



Islamic University of Technology (IUT)

Department of Computer Science and Engineering (CSE)

Ambulatory Human Posture Detection using MARG Sensor Arrays and Madgwick Filter

Authors

Mohammad Ishrak Abedin, 160041051

Saad Bin Ashraf, 160041068

Rizvi Ahmed, 160041076

Supervisor

Dr. Abu Raihan Mostofa Kamal

Professor, Department of CSE

*A thesis submitted to the Department of CSE
in partial fulfillment of the requirements for the degree of B.Sc.*

Engineering in CSE

Academic Year: 2019-2020

March, 2021

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by Mohammad Ishrak Abedin, Saad Bin Ashraf and Rizvi Ahmed under the supervision of Dr. Abu Raihan Mostofa Kamal, Professor of Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Gazipur, Dhaka, Bangladesh. It is also declared that neither this thesis nor any part of it has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others have been acknowledged in the text and a list of references is given.

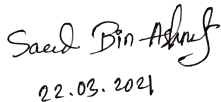
Authors:



22/03/21

Mohammad Ishrak Abedin

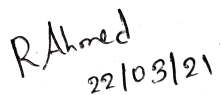
Student ID: 160041051



22.03.2021

Saad Bin Ashraf

Student ID: 160041068

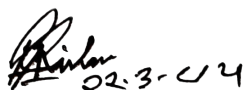


22/03/21

Rizvi Ahmed

Student ID: 160041076

Supervisor:



22.3.21

Dr. Abu Raihan Mostofa Kamal

Professor

Department of Computer Science and Technology

Islamic University of Technology



Acknowledgement

We would like express our gratitude towards IUT authority for granting us the fund and providing assistance required to implement our proposed system. We are indebted to our professor, Dr. Abu Raihan Mostofa Kamal for providing us with insightful knowledge and guiding us at every stage of our journey. Finally, we would like to express our heartiest appreciation towards our family members for their continuous support, motivation, suggestions and help, without which we could not have achieved the scale of implementation that we have achieved.

Abstract

This report proposes a novel and low cost method of calculating and collecting real time human posture and orientation data with the help of MARG (Magnetic, Angular Rate and Gravity) sensors, micro controllers, Madgwick filter and an on board mobile computer. With the help of MARG sensor arrays, the proposed method overcomes the movement and environmental restrictions of traditional vision based posture data collection systems. Madgwick filter, being very light weight but accurate, allows the development of the whole system at a cost cheaper than other similar sensor based alternatives, using commercially available simple hardware. The low cost implementation, accuracy and mobility provided by the system can open new opportunities for the use of such systems in research, medical, input modality, sports, military, computer graphics and other fields, specially in places where cost and mass production are crucial factors.

Keywords— MARG, IMU, Madgwick Filter, Human, Motion, Posture, Orientation, Ambulatory, Cost Effective

Contents

1	Introduction	1
2	Literature Review	2
3	Limitations of Existing Systems	3
4	Proposal, Contributions and Utility	4
4.1	Proposal	4
4.2	Contributions	5
4.3	Utility	5
5	Background Study and Sensor Mapping	6
6	Approach	8
6.1	Orientation measurement using SOU and Madgwick filter	9
6.1.1	Madgwick Filter	11
6.2	Data Collection and Integration using On Board Computer	15
6.3	Data Cleaning	16
6.4	Absolute to Relative Conversion	16
6.5	Cartesian Conversion	17
6.6	Smoothing	18
6.6.1	Convolutional Filters	18
6.6.2	Adaptive Filters	19
6.6.3	Frequency Domain Filters	19
6.6.4	Predictive Filters	20
6.7	Data Mapping and Export	20
6.8	Eular Discontinuity Filtering	22
6.9	Visualization	22
7	Experimentation	23
7.1	IMUs (MPU 9250) and Arduinos (SOU)	23
7.2	Raspberry Pi and Driver	24
7.3	Pose Calibration and Reading	26
7.4	Data Post Processing	26
7.4.1	Data Cleaning	26
7.4.2	Data Summarization	27

7.4.3	Absolute to Relative Conversion	27
7.4.4	Cartesian Conversion and Smoothing	28
7.5	Mapping, Discontinuity Solution and Visualization in Blender	32
8	Evaluation	33
8.1	Error Measurement and Error Propagation	33
8.1.1	Per Sensor Error Measurement	34
8.1.2	Error Propagation	34
8.2	Cost	35
9	Conclusion and Future Work	35

List of Tables

1	Nodes Relative Parents	8
2	Output Format of a Single SOU	15
3	Arduino Nano and MPU 9250 pin mapping	23
4	Data Format after Time Stamping	25
5	Static Error per Axis	34
6	Dynamic Error per Axis	34
7	Per Node Propagated Error	35

1 Introduction

Human posture recognition is the process of collecting human limb orientation data so that the posture can later be reconstructed and reused from that data. Human motion capture and analysis have use cases in various fields [1, 2]. They are prominently used in computer graphics industries to produce animations and perform visual effects. Motion capture data can be used in research fields in gait analysis and behavioral pattern detection. They also contribute in medical and biomedical engineering in both treatment of physical disabilities, rehabilitation and manufacturing and testing of prosthetic limbs. Motion capture data are also used in virtual and augmented reality and in game and interaction design. They can also be used in the measurement of real time athletic and sports data. Motion capture data are also useful in robotics for calibration and maintenance of large moving machines.

The methods of collecting human orientation data can be divided into two main approaches, with the help of computer vision and via body mounted Inertial Measurement Unit (IMU) or MARG (Magnetic, Angular Rate and Gravity) sensor arrays. While the aforementioned vision based approach is getting more popular day by day due to the availability of normal and depth cameras at an affordable price, they have some inherent problems such as mobility restriction, dependency on environmental factors and lower accuracy as discussed later in this report.

While sensor based measurements can be accurate if done properly, they require a fusion system to fuse the data from different sensors, that are accelerometer, gyroscope and magnetometer. Among these, most commonly used filters are the Kalman filter [3] and the complimentary filter proposed by Mahony et al. [4]. While Kalman filter suffers from the necessity of excessive calculation, complimentary filter is not validated for MARG sensor arrays, a requirement for accurate calculation of orientation.

With the help of the filter designed by Sebastian Madgwick [5], we propose a novel sensor based system in our report that can accurately calculate human posture data while allowing complete mobility. Madgwick filter, being efficient for low powered devices, enables us to perform the whole orientation calculation using Arduino Nanos, cheap and cost efficient micro controllers. As a result, we are able to propose a complete and accurate motion capture system that can calculate posture of major human limbs at a very affordable price, built using only commercially off the shelf available hardware and open source and/or free software.

2 Literature Review

Study of contemporary literature indicates that most of the recent motion capture technologies are based on computer vision based systems and being extensively studied for a long period of time [6, 7, 8, 9, 10, 11, 12]. Though the process of using IMU and MARG sensors for human motion capture came into literature significantly long ago [13, 14], no significant work describing a full body implementation could be found. Most of the sensor based motion capture systems are built using Kalman filter or one of its derivatives [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 14, 25]. However, the aforementioned works are either hypothetical [14], or implements partial body segment motion capture [22, 23, 24].

Another major problem with Kalman filter is that, even in its base form, it is quite resource expensive and accuracy is not better than Madgwick filter [5]. Moreover, Kalman filter is more suitable for displacement measurement rather than orientation measurement as can be seen from the works of Zhao et al. [25] and Lee et al. [19].

Some authors proposed improvement of the accuracy of Kalman filter by performing modification to it. For instance, Yun et al. [15, 17] and Zhang et al. [16] tried to introduce a quaternion based Kalman filter. However, these works are again mainly focused towards displacement calculation than orientation calculation. H. Ahmed and M. Tahir [21] proposed an acceleration dependent weighted axis based Kalman filter to improve its accuracy, but again it is focused towards displacement data.

Most of the sensor based implementations, as stated before, are generally focused towards partial body orientation measurement and usually used in biomedical engineering, physical exercise measurement [20] and musculoskeletal injury treatment and recovery therapies [26, 27]. The use of Madgwick filter in the domain of physical rehabilitation is significant.

The resource expense of Kalman filter [3] and invalidation of Mahony et al. [4] for MARG sensors make them unsuitable for the implementation in a low cost motion capture system. Sebastian Madgwick, in his report, stated, “*This level of accuracy may be sufficiently effective for applications such as human motion capture*” [5] due to the light weight and accuracy of the filter. So, we propose our method based on the implementation of Madgwick filter to fuse the data of MARG sensor arrays to collect human posture data.

3 Limitations of Existing Systems

As pointed out in literature review, most of the existing system for human body orientation and motion detection relies on computer vision. Some utilizes monocular vision [6, 7], while others rely on depth maps generated by depth sensors [8, 9] or combine depth maps with RGB images [10, 11, 12] to process and get the final body orientation. However there are couple of problems with the vision based approach for the reconstruction of human body orientation and posture from captured images.

1. **Constrained Environment:** Vision based systems rely on cameras and sensors to capture the motion images. These cameras remain generally static to their positions to capture the changes in motion accurately and then use those information to recreate the overall posture of the body. For a proper and accurate motion capture, the cameras cannot be moved arbitrarily. They might be mounted on a rig and follow a predefined path, but even so, capture of all motions from necessary angles is not possible. Hence, it hinders the mobility of the whole system and constrains which motions can be captured and which cannot be. For instance, a traditional setup cannot capture an athlete running in a course unless it is mounted on a rig that circuits the whole course along with the athlete at the same velocity.
2. **Ambient Light Dependency:** For accuracy and better precision, vision based system relies on both depth maps generated generally using infrared based depth sensors and RGB images captured using traditional camera. In a dark or dimly lit environment, traditional cameras suffer to capture good quality images which acts as a hindrance towards accurate calculation for the vision based systems. So, a well lit environment is a prerequisite for proper motion capture using vision based systems.
3. **Trackers:** In most cases, just depth and RGB cameras are not enough for precise tracking and external trackers, such as tracker dots or markers are placed, as shown in figure 1. This, in some cases, can hinder human natural motions and introduce restrictions in the overall process.



Figure 1: Motion Capture Suit with Tracking Dots

4. **Occlusion and Background Clutter:** One of the main scenarios where vision based tracking

suffers is where there are multiple persons and the background is cluttered. In case of multiple persons, both target and non-target, vision based systems fail to discern who should be tracked and who should not be, which creates artifacts in the final result. If the background is of same color as the cloth of the target actor or if the person blends partially or completely with the background, such clutters also decrease the performance of vision based motion tracking systems to some extent.

5. **Cost:** Most high accuracy vision based tracking systems are really expensive and require both costly hardware and proprietary software to capture the information and transfer them to the desired system or project.

Other than vision based motion capture systems, there are other sensor and IMU (Inertial Measurement Unit) or MARG based motion capture systems, such as *Rokoko Smartsuit Pro*. However, these solutions are both expensive and as already mentioned before, no open source solution or significant documentation on sensor based motion tracking could be found.

Compared to these, we propose a method of using commercially available hardware and open source or free programs and software to build a sensor based system that is both cheap and accurate and capable of providing full body human orientation without hindering any mobility or being dependent on external environmental factors.

4 Proposal, Contributions and Utility

4.1 Proposal

Following what is already discussed in the aforementioned sections, we propose a sensor based, cost efficient motion capture system using Madgwick filter that can record human limb orientation and motion on the go, which can either be remotely transferred or collected later for various implementations. Our focus here is on the architecture and working methodology of the system and not directly on the devices associated with it. Following the proposed methodology, the components and programs of the system can be swapped out or upgraded or changed depending on the application criteria of the system.

The following two subsections, subsection 4.2 and 4.3 discuss about the contributions and utility of our system respectively. Subsequently, section 5 discusses about the theoretical background behind human posture and limb orientation measurement using kinematics equations and motion sensors. Section 6 covers the mathematical concepts behind our proposed system and following that, section 7 covers the experimental procedures. Lastly, section 8 contains the evaluation of the proposed system.

4.2 Contributions

The major sectors where we believe our proposed method of human posture system can contribute are listed below.

- A sensor based orientation measurement system that is not dependent on external environmental factors such as lighting conditions, background clutter and occlusion and does not need a constrained preset environment.
- Provides full mobility to the users while using the system, allowing more complex and natural human motions to be captured.
- A system that can be built with readily available cheap commercial hardware such as Arduino, MARG sensor arrays and Raspberry Pi.
- A system that is completely modular and can be extended to accommodate even higher degree orientation measurements and supports orientation capture of beings and things other than humans, such as non-bipedal animals or machines.
- A system that can be completely built with open source software and programs.

4.3 Utility

The major sectors where we believe our proposed system can be utilized, but not restricted to are given below.

- **Research and Data Science:** Any sector where contemporary motion capture systems are used, such as *Microsoft Kinect*, *Intel RealSense* Camera or others, for example, in fields such as Gait analysis or other human posture based research, our system should be able to facilitate similar applications with equal or more accuracy. Machine learning and data science applications or researches that need human posture, orientation and movement data, can use our system for data collection and analysis, rather than camera based systems. It can provide more accurate and granular data than generally available vision based alternatives.
- **Biomedical Engineering and Medical Sector:** Our system can be utilized in the therapy of patients suffering from musculo-skeletal disorder or other physical disabilities along with the validation and control of prostheses [26, 27].

- **Computer Graphics and Animation:** Both vision and sensor based systems are already in use for capturing human and animal motion capture data in animation industries. Our system can provide a cheaper method of achieving such tasks which would specially be helpful for individual developers and small studios.
- **Sports and Athletic Measurement:** Calculation of the movements of sports persons or athletes require very precise measurement at high sampling rate for a longer duration of time, which requires expensive vision based solutions. We believe our system can provide a cheaper alternative solution to this problem.
- **Interaction Modality, Gaming and Virtual Reality:** Human orientation can serve as an alternate input method for computers and are extensively used in virtual reality and gaming, which our system can provide the required functionality for.
- **Military and Sensitive Real Time Calculation:** Study of the motion of soldiers, divers or other personnel who work at high risk or congested environments require sophisticated technology. We believe that our system should be able to provide the basic functionalities to capture the motions in such situations.

5 Background Study and Sensor Mapping

Our goal is to calculate the orientation of each of the limbs of a human body and use those to recreate the posture at any given instance of time. The posture from the limb orientations can be derived using Forward Kinematics (FK), which is basically a series of transformations to find the final position of an end-effector in a serial chain [28, 29]. We can represent the human body using a ball and stick model as shown in figure 2.

Each green ball node in figure 2 denotes an origin of rotation, where as the red lines show the connections or links between joints. Node 'N0_0' is considered as the anchor point or the center of calculation and only its rotation is absolute. Every other node receives a relative rotation with respect to its previous node, which can be found out using the red lines shown in figure 2. For instance, rotation of 'N1_1' is relative to 'N1_0' and rotation of 'N4_0' is relative to 'N0_0'. This means, if we are calculating absolute rotation from a reference pose, for instance, the one in 2, then if 'N1_0' rotates by 90° downward, then the absolute rotation of 'N1_1' would also be 90° downward, where as the relative rotation of 'N1_1' would be 0° , since this joint actually had not rotated. We can utilize the relative parent information of the nodes to convert these absolute rotations to node specific relative rotations. In order to find the relative rotation of a node, we just need to subtract its parent's absolute rotation from its absolute rotation. These

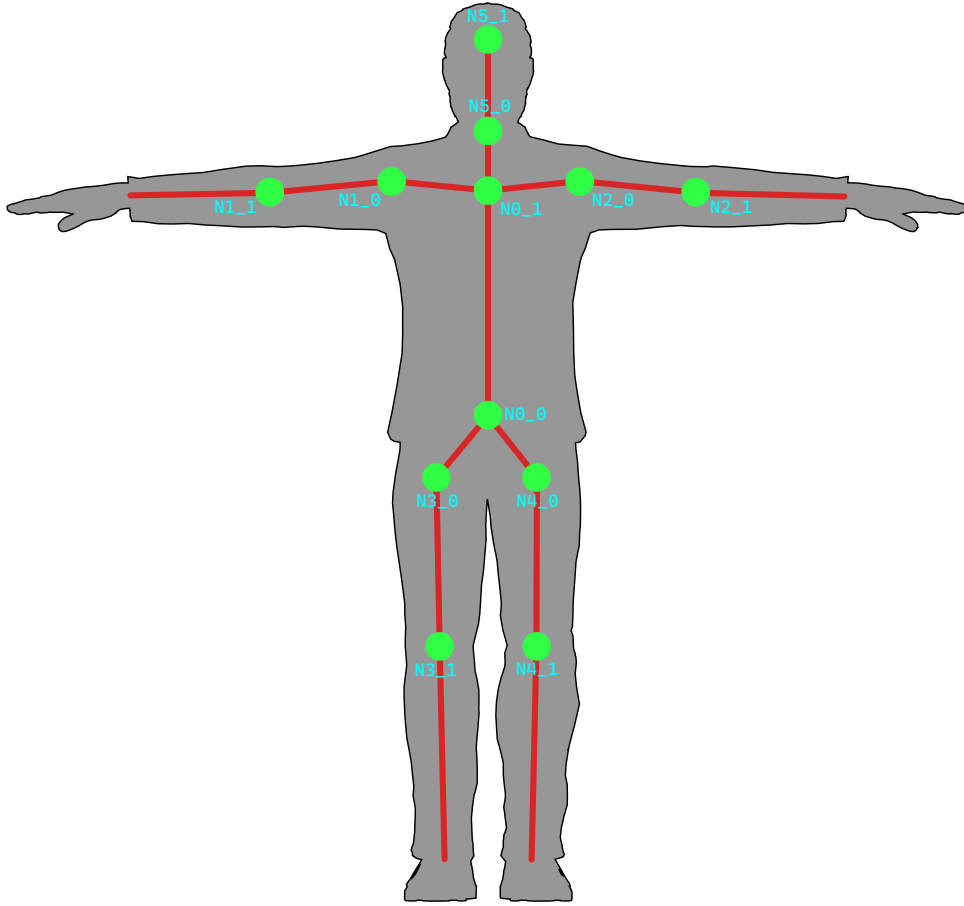


Figure 2: Human Body Ball and Stick Model

rotation information of each of the human body joint is sufficient enough to recreate posture at any given instance of time.

We measure the absolute orientation of each of these nodes or points in human body by placing a sensor node there and then converting the absolute rotations to relative rotations in the post processing stage. A sensor can be placed anywhere along the joint and the line or bone connecting it to the next joint and it would give the same reading for that joint. For instance, the sensor dedicated towards the measurement of right elbow joint orientation, 'N1_1' can be placed anywhere between the elbow joint till the wrist joint. The nodes are named in a way to match the positioning of the sensor nodes. For even more granularity of data, more nodes can be placed within the spine connection or the connection can be further extended to calculate orientation for wrists and feet. For our purpose, we have considered the 12 nodes or sensor placements as shown in figure 2. Their relative parent nodes are shown in table 1.

Node	Parent	Node	Parent
N0_0	Anchor	N0_1	N0_0
N1_0	N0_1	N1_1	N1_0
N2_0	N0_1	N2_1	N2_0
N3_0	N0_0	N3_1	N3_0
N4_0	N0_0	N4_1	N4_0
N5_0	N0_1	N5_1	N5_0

Table 1: Nodes Relative Parents

6 Approach

Our approach to this problem begins with the collection of limb orientation data using *Singular Orientation Units (SOUs)* and after following different stages, ends with the final utilization of the data which can be used for various purposes. A logical flow of processing is shown in figure 3.

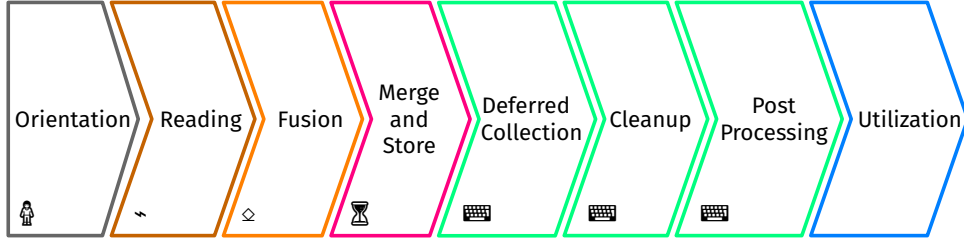


Figure 3: Logical Processing Pipeline

Among these stages, orientation, reading and fusion are done with the help of *MARG (Magnetic, Angular Rate, and Gravity)* sensors and micro controllers. Merge and store is done using a local on board computer. Deferred collection, cleanup and post processing is done using any preferred computer after the reading process is completed. Finally data can be utilized as intended. Post processing is also partially dependent on the final utilization use cases. For the simplicity of explanation and understanding, we divide our approach into the following steps.

- Orientation Measurement Using SOUs and Madgwick Filter (Orientation, Reading, Fusion)
- Data Collection and Integration using On Board Computer (Merge and Store)
- Data Cleaning (Deferred Collection, Cleanup)
- Absolute to Relative Conversion (Post Processing)
- Cartesian Conversion and Smoothing (Post Processing)
- Data Mapping and Export (Utilization)

- Euler Discontinuity Filtering (Post Processing, Utilization)
- Visualization (Utilization)

In our case, we utilized the readings by mapping the data to a 3D rigged model in Blender. So, some parts of the post processing are done accordingly.

6.1 Orientation measurement using SOU and Madgwick filter

Among different approaches in measuring the orientation of a rigid body, IMU based sensor array is popular for measuring orientation where sensor size, mobility are of concern and the working environment is dynamic. An Inertial Measurement Unit (IMU) includes an accelerometer and gyroscope allowing the measurement of rotational and transitional movements. However, in our approach we will be using MPU 9250, a MARG sensor. A MARG (Magnetic, Angular Rate and Gravity) sensor is a combination of IMU sensor and a magnetometer sensor. The MPU 9250 incorporates a tri-axis accelerometer, gyroscope and magnetometer sensor whose axes are mutually orthogonal to each other as portrayed in figure 4a and 4b.

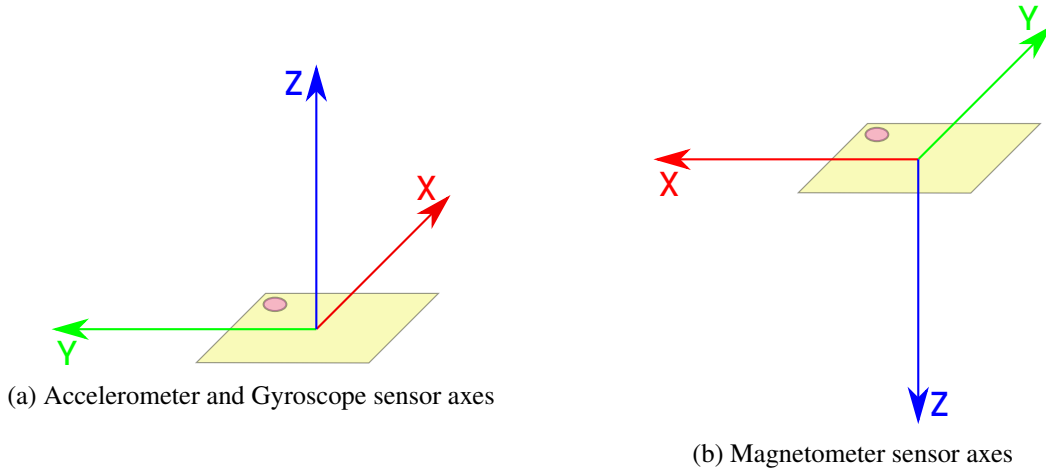


Figure 4: Axes orientation of sensors in MPU 9250

The complete sensor package, also known as AHRS (Attitude and Heading Reference Systems) provides a complete solution to the measurement of orientation relative to the direction of gravity and the earth's magnetic field. For our implementation, we have packed our orientation measurement system into a single unit known as the **Singular Orientation Unit (SOU)** containing two MPU 9250 along with an Arduino Nano to drive the sensors. The sensors in MPU 9250 have their own properties that play a crucial role in fusing them together at a later stage to get accurate orientation measurement. The orientation of an object is expressed in terms of yaw, pitch and roll, which are relative to a predefined

earth's frame, which will be explained in details later.

The gyroscope sensor provides rate of change of angular velocity along the three axes. To calculate the orientation along each axis, the velocity is integrated over time using Newtonian physics to give its relative position, which in this case, is an angle. The gyroscope sensor has some inherent constraints and shortcomings associated with it.

- The angle calculated is only relative to itself and not relative to earth's frame.
- The gyroscope readings drift over time due to temperature and motion which produces erroneous readings. So, gyroscope sensor can only provide accurate readings for a short period of time.
- Suffers from Gimbal lock when two of the axes align with each other.

To derive the reference earth's frame, the accelerometer and magnetometer sensors are used, as they measure the earth's field namely gravity and magnetic flux. The accelerometer measures acceleration along the three axes. The accelerometer is used to get the orientation with respect to the earth's frame using inverse tangent. Though we can derive the orientation relative to earth's frame, we cannot measure the orientation when any one of the axes of the accelerometer is in parallel to earth's gravitational field, because tangent is undefined at 90 degrees in that direction. The problem with accelerometer is that the readings are noisy and cannot detect fine movement such as small vibrations from an object, unlike gyroscope sensor.

The magnetometer measures the earth's magnetic field which is also used to derive the earth's frame of reference. Like accelerometer, magnetometer is unable to measure orientation when the earth's magnetic field is in parallel with any of the axes. The magnetometer along with accelerometer, is used to compensate gyroscope sensor bias drift [30].

Overall it can be seen that none of the sensors provide effective solution to calculate the orientation of an object accurately. What one sensor lacks is covered by another sensor. So, it would be highly effective in providing accurate orientation estimation, if the three sensors readings could be fused together to give a unified orientation measurement.

When working with any mechanical sensor, such as MPU 9250, it is mandatory to calibrate it. Calibration is necessary for the correct configuration of sensor parameters and registers. Calibration of accelerometer and gyroscope is done simply by placing them on a horizontal flat surface and calculating

the offsets for each axis, from a reference reading of $[0 \ 0 \ 1]$ for accelerometer whose unit of measure is g and $[0 \ 0 \ 0]$ for gyroscope whose unit of measure is radians/second. These offsets are then subtracted from the sensor readings to get calibrated sensor values. The calibration of a magnetometer, on the other hand, is different and a bit more complex. Magnetic fields are naturally present as earth's magnetic fields, which we want to measure. Alongside the earth's magnetic field that affects the magnetometer, artificial magnetic flux sources around the sensor body are also measured by the sensor. These sources are in some cases stronger in magnitude, thus shadows the magnitude of the earth's magnetic field. These sources can either be hard or soft iron biases. Hard iron biases are magnetic fields created by permanent magnetic sources, and in most cases, the removal of hard iron biases provides readings with sufficient accuracy. However, where precision is required, soft iron bias must also be accounted for and removed from the readings. Soft iron biases are offset magnetic fields created by soft iron materials such as wires or circuitry.

6.1.1 Madgwick Filter

The Madgwick filter [5] is a sensor fusion algorithm that combines multiple measurements from different sensors and unifies them to return a combined result. The advantage of sensor fusion for our implementation is that, it combines the suitable characteristics of different sensors together which in turn, negates the shortcomings of each of the sensors as mentioned in the previous section. For any filter, it is required to transform the sensor's inertial frame to some common frame, ideally the earth's frame. The filter achieves this by generating a rotational matrix. Before diving into the mechanics of the filter and its matrix generation, to calculate the absolute orientation of an object, we have to look at some complementary concepts associated with the Madgwick filter.

Quaternions: A quaternion is a four-dimensional number system that can be used to represent the orientation of a rigid body or coordinate frame in three-dimensional space [5]. A quaternion is an extension of complex number containing four numbers, where three of them are imaginary components and one real component. A quaternion can be represented as a vector as shown in equation (1).

$$q = [s \ xi \ yj \ zk] \quad (1)$$

In equation (1), s is the real component and the other three are imaginary components of a pure quaternion in a vector. Quaternion is extremely suitable for representing rotation from an initial frame to a destination frame, which is the main requirement for the approach. In quaternion space, an arbitrary orientation of frame B relative to frame A can be achieved by a rotation of angle θ around an axis ${}^A\hat{r}$

defined by frame A in diagram 5.

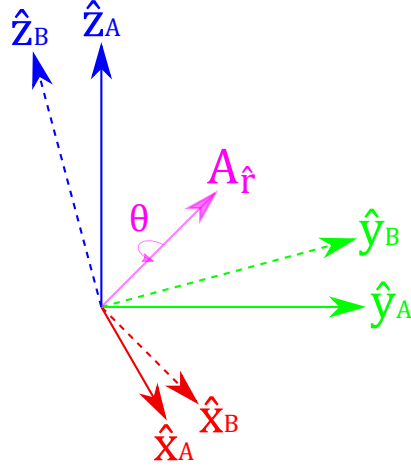


Figure 5: Representation of rotation in Quaternion space

In diagram 5, the mutually orthogonal unit vectors \hat{x}_A , \hat{y}_A and \hat{z}_A , and \hat{x}_B , \hat{y}_B and \hat{z}_B define the principle axes of coordinate frames A and B respectively. The notation system for representing the quaternion is adopted from Craig [31]. The leading super-script and sub-script are used to represent the relative frames of orientation and vector. Another application of quaternion number system is to overcome the Gimbal lock effect [32]. Gimbal lock is present in commonly used Euler representation of orientation of an object, which is removed when the same orientation is represented, in four dimensional number system using quaternion.

Like other number systems, quaternion also has conjugate and quaternion product that defines the compound orientation. Suppose for two given quaternions described by ${}^A_B\hat{q}$ and ${}^B_C\hat{q}$, the compound orientation ${}^A_C\hat{q}$ is defined by equation 2.

$${}^A_C\hat{q} = {}^B_C\hat{q} \otimes {}^A_B\hat{q} \quad (2)$$

The quaternion described by ${}^A_B\hat{q}$ can also be represented as rotational matrix A_BR [5] which is used in mapping the sensor inertial frame to the earth's frame.

Sensor fusion using Madgwick filter: The complete theory [5] and the mathematics behind the concept of Madgwick filter and how it combines the three sensor values to generate the unified orientation quaternion are beyond the scope of this report. However, some details are provided on how the different components work on each sample of sensor data in unifying the sensor values.

The raw calibrated gyro readings are converted to a vector representation as in equation (3).

$$S_\omega = [0 \ \omega_x \ \omega_y \ \omega_z] \quad (3)$$

The gyroscope measures the angular rate along the x, y and z axes. To find the quaternion derivative describing the rate of change of orientation of the earth's frame relative to the sensor frame [33], denoted by ${}^S_E \dot{q}$, the vector in equation (3) is left multiplied as shown in equation (4).

$${}^S_E \dot{q} = \frac{1}{2} {}^S_E \hat{q} \otimes S_\omega \quad (4)$$

To calculate the orientation of the earth's frame relative to the sensor frame at time t , denoted by ${}^S_E q_{\omega,t}$, the quaternion derivative, ${}^S_E \dot{q}_{\omega,t}$ is numerically integrated as shown in equation (5) and (6).

$${}^S_E \dot{q}_{\omega,t} = \frac{1}{2} {}^S_E \hat{q}_{est,t-1} \otimes S_{\omega_t} \quad (5)$$

$${}^S_E q_{\omega,t} = {}^S_E \hat{q}_{est,t-1} + {}^S_E \dot{q}_{\omega,t} \Delta t \quad (6)$$

In equations (5) and (6), S_{ω_t} is the angular rate measured at time t , Δt is the sampling period and ${}^S_E \hat{q}_{est,t-1}$ is the previous quaternion estimation. The quaternion ${}^S_E \hat{q}$ is the required rotational matrix representation that maps the sensor inertial frame to the earth's frame which is the required objective of Madgwick filter. In order to minimize the error from the measured sensor values, Madgwick filter updates this rotational matrix representation on each sample of data in order to align the sensor's inertial frame and the earth's frame.

As mentioned earlier, the tri-axis accelerometer gives the magnitude and the field of gravity combined with the linear acceleration due to motion. The tri-axis magnetometer will measure the magnitude and direction of the earth's magnetic field in the sensor's frame compounded with local magnetic flux and distortions. Madgwick filter assumes that initially the accelerometer and magnetometer will only measure the earth's gravitational and magnetic field.

Using the field information, a measurement of the field's direction within the sensor frame will give an orientation of the sensor frame relative to the earth's frame. The derived orientation will have infinite solution rather than a unique solution [5]. Since the filter uses quaternion to represent orientation, it

requires a complete solution. The solution is the rotational quaternion representation which is formulated as an optimization problem where an orientation of the sensor, denoted by ${}^S_E\hat{q}$, is that which aligns a predefined reference direction of the field in the earth's frame, ${}^E\hat{d}$ with the measured direction of the field in the sensor frame, ${}^S\hat{s}$. The required quaternion representation ${}^S_E\hat{q}$ may be found as the solution to equation (7), where the objective function is formulated as equation (8).

$${}^S_E\hat{q} = \min_{{}^S_E\hat{q} \in \mathbb{R}^4} f({}^S_E\hat{q}, {}^E\hat{d}, {}^S\hat{s}) \quad (7)$$

$$f({}^S_E\hat{q}, {}^E\hat{d}, {}^S\hat{s}) = {}^S_E\hat{q}^* \otimes {}^E\hat{d} \otimes {}^S_E\hat{q} - {}^S\hat{s} \quad (8)$$

$${}^S_E\hat{q} = [q1 \ q2 \ q3 \ q4] \quad (9)$$

$${}^E\hat{d} = [0 \ d_x \ d_y \ d_z] \quad (10)$$

$${}^S\hat{s} = [0 \ s_x \ s_y \ s_z] \quad (11)$$

Now to find the solution, which is a minimization problem, Madgwick filter uses gradient descent algorithm to move to the solution. The solution surface is defined by the objective function and the Jacobian [5]. The Jacobian matrix, holds the partial derivatives required by the gradient descent algorithm. When multiplied by the objective function to be optimized, the gradient is calculated. This gradient is then used to update the quaternion that rotates the predefined reference frame to minimize the error from the measured sensor values.

To summarise the overall view of how Madgwick filter works on each sample of data: Madgwick filter reduces error of the measured sensor values by transforming the sensor's inertial frame to the earth's frame. This is done using information from earth's fields and creating rotational matrix in the form of quaternion. The filter calculates the orientation from gyroscope readings by numerical integration. To map this transformation to the earth's frame, the gyroscope orientation is left multiplied with a quaternion. This rotational quaternion is generated using earth's field sensors which are the accelerometer and magnetometer. The matrix is updated and optimised using gradient descent algorithm. Madgwick filter requires that each sensor has the complete tri-axis orientation, and assumes a predefined reference of the field in the earth's frame. Optimising the rotational matrix will allow to rotate the predefined reference frame to minimize the difference between the measured and actual sensor values. The filter optimises for all three axes for both sensors using one step per sample. The algorithm also uses Jacobian matrix containing partial derivatives which is multiplied by the objective function to be optimized to calculate the gradient. The gradient is then used to update the rotational matrix.

Output from Madgwick filter is a quaternion that represents the absolute orientation estimation. This quaternion is converted to Euler representation which is then used to visualize the human posture. The output format of an SOU is shown in table 2.

SOU ID (N0, N1,, Nn)	Yaw1	Pitch1	Roll1	Yaw2	Pitch2	Roll2
------------------------------	------	--------	-------	------	--------	-------

Table 2: Output Format of a Single SOU

6.2 Data Collection and Integration using On Board Computer

A driver running in the on board computer serves as the hub of integration for the data coming from different SOUs. Having a small on board computer allows us to perform the integration of data right when they are produced while allowing complete mobility to the user.

The driver serves two main purposes. Firstly, it serves as the source of initialization for all the SOUs through the micro-controllers controlling them. Secondly, it collects individual orientation data from each of the SOUs in parallel, time stamps them and stores them in a file for later use.

There are two main approaches that can be taken to build the driver for data collection.

1. Polling based approach
2. Asynchronous Event Driven approach

Polling based approach relies on the driver querying each of the SOUs to get a single reading based on the predefined format. It has some benefits, for instance, it can synchronise the clock and ask for data after a certain interval from each of the SOUs. However, it depends on continuous processing and if due to some error, a SOU temporarily fails to provide data, there can be a blocking condition. Also this method fails to capitalize the parallel processing power of modern computers properly.

Asynchronous Event Driven approach relies on an event system to collect, time stamp and store the data from the SOUs independent of each other. Whenever an SOU is ready to provide a reading, it fires an event via serial port and the driver then collects the data, time stamps them based on its own clock and stores it. Since storing is done in a shared resource, only the storing part has blocking condition, which is resolved with the help of mutex lock. Other than the storing, all other tasks can be multi-threaded and processed in parallel. This method also allows a simple scalable architecture to increment the number of SOUs without doing any modification to the code base, since each of the SOU is handled independently.

For the same reason, this method also supports SOUs processing at different sampling rates which might be suitable in some use cases.

6.3 Data Cleaning

After data collection in the on board computer is completed, it can later be transferred to any suitable processing medium, such as a personal computer for further utilization. It is often the case that the initial readings are either corrupted, wrong, noise induced or have some special debug texts collected due to the choice of transfer medium, such as via USB. As a result, some part of the corrupted data needs to be cleaved. Both *fixed length* or *dynamic cleaning* can be used. For *fixed length cleaning*, some part of the readings are removed, generally from the beginning as only the initial texts are corrupted or contain debug texts. For *dynamic cleaning*, structure or format for each of the reading is mentioned and matched with the readings. If any reading violates the predefined format, that entry is removed. This can be done using regular expressions, state machines or any other string matching technique.

Starting time stamp for data entry is also changed when first few data are removed. So, minimum value of times stamps needs to be subtracted from each time stamp to make them start counting from 0.

$$\overrightarrow{Time\ Stamp} = \overrightarrow{Time\ Stamp} - \min(\overrightarrow{Time\ Stamp})$$

6.4 Absolute to Relative Conversion

As mentioned in the *Mapping* section, the SOUs provide us absolute orientation of limbs. According to our design, these orientations are in *3D Euler space*. In order to utilize these data, we want to map it to a 3D human model in Blender. However, for a human rig, the data required needs to be in relative orientation of each joint, as shown in figure 2. And as already mentioned before, we can achieve these relative rotations of each node by subtracting its parent node's absolute rotation from its absolute rotation. An example is shown in figure 6, where the node attached with the square is the parent and the other one is the child. If both nodes rotate downward by 15° as shown in state 2, we would get absolute reading of 15° for the parent and 30° for the child. We can convert the child rotation to its relative value by subtracting its parent's 15° from its 30° rotation.

The parent child relations that we have used are shown in table 1. Only 'NO_0' is the anchor node and its absolute reading is also its relative reading.

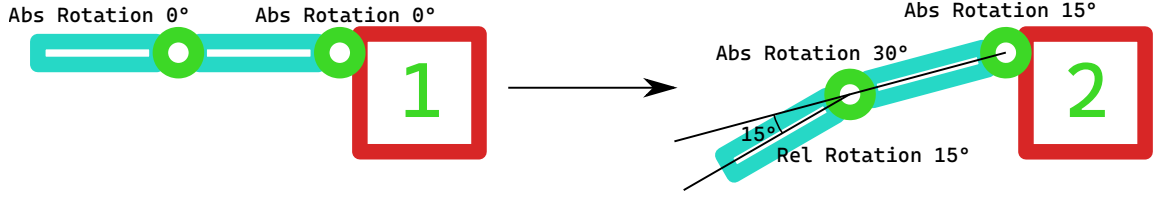


Figure 6: Absolute to Relative Conversion

6.5 Cartesian Conversion

In our approach, we considered Euler angular values to denote orientation of limbs. Due to the nature of the gathered data, it is mandatory to perform smoothing. A crucial part of smoothing is weighted averaging operation where the raw values need to be averaged according to the smoothing algorithm. But this poses a problem in our data.

Let us consider an example. If two consecutive entries of our data are 360° and 0° , practically they are the same data point, so the average of these two points should be 360° or 0° . But the conventional approach gives us $\frac{360^\circ + 0^\circ}{2} = 180^\circ$, which is wrong.

To deal with this problem, we consider our raw data input as polar coordinates rather than simple 1D values. In this approach, we consider angular values as angles and took an arbitrary value 1 as the radius. So our data points, a and b would be converted to $360^\circ \rightarrow (1, 360^\circ)$ and $0^\circ \rightarrow (1, 0^\circ)$. Now, if we convert the polar coordinates to the Cartesian coordinates, we get,

$$x_a = 1 \times \cos(360^\circ) = 1, y_a = 1 \times \sin(360^\circ) = 0$$

$$x_b = 1 \times \cos(0^\circ) = 1, y_b = 1 \times \sin(0^\circ) = 0$$

Averaging the Cartesian coordinates from these two points, we get,

$$x_{avg} = 1, y_{avg} = 0$$

Now we revert back to the polar coordinates,

$$avgAngle = \tan^{-1}\left(\frac{y_{avg}}{x_{avg}}\right) = 0$$

Following this transformation, we get an average value of 0° , which is our expected outcome. We can see that from the Cartesian transformation, we get two values for each point, x component and y component. For averaging along consecutive data points, we have averaged along the x component values and y component values.

In our application, we have angular outputs along 3 axes. Each of these output points has 6 values altogether after transformation, and these values of consecutive points are averaged along the 6 separate components.

6.6 Smoothing

Smoothing is one of the most important parts of our data processing pipeline. Our data have considerable amount of noise. Reasons being,

- There are 6 SOUs feeding data, each of which has their own noise and the noise accumulates.
- We are working with skeletal orientation, but we can only get it through muscular orientation. Muscles are soft bodies which generate noises due to vibration with each movement.

To get rid of all the noises, according to the use case and available data, a number of smoothing approaches can be followed. Generally for smoothing data, most high end systems rely on modified versions of Extended Kalman Filter. But this filter needs raw data to work correctly and is intended towards displacement calculation. Also these filters are computationally expensive. Our task is orientation calculation and we would prefer an inexpensive approach. The possible solutions can be categorized into four classes.

6.6.1 Convolutional Filters

Convolutional filters use a fixed kernel and convolutes it with the original signal. Depending on the kernel, the smoothing differs. The most common and one of the simplest kernels is the Gaussian kernel, which is based on Gaussian distribution. We have used a modified version in our work. A Gaussian kernel takes two parameters, filter length and the standard deviation of the kernel.

The pro of this approach is that it is fast. Using FFT (Fast Fourier Transform), it can perform smoothing easily and does not require any complex hardware. Also the kernel is very basic and easy to

understand. So anyone with given parameters can implement the filter with minimal coding.

There are some drawbacks to these filters. They are global filters, which means they work on the whole signal. Now, a signal can behave differently at different points in time. There is no pattern or repetition (unless voluntarily done). So the smoothing parameters should not be same at all the timestamps. But it is not possible in these filters. Consequently, the filtering would not be similarly effective at all places. Again, for the same reason, selecting a global set of parameters is tough. Even if one is selected, it is not guaranteed to work well. Therefore, selecting different parameters for different types of signals is also not possible. So, we have to select one set of parameters and expect it to work throughout the data collection procedure's lifetime, which is highly unlikely to be possible.

6.6.2 Adaptive Filters

Adaptive filters are similar to convolutional filters, which try to address the problems of convolutional filters. Adaptive filters are basically a set of locally implemented convolutional filters. The problem for convolutional filters are that they are global. Adaptive approach takes the filters and implements them locally.

Adaptive filters can perform better filtering for different types of signals. The parameters have more flexibility to be controlled in different places of the signal, not just one set of parameters that have to work with the whole signal. This can give sufficiently well results if the parameters are properly set.

For the cons, as stated above, it is very tricky to implement. The local statistics are needed for each of the filters, even then it still might not be enough. The hyper parameters would be a variable for different parts of signals. All these need to be controlled manually which is a big downside for a device that should have automatic handling of smoothing. Our ideal filter should have minimal manual intervention.

6.6.3 Frequency Domain Filters

Frequency domain filters are an alternate representation of the convolutional filters. Converting a signal to frequency domain exposes a unique set of parameters of the original signal. It can distinguish between noise and original signal, and can control the amount of smoothing easily. The noises can be distinguished, isolated and discarded. For easier representations, masks can be constructed based on the transformation steps and multiplied with the original frequency domain signal to get the desired

smoothed signal.

There are several pros of this approach. It is very fast and allows arbitrary cut offs and manipulations to control each of the frequency signals in frequency domain, so that we get considerable flexibility. It is also easy to build and implement. The filters used are low pass or semi low pass filters.

Like convolutional filters, these are also global filters. This again gives the problem of not being compatible throughout the whole signal. Transforming it into a time varied adaptive filter is possible but very tough. The cut-off and suppressing parameters are set on a trial and error basis, which is time consuming.

6.6.4 Predictive Filters

Predictive filters are the class of filters that smooths the signal based on prediction. Kalman filters are Bayesian filters which use probability to predict signal values at given point in time. Furthermore, all the filters mentioned in the previous subsections can be a predictive filter if we apply machine learning to it. The adaptive and local frequency domain filters have considerable number of parameters to be optimized. Whether the implementation is local or global, setting these parameters manually for each iteration and each type of signal is time consuming and complex. If we want to automate the parameter setting, we have to utilize previous signal data and predict these parameters based on the input signal. Thus we can automate the manual procedure of optimization.

The predictive filter classes can be very accurate. Its biggest upside is that it does not need any manual intervention. But the main downside is that it requires large amount of data to work properly. Large amount of data requires considerable amount of analysis, which is computationally expensive to implement. Setting up the proper learning methods and hyper-parameters in predictive learning is also a novel task and research area by itself.

Since our use case is general, we used a hand crafted Gaussian kernel for our work since it was sufficient enough for it. However, it might not be suitable for other use cases and different filters should be chosen accordingly based on the application scenario.

6.7 Data Mapping and Export

After the data have been cleaned and smoothed, it is now ready to be used for application, in our case, visualization in Blender. However, Blender or any other animation or 3D modeling software or graphics packages have their own representation of orientation. The data we receive from our system contains the rotation in 3D Euler space in *Yaw*, *Pitch* and *Roll* format. Blender accepts orientation data for joints in one of the various formats, such as Quaternion, XYZ Euler, XZY Euler, Axis Angle and other Euler combinations as shown in figure 7. Orientation for joints in Blender is relative, the conversion that we have already done. We need to map from our *Yaw*, *Pitch* and *Roll* orientation to the target orientation of Blender with appropriate conversion logic. Moreover, each node of our system maps to a certain joint in the 3D skeleton rig in Blender. All these conversions and mapping can be automated by mentioning the mappings and orientation conversions in a narrative file, such as *JSON* or *YAML*, or in any markup language such *XML* and then using a program to parse these mappings and perform the conversion.

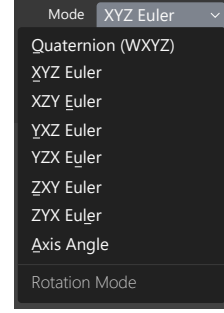


Figure 7: Blender Rotation Modes

After the mapping is done, data need to be actually exported and connected to the Blender rig. Blender offers creation of modules or scripting via *Python* to achieve such automation tasks. Moreover, Blender being an open source program, the source code can be modified and recompiled if extensive change is needed. Along with the mapping, time stamps of each data need to be converted to target frame animation for Blender via scripting. Blender supports fractional values as frame number. For instance, if the time stamps are calculated in millisecond unit, then equation 12 and 13 can be used to convert time stamps to target frame number for Blender.

$$fn = \frac{t}{\delta t} \quad (12)$$

$$and, \delta t = \frac{1000}{r} \quad (13)$$

where,

fn = Frame Number

t = Time Stamp of Entry

δt = Time per Frame

r = Frame Rate per Second

6.8 Euler Discontinuity Filtering

As mentioned before, we are using 3D Euler angles for rotation calculation following NED (North, East, Down) convention, where *Yaw* spans from 0° to 360° , *Pitch* spans from -90° to 90° and *Roll* spans from -180° to 180° . Values outside these ranges are still possible when we are performing the subtraction for absolute to relative conversion. Now, in Euler angles, there is discontinuity. For instance, for 0° to 360° rotations, if increment is 1, the value after 359° degree is 0° again, and so the space is discontinuous here. Similarly, in XYZ Euler convention, $(0^\circ, 0^\circ, 0^\circ)$ and $(180^\circ, 180^\circ, 180^\circ)$ rotations express the same orientation as shown for the two springs in figure 8. However, if these two values come right after one another, most software will try to interpolate between these two values, which will result in a complete rotation, which is wrong and should not be done. This problem is known as Euler discontinuity.

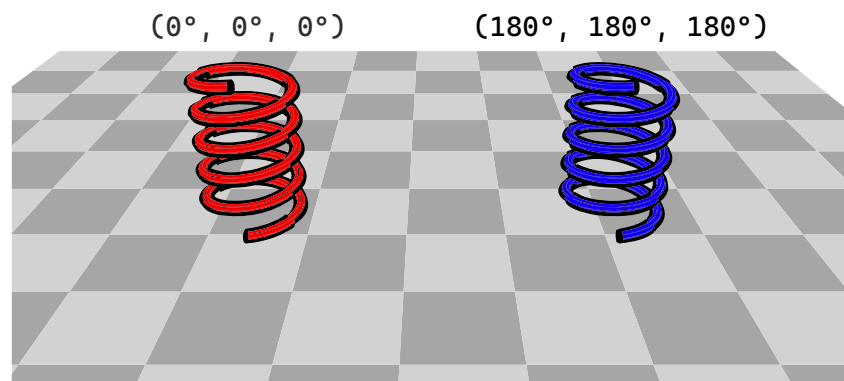


Figure 8: Euler Discontinuity and Same Orientation

Euler discontinuity is solved using Euler Discontinuity Filter, which from all possible rotation values for a given orientation, picks the one that minimizes overall rotation for discontinuous values. Almost every graphics package comes with its own Euler Discontinuity Filter. Blender also has its own and it is also called '*Euler Discontinuity Filter*'. It might be named differently in different packages. Alternatively, if the graphics package supports, direct Quaternion values can be used which do not have discontinuity, or discontinuity can be eliminated using external program. However, if the target is some specific program or graphics package and that program or package provides its own discontinuity filter, it is generally best to use that directly.

6.9 Visualization

After Euler discontinuity has been eliminated from the orientation data, it is ready to be used now. Since we are using Blender, we can either try to visualize it directly inside Blender, render out the animation after setting up camera using any one of the available renders or alternatively, bake the animations within an *FBX* file and use it in game development or any other purpose. For our application, we did simple

visualization and render using Blender rendering and animation pipeline.

7 Experimentation

7.1 IMUs (MPU 9250) and Arduinos (SOUs)

As mentioned earlier, an SOU contains two MPU 9250 and Arduino Nano bundled together. The SOU is responsible for calculating the orientation. The connection scheme inside the SOU is shown in diagram 9.

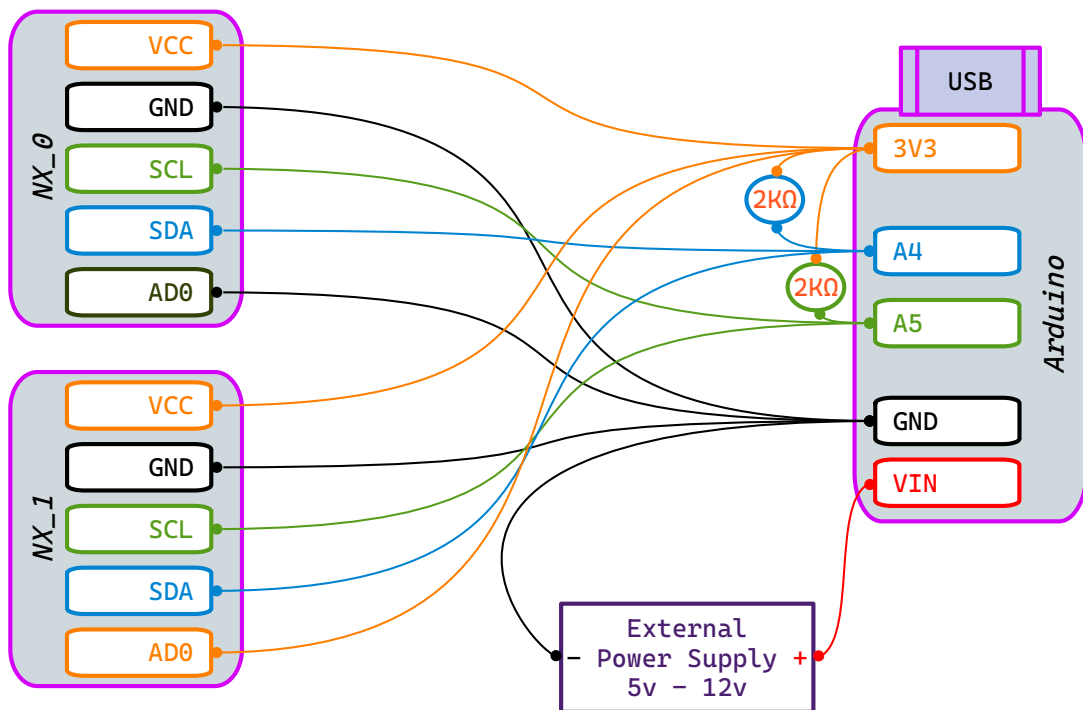


Figure 9: Connection scheme inside SOU

According to diagram 9, the connections between Arduino Nano and the MARG sensors are given in table 3:

Arduino Pinout	MPU 9250 Pinout
SCL	A5
SDA	A4
3V3	VCC
GND	GND
AD0	3V3/GND

Table 3: Arduino Nano and MPU 9250 pin mapping

Few points to be noted about the setup and more specifically the naming convention that has been followed in the approach are given next.

- The AD0 pin of MPU 9250 is either connected to 3V3 or GND pin of Arduino Nano. If one is connected to GND, the other one must be connected to 3V3. This ensures that each MPU 9250 has a unique address, used during communication between MPU 9250 and Arduino Nano.
- MPU 9250, also termed as node in the report, are named according to how AD0 pin is configured. If AD0 is connected to GND, then the node is named as NX_0 where N stands for node, X can be any number representing the node ID. If AD0 is connected to 3V3, then it is named as NX_1. The naming of nodes in this specific order is crucial to how the nodes are placed on a limb for estimating the orientation.
- The communication protocol used is I2C, which is a serial communication protocol.
- Data is transmitted between the MPU 9250 and Arduino Nano through the SDA line.
- SCL line is used for synchronising the timings and clock information between the MPU 9250 and Arduino Nano.
- The 2 kilo ohm resistors, added in parallel between 3V3 and SCL and SDA lines are used as pull up resistors to ensure that the SCL and SDA lines can pull up to high state.
- The computer or the data collection device communicates with the Arduinos through the USB serial interface.

7.2 Raspberry Pi and Driver

We have chosen Raspberry Pi 4 (4GB variant) as our on board computer for data collection and integration. The Raspberry Pi is attached with the user whose posture data is being collected and powered by an external 5V (stepped down from 8.4V) lithium polymer battery. The Arduino of each SOU is connected to the Raspberry Pi via a USB cable. A USB hub is used to accommodate all 6 SOUs. Both USB 2.0 and USB 3.0 or anything better is sufficient enough to handle all the incoming data.

The driver is written in *.NET 5.0*. The main reasons behind choosing *.NET 5.0* are the facts that it is event driven and both cross-platform and cross-architecture. It means it allows us to write a driver using a single code base that would seamlessly run on Windows, Linux or other supported operating systems while being supported on both *x86* and *ARM* architectures. It gives us the flexibility to change our intermediate on board computer or upgrade it. And if anyone wishes, data can be directly collected

to a PC via USB, omitting the middle on board computer. Moreover, *.NET 5.0* gives us the option to package the driver code base along with necessary run time into an executable and transfer it to the on board computer. It removes the dependency of having *.NET* run time installed in the on board computer.

.NET being an event driven environment, allows us to perform easy parallel processing of each of the individual SOU inputs. The driver first checks all the available COM serial ports, establishes connections with those and binds event on input receive. Since each of the SOU has its own identity in the data it writes, the driver just receives the data, puts a time stamps along with it and then tries to write it to a file which is guarded by a mutex lock to prevent erroneous concurrent writing. Every “time stamp” is appended with the data before accessing the data file and hence time stamping is a parallel process and not dependent on any other SOU data. The data format after being time stamped is shown in table 4.

Time Stamp	SOU Identity	Yaw1	Pitch1	Roll1	Yaw2	Pitch2	Roll2
------------	--------------	------	--------	-------	------	--------	-------

Table 4: Data Format after Time Stamping

Some added benefits of choosing *.NET* framework are that *.NET* is a matured framework with sufficient support materials and *ASP.NET* can be used to convert the driver for web based real time application.

Using the built in normal timer of Raspberry Pi, the *.NET* driver is capable of measuring time stamps at a minimum of ~ 17 milliseconds apart, per parallel calculation. Each SOU of our system produces around 26 data per second, which would result in 2 consecutive data being ~ 38 milliseconds apart, which the driver can handle easily.

We have tested our driver running on Raspberry Pi with 12 mock SOUs, double the amount we have used in our actual experiment and the driver was still able to properly handle all the data writing and time stamping tasks. 12 SOUs should be sufficient enough for most measurement purposes. However, if any task requires more SOUs than this and Raspberry Pi 4 or USB cannot handle it, one can always opt for more powerful on board computer alternatives and/or data transfer media.

While collecting the data, the driver can be initialized on the Pi easily using a remote computer control system, such as *Remote Desktop* or *SSH*. Alternatively control can be given to the user who is providing orientation data with the help of Raspberry Pi’s *GPIO* pins or other input media.

7.3 Pose Calibration and Reading

To ensure that correct orientation estimation is calculated, it is mandatory that the sensors are calibrated as mentioned earlier. For magnetometer calibration, we used *MagMaster*, a software that calculates the hard and soft iron biases for us. The software also provides information about how to take readings from the sensor and integrate this bias information in the Arduino code to get calibrated magnetometer readings. The calibration is done beforehand.

For the calibration of accelerometer and gyroscope in each MPU 9250 in real time, an initial stationary pose is followed, known as T pose, that places all sensors in the mandatory flat and stationary position. T pose is also used, so that the real person and the 3D model starts from the same reference pose. For a complete human posture measurement, the node configuration should be according to diagram 2.

Assuming that the nodes are numbered correctly and placed flat on the limbs, the person providing the posture needs to mimic the T pose and be as stationary as possible. The calibration process is initiated when the driver program is executed, which in turn executes the Arduino program and begins the sensor calibration procedure. The sensor calibration for each individual SOU starts in parallel and roughly requires 10-20 seconds for the calibration to be completed. After the calibration procedure is completed, the user can freely move at will and provide posture information which will be recorded by our system.

7.4 Data Post Processing

We have chosen Python as our main post processing language since it has a vast collection of libraries for data processing and similar activities and real time operation is not our concern here. Once the data are collected using the on board computer, it can be post processed in any suitable computer, in our case we have used x86_64 Windows PC.

7.4.1 Data Cleaning

The first few data written by the driver are generally corrupted, due to the calibration of the sensors and initialization of parallel processing in the driver. Along with that, Linux in ARM architecture produces some garbage debug code during the initialization of serial ports. These corrupted entries need to be removed. We have used a combination of *static (fixed length)* and *dynamic cleaning*, where we used a regular expression for the format mentioned in table 4 to detect the corrupted entries. Then we remove all the data until the last entry that is corrupted. The regular expression that we have used is,

`^[0-9]+,[a-zA-Z0-9]+(,[\\\-0-9\\.]+){6}$`. The regular expression needs to be modified based on the formatting of data, in case any other formatting is used.

7.4.2 Data Summarization

After data have been cleaned, a data summarizer written in Python is used to summarize overall data collection task and generate performance summary for each SOU or node. The summarizer reports the overall time frame in seconds during which data have been collected and total number of SOUs or nodes that contributed to the data. Then for each node, overall data count, mean time difference between two consecutive data of the SOU, along with minimum and maximum time difference is provided. If the system is functioning properly, data count for all the SOUs will be very close in number, and their mean time difference should also be close, which in our case, as stated before, should be around 38 milliseconds. The summarizer works as a quick debugging step to check whether the whole system has worked properly or not. An example summary of a 3 SOU system is given next.

```
Time Frame: 90.075 seconds
Node Count: 3
Node: N0|Data Count: 2336|MeanDelT: 38.559315|MinDelT: 7.00 |MaxDelT: 50.00
Node: N1|Data Count: 2350|MeanDelT: 38.343976|MinDelT: 29.00|MaxDelT: 47.00
Node: N2|Data Count: 2353|MeanDelT: 38.213861|MinDelT: 34.00|MaