopStream*. This happens when the button is pressed again and the device stops calculating new data. The program will then stop and note the master program that it detected a stop sign and how many measurements it recorded.

C. Testing

The serial communication between the ATmega328 and Matlab was also used to read out other data from the ATmega328 during the development of the position-system. The graphics shown in the previous sections were made with the use of Matlab. This shows that the communication between Matlab and the ATmega328 works fine and requirement [4,1] is therefore met.

11. Implementation on ATmega328

All theories and methods described in the above sections are small parts that must be brought together to form one system that will be programmed onto the ATmega328. This master program will contain multiple functions, an initialisation part, a main loop and accompanying libraries. This section will first describe the libraries used and the changes made to these libraries in order to make them suit our needs. Second are the auxiliary functions that are not described in other chapters, these are needed but so small they do not deserve a separate section. After this the two main actions the ATmega328 will perform are set apart, the initialisation and the main loop. These two subsections are where the real action of the ATmega328 happens, such as acquiring data, communications with the computer and doing calculations. An overview of the system from a hardware programmer point of view is given in figure 11.1. In this figure the three main hardware components are the ATmega328, the MPU9250 and the Computer. The MPU9250 is a combination of a MPU6050 and an AK8963, each has its separate I2C address and setting methods, together these systems provide the accelerometer, gyroscope and magnetometer data. Take note of the difference between the MPU9250 and the MPU6050, the MPU9250 is the overall chip which will be addressed as the general controller and the MPU6050 is mentioned when speaking of the accelerometer and gyroscope unit. The computer is the computer of the user that will use the device. On the computer the position and orientation data arrive through serial interface to Matlab, where the data is used in the visualisation-software (GUI) described in thesis [26]. The ATmega328 runs the initialisation procedure on start-up once and then continues into the main loop, which it will run until shut-down. The main loop uses the three libraries placed on the ATmega328. The ATmega328 communicates with the MPU9250 through an I2C interface. The complete code *TIPSMPU9250* can be found on GitHub as indicated in appendix D and the used libraries are indicated in appendix C.

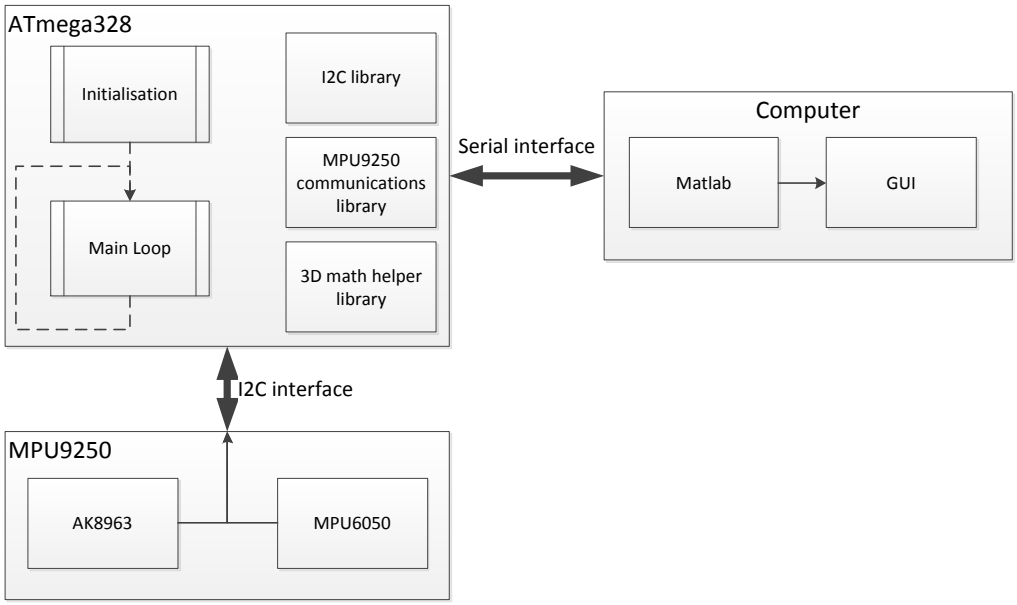


Figure 11.1: System overview from a hardware programmers perspective

A. Libraries

There are three libraries used for this product. These libraries deal with I2C communications between the ATmega328 and the MPU9250, the quaternion three dimensional maths and with the activation and

setting of the MPU9250. The main purpose and functioning of the I2C library is already discussed in section 4 and is not altered for our needs.

The MPU9250 library first offers a register mapping for both the MPU6050 and AK8963, each with their separate I2C address. These register numbers are needed to indicate to the I2C protocol which value needs to be read or written. To properly use the MPU9250 a couple of settings must be set. To start the MPU9250 offers multiple measurement accuracies that can be selected by setting the corresponding registers to the desired accuracy. For this project the most precise, but with the least range, accuracies are set. This is because the device will be handled with relatively slow movements and needs high precision to match the requirements. The other library settings are to communicate with the build in *Digital Motion Processor* (DMP) of the MPU6050. This DMP combines the accelerometer and gyroscope data to generate its own orientation quaternion. Unfortunately, the DMP does not implement the magnetometers in his calculations and there is no documentation on the exact operation that the DMP performs, including filtering, gimball-lock prevention etc. So for this project the DMP is not used to calculate orientation. This gives us the opportunity to calculate our own orientation using chosen filters and calculation methods without being bound to a semi-working processor of which we do not know the faults. Apart from mapping registers, the MPU9250 library also gives I2C functions to read out measurements from the MPU9250. There was a small flaw that was altered in this project. The magnetometer data is coming from two registers, one for the higher byte and one for the lower byte. The function in the library that reads out these registers putted them in reverse order, making wrong raw data readouts. The registers were set in the right order.

The 3D helper math library contains functions and variables that are useful for quaternion calculations. There are three introduced classes, the Quaternion, the VectorInt16 and the VectorFloat. The Quaternion class contains four floats representing the four quaternion variables [w, x, y, z] and both Vector classes contain the three vector variables [x, y, z] in Int16 or float format for the corresponding class. These are used to work with the data coming from the MPU9250. Because the data coming from the MPU9250 is always from the three axes and given in integers, this data is easily put in the VectorInt16 class. The classes also include functions that are common in quaternion maths, such as rotations and normalisations. Some additional functions were made, such as vector multiplications and conversions between the three class types. The math behind these additions are given in section 8.

B. Auxiliary functions

There are some smaller functions in the master program that are needed for the device to function, but that do not require much calibration or precision. These functions are shortly described in this subsection and their implementation in the program is indicated.

The first small function is the built-in timer of the ATmega328 that is used to track the time between calculations. This timer is simply read out at the start of operations and the difference between the previous value of the timer and the current value of the timer gives the corresponding interval time in microseconds. This interval time is used to integrate the gyroscope angular rate to angular displacement and to integrate the acceleration to displacement as described in subsection 8-B. The code for this is implemented as follows:

```
timer = micros();       \\ start timer in initialization

\\ every iteration of the main program

dT = micros() - timer;  \\ update the interval
timer = micros();       \\ reset timer
```

In this code snippet the *timer* saves the readout from the built-in *micros* function that tracks the duration of operation of the ATmega328 since power-up. *dT* is the time interval in microseconds that is used as integration constant and indicates the duration of the calculations and data fetching.

The second auxiliary function is a button debouncer that is needed to use the button described in specification [3,1,8]. A button gives a noisy signal when being pressed and this noise needs to be filtered to a simple on or off state for the device to start or stop. The button must function as a clicker button, which means a press and release of the button changes the on or off state. This was chosen because a press-and-hold button would be inconvenient for a doctor that has to take a long measurement. To implement the debouncing and clicker function a simple piece of code is used that only reads inputs after a small interval in which the button bounces and gives noisy data. The code looks as follows:

```
int buttonDebounce() {
int reading = digitalRead(buttonPin);   // read the state of the
    ↪ switch

if (reading == HIGH && lastButtonState == LOW && (micros() -
    ↪ lastDebounceTime > debounceDelay)) {
if (state == HIGH) { // clicker function
          state = LOW;}
else{
          state = HIGH;}
lastDebounceTime = micros(); // reset timer
}
lastButtonState = reading; // save the reading
return state;
}
```

In this snippet the integer *reading* is the raw input from the button, it is either a 0 or 1, but due to the bouncing of the button, it fluctuates heavily between those two during presses. The first if-statement tests if the button went from low to high value and if we've waited long enough to ignore any noise on the circuit. To do this it compares reading to the integer *lastButtonState* and compares the interval time *(micros() - lastDebounceTime)* to the user set *debounceDelay*. The interval is calculated similar to that of the interval described in the previous paragraph. If these conditions are met the *state* of the button is updated to the reverse of its previous *state*, this ensures that the button works as a clicker button. Last the *reading* is stored in *lastButtonState* for the next iteration of the function and *state* is returned to the main program. The *debounceDelay* is set to 50 in the device, meaning that the button can change state every $50\mu s$.

The last function discussed here is a small function to convert floating point numbers to integers. This function will be used as little as possible, because it induces round-off losses. It is still usable for instance after noise filtering, since the input variables were integers and rounding them back to integers after the filter only removes induced information that was not actually measured in the beginning. The function is made as follows:

```
int float2int(float flo) {
int in; // output integer
if (flo < 0) { // check if negative number
        in = (int)(flo - 0.5); //if negative round to nearest negative
}
else {
        in = (int)(flo + 0.5); //if positive round to nearest positive
}
return in;
}
```

In this snippet the input floating point *flo* is checked on being positive or negative and on being negative gets subtracted by 0.5 to round it to the nearest integer (since floating point to integer conversion always rounds down). For positive numbers, the opposite is done.

C. Initialisation

When starting up the ATmega328 will first execute its initialisation routine. In this routine all devices must be prepared for operations and all buffered values must be reset to their initial value. An overview of the initialisation routine is given in figure 11.2.

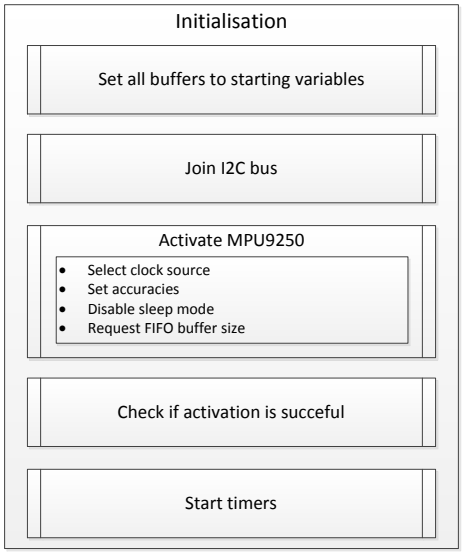


Figure 11.2: Initialisation routine overview

The first thing that is done is joining the I2C bus using the I2C library and then activating the serial interface by setting the correct baut rate. After this the I2C interface is used to activate the MPU9250 by selecting the clock source for the MPU9250, setting the right accuracies for the gyroscopes and accelerometers and finally disabling the sleep mode of the MPU9250. After this the size of the *First In First Out* (FIFO) buffer is requested from the MPU9250. This FIFO buffer will contain the accelerometer and gyroscope data that will later be requested from the MPU6050. The AK8963 needs no activation, since it will be used in a mode that requests the readout directly at the data gathering moment. The last operation performed is initialising the timers for the button debouncer and interval counter as described in the previous subsection. During the initialisation of the MPU9250 a check is performed to see if the connection was successful, otherwise the main routine is not executed and an error message is send to the computer.

D. Main loop

The main loop is the part of the master program that will be run after initialisation until power down. During this time the ATmega328 has three main tasks to perform: gather measurements from the MPU9250, use these measurements to calculate orientation and position and send the calculated orientation and position to the users computer. The calculations only needs to be done if the MPU9250 has new data ready in the FIFO buffer, at which moment the MPU9250 will set an interrupt flag. Because of this the main loop consists of two parts: Firstly waiting for new data and reading that data in, secondly the calculations and sending to the computer are done. An overview of the main loop routine is given in figure 11.3.

The implementation of the main loop contains two subroutines, one that checks if the interrupt flag is set and then gathers the new measurements from the MPU9250 and a calculations subroutine that waits on a flag to be set by the other subroutine that indicates that the new measurements are ready for calculations.

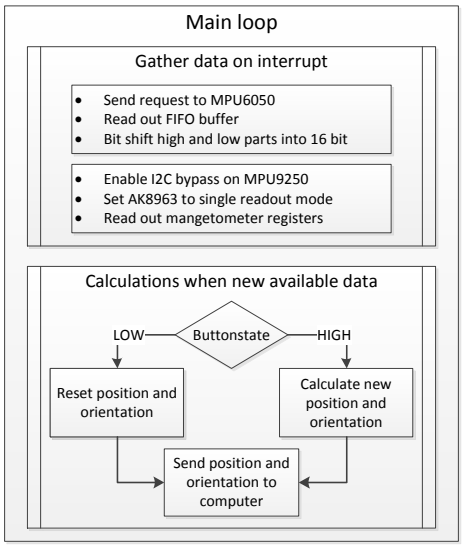


Figure 11.3: Main loop routine overview

In the Data gathering subroutine the ATmega328 must communicate with both the MPU6050 (accelerometers and gyroscopes) and AK8963 (magnetometers). This way of communicating is explained in section 4-B. After the measurements are read from the MPU6050 and AK8963, the ATmega328 will begin calculations on these measurements to produce an orientation and position. Before this is done, the current state of the clicker button is checked. If the button is not activated the device will not give an output orientation and position and the starting values of these must be determined. For position and orientation this is simply the origin (0,0,0). However, the chip is not level at this moment and when measurements start the orientation should not suddenly jump to that position. In subsection 8-A3 it is explained that only the change in the orientation quaternion is required to determine the orientation of the device and the translation. When the button is pressed the first rotation should be zero (not the orientation) and thus the first orientation quaternion determination should not differ from the starting orientation quaternion. In order to start this quaternion at the right orientation the accelerometers and magnetometers are used to determine orientation as explained in section 7. The output orientation from this will be used as starting orientation from where the rotation calculations can start. If the button is pressed the position and orientation will be tracked. Doing this is explained in sections 7 and 8. The last thing that is done in the main loop is sending to the computer which is explained in section 10.

12. Testing overall system

In the previous sections tests were done for all sub-parts and the results were readily discussed. The final step is now to test the entire system to see if the sub-parts work together correctly. As explained in section 7 the orientation cannot be determined correctly for yaw rotations and as said in section 8-E the position cannot be correctly detected with the current filter. To measure the entire system these values are needed, thus making a complete test not possible at the moment. The method to test, however, can already be described.

A. Testing plan

To test the overall system, first the communication between the ATmega328 and the MPU9250 and the communication between the ATmega328 and Matlab is tested. To do this the values of the accelerometers, gyroscopes and magnetometer are read out. Thereafter the orientation and displacement can be tested in two different settings.

In the first setting the MPU9250 is clamped with a microclamp. First the MPU9250 is rotated around its axis. Thereafter the MPU9250 is rotated around with the microclamp in a circle, the base of the microclamp remaining in the same position. Around the base of the microclamp there are positions marked corresponding to a certain angle, which will be used as reference for measurement points. These circles are done with the MPU9250 in different orientations. During these movements the position of the MPU9250 and probe-tip and the orientation of the imaginary probe are read out by Matlab. After a circle is completed the results tells if the position of the sensor also follows a circle and if it comes back to the same point as where it started. This outcome determines how well the position is tracked and small noise can be seen as points that differentiate of the circle.

In the second setting the MPU9250 is hold by the tester. The tester follows the outline of a rectangular sheet with the MPU9250 in different orientations. The tester follows the rectangular in different directions. The position of the MPU9250 and probe-tip and the orientation of the imaginary probe are read out by Matlab. After completing one lap around the sheet of paper the output data should also describe the outline of a rectangle. The amount of misshaping gives the amount of error in position that the probe has.

An overview of the test set-up is given in figures 12.1 and 12.2. In these figures the microclamp holding the MPU9250 can be seen. There is a pointer also clamped that is used to see if the clamp (and thus MPU9250) are aligned with one of the angles. The second set up is simply the tester holding the chip and tracing around the white paper sheet that can be seen in the figures.

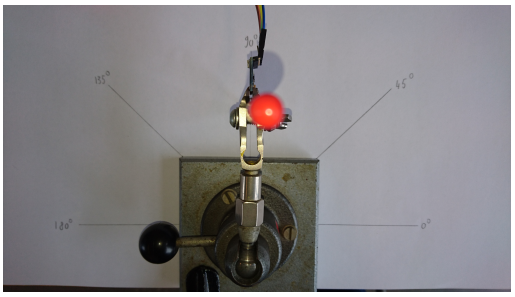


Figure 12.1: Top down view of testing

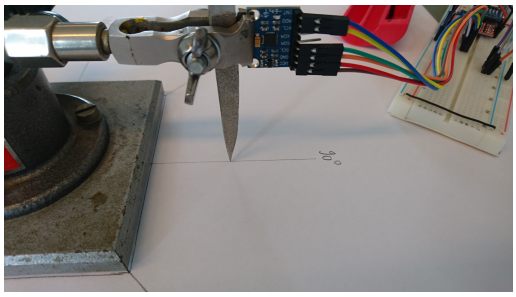


Figure 12.2: side view of testing

B. Results

The first step of the testing plan indicates if the communication between the ATmega328, MPU9250 and Matlab is working well. This can be done by comparing the readout done by Matlab with the expected values, if the values are as expected the communication works fine. This test was successfully performed. The other tests were not done due to the known errors in sections 7-C and 8-E.

13. Discussion

There are two problems that need to be solved in order for the device to work correctly, the determination of the yaw angle and the determination of displacement.

First the problem with yaw angle determination. As explained in section 7-D the magnetometers are needed to determine the yaw angle. Due to inaccurate calibrating, the yaw angle didn‘t give the right angle. To fix this there are two solutions proposed, either design a better calibration algorithm that can be used before each operation or use the Madgwick filter. The Madgwick filter combines the orientation techniques in a more accurate but also more complex way than the complementary filter and immediately produces a quaternion orientation. The implementation will cause problems with the programmable space on the ATmega328 however. So a larger microcontroller is suggested. Another option is to send the raw data from the device to the users computer and let the computer calculate the orientation. This does come with more delay and possibly unwanted stress on the users end. The best solution to get the right yaw angle is the combination of the two solutions, so use a better calibration algorithm and use the Madgwick filter.

Solving the problem with the displacement does not lie in the hardware but in the high-pass filter that is applied. It is suggested in section 8-G that a more dedicated and larger filter could be used. The downside to this is the delay that will be induced and the higher demand of programmable space. This can be solved in the same way as for the yaw calculation by using a larger microcontroller or the users computer. It might also be possible that other filtering techniques provide better results.

14. Conclusion

The functional requirements of this paper were to design a position-system that gives the position and orientation of a probe during measurements with the probe by a doctor in a hospital or practice. The theoretical basis for these requirements are given in this paper and is tested and proved to work for ideal input variables. The roll and pitch determination techniques work in theory as well as in practise and meet the requirements. Unfortunately it became clear that the determination of the yaw angle of the device was not precise enough. For the other part of position determination, the displacement, the filtering and integration work well to counter drift. But the actual displacement is off with a resulting total displacement lower or higher than the actual distance travelled.

The error in yaw angle is either due to the wrong calibration of the magnetometers or due to the hardware being unsuitable for these accuracy demands. The theoretical approach of the complimentary filter does pass the requirements of combining magnetometer angle determination with gyroscope angle determination. The problem lies in the angle determination of the magnetometer data, which is incorrect due to the raw data not measuring the earth magnetic field correctly. It is therefore suggested to use a better algorithm for magnetometer offset determination and correction. This calibration algorithm will take up more computation time and space on the microcontroller. One other possible improvement is the use of the Madgwick filter that combines the magnetometers and gyroscopes with a more precise method. This filter is more complex and requires more programmable space than the ATmega328 offers. One implementation option for now is to not measure yaw and instruct the user to not rotate the device around the z-axis. Although it is primitive and errors due to the user are induced, this will give a orientation output that can be used to determine position.

The error in the displacement is due to the filtering and integration method, those are not accurate enough, the hardware does give enough precision. This must thus be solved by a better mathematical solution for filtering and integrating. More precise filters and integrators will take up more computational time and space on the microcontroller and induce a delay in the output. With better techniques it is expected that the required accuracy can be met. For now the device can be used to indicate displacement without drift, but the actual distance will be off.

The system in theory is still better than the use of optical mouse sensors. This is because optical mouse sensors will not give three dimensional data, which is useful if interpolation over objects is needed. Also the use of these sensors would obscure the probe-tip as they must be placed on the surface of the subject as well, thus making the tip far larger. Lastly these sensors require that there is always surface under them. When tracing weird shapes, such as limbs or organs, the sensors might miss this surface and the position can then not be tracked.

With the use of another calibration method and the Madgwick filter there will be a good change that the determination of the yaw angle will work well in future research. Also the use of a better filtering and integration method will contribute to a better accuracy of the displacement determination. With those adjustments to the current system a good step toward a working system can be made in future research.

REFERENCES

[1] M. H. Truong and A. W. Kremers, "Calibration procedure for complex permittivity extraction using open-ended coaxial probe," Bachelor thesis, TU Delft, June 2016.

[2] J. Smisek, M. Jancosek, and T. Pajdla, *Consumer Depth Cameras for Computer Vision*. London: Springer, 2013, ch. 1, pp. 3–26. [Online]. Available: http://download.springer.com/static/pdf/368/bok%253A978-1-4471-4640-7.pdf?originUrl=http%3A%2F%2Flink.springer.com%2Fbook%2F10.1007%2F978-1-4471-4640-7&token2=exp=1461324572~acl=%2Fstatic%2Fpdf%2F368%2Fbok%25253A978-1-4471-4640-7.pdf%3ForiginUrl%3Dhttp%253A%252F%252Flink.springer.com%252Fbook%252F10.1007%252F978-1-4471-4640-7*~hmac=5ca6961bc1271000bb63b437dd513c0299729a191f493117aa309ed6aed4bd67

[3] G. Ehnholm and E. Vahala, "Method and apparatus for determining probe location," U.S. Patent US 08/958,309, 10 27, 1997. [Online]. Available: https://www.google.com/patents/US5882304

[4] T. Ng, "The optical mouse as a two-dimensional displacement sensor," *Sensors and Actuators A: Physical*, vol. 107, no. 1, pp. 21–25, 10 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924424703002565

[5] T. H. H. U. . T. M. . T. Sato, "Portable orientation estimation device based on accelerometers, magnetometers and gyroscope sensors for sensor network," *Multisensor Fusion and Integration for Intelligent Systems, MFI2003. Proceedings of IEEE International Conference on*, Augustus 2003.

[6] Mpu-6050 six-axis (gyro + accelerometer) mems motiontracking devices. [Online]. Available: http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/

[7] Mpu-9250 nine-axis (gyro + accelerometer + compass) mems motiontracking device. [Online]. Available: http://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/

[8] J. Rowberg. Mpu9150 library. [Online]. Available: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU9150

[9] Arduino. i2c library. [Online]. Available: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/I2Cdev

[10] "Mpu-9250 product specification," April 2014. [Online]. Available: http://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/

[11] T. Ozyagcillar, "Calibrating an ecompass in the presence of hard- and soft-iron interference," *Freescale Semiconductor, Inc*, November 2015.

[12] Y. Matselenak, "Advanced hard and soft iron magnetometer calibration for dummies," *DIY Drones*, June 2014.

[13] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering 82*, 1960.

[14] Kalman filter vs complementary filter. [Online]. Available: http://robottini.altervista.org/kalman-filter-vs-complementary-filter

[15] S. O. Madgwick, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays," in *2011 IEEE International Conference on Rehabilitation Robotics*, June 2011.

[16] Accelerometers. [Online]. Available: http://www.hobbytronics.co.uk/accelerometer-info

[17] My imu estimation experience. [Online]. Available: https://sites.google.com/site/myimuestimationexperience/sensors/magnetometer

[18] Simple and effective magnetometer calibration. [Online]. Available: https://github.com/kriswiner/MPU-6050/wiki/Simple-and-Effective-Magnetometer-Calibration

[19] J. B. Kuipers, "Quaternions and rotation sequences: A primer with applications to orbits, aerospace and virtual reality," *Princeton University Press*, 1999.

[20] Gimbal lock. [Online]. Available: https://en.wikipedia.org/wiki/Gimbal_lock

[21] L. D. Slifka, "An accelerometer based approach to measuring displacement of a vehicle body," *University of Michigan, Dearborn*, 2004.

[22] J. L. Weiner and G. R. Wilkens, "Quaternions and rotations in e4," *THE MATHEMATICAL ASSOCIATION OF AMERICA*, January 2005.

[23] Center and radius of a sphere from four points. [Online]. Available: http://www.abecedarical.com/zenosamples/zs_sphere4pts.html

[24] circlefit3d, fit circle to three points in 3d space. [Online]. Available: http://nl.mathworks.com/matlabcentral/fileexchange/34792-circlefit3d-fit-circle-to-three-points-in-3d-space
[25] Arduino and matlab: let them talk using serial communication! [Online]. Available: http://www.instructables.com/id/Arduino-and-Matlab-let-them-talk-using-serial-comm/?ALLSTEPS
[26] B. Hettema and W. Bouwmeester, "Simulation environment for an open-ended coaxial probe and visualisation of permittivity measurements," Bachelor thesis, Delft University of Technology, June 2016.
[27] J. Rowberg. Mpu6050 library. [Online]. Available: https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050

APPENDIX

A. *Figures*

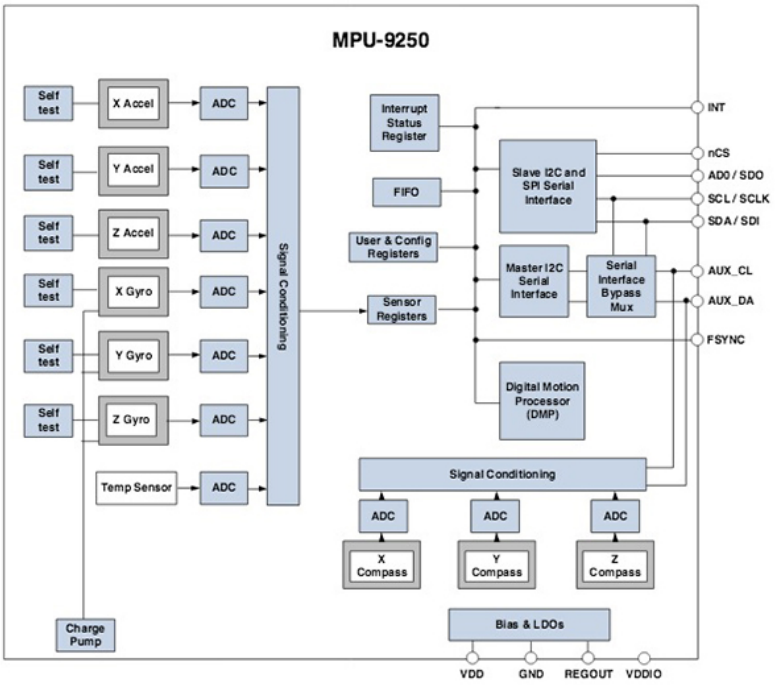


Figure A.1: Block diagram of MPU9250

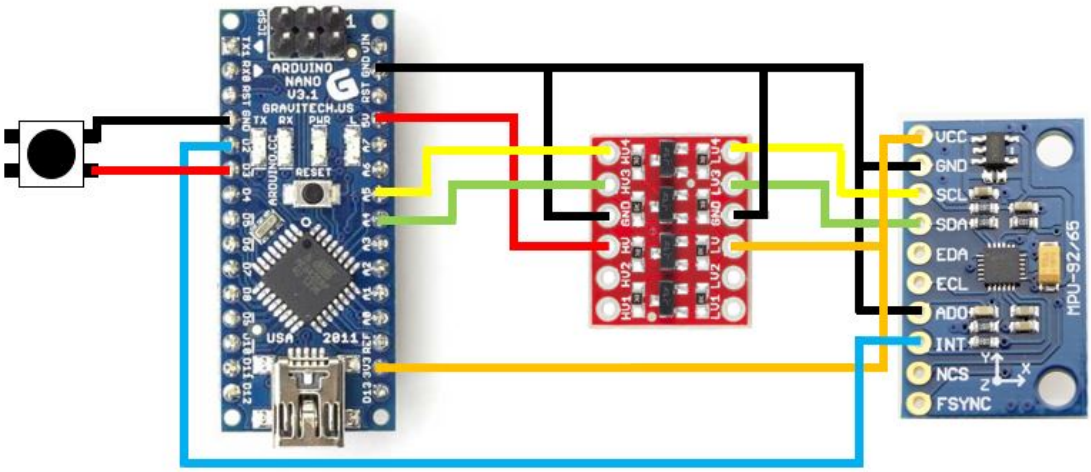


Figure A.2: Positioning pin diagram

B. Tables with results complementary filter

Table III: Value of some sample during a roll/pitch rotation of 90 degrees

Sample	Roll (degrees)	Pitch (degrees)
200	88.3543	88.6656
210	88.4393	88.7014
220	88.4555	88.7157
230	88.4491	88.7327
240	88.4372	88.7423
250	88.4205	88.7479
260	88.4194	88.7555
270	88.4182	88.7427
280	88.4107	88.7373
290	88.3897	88.7300
300	88.3633	88.7295

Table IV: Outcome of yaw calculation in degrees with Matlab, X are situations that don‘t occur in reality

mx	my	mz	pitch = 0° roll = 0°	pitch = 90° roll = 0°	pitch = 0° roll = 0°
1	0	0	0°	X	0°
-1	0	0	180°	X	180°
0	1	0	-90°	-90°	X
0	-1	0	90°	90°	X
0	0	1	X	0°	90°
0	0	-1	X	-180°	-90°

C. Used libraries

- I2Cdev.cpp [9]
- I2Cdev.h [9]
- MPU9150.cpp [8]
- MPU9150.h [8]
- MPU6050_6Axis_MotionApps20.h [27]
- MPU6050_9Axis_MotionApps41.h [27]
- helper_3dmath.h [27]
- circlefit3D [24]
- Main.Script, setupSerial.m and readTemp.m [25]

D. Source code

The source code for the ATmega328 and Matlab files can be found on Github at https://github.com/LucvW/PositioningTIPS . The repository contains the following files:

- *TIPSMPU9250.ino*, the ATmega328 source code
- *readOut.m*, the main read out program
- *getArduinoData.m*, the data read out algoritm
- *testyaw.m*, the program used to test yaw
- *radius.m*, the radius and centre algorithm
- *CalibrationTipDistance.m*, the test of radius and centre algorithm
- *CalibrationTipOrientation.m*, the test of orientation algorithm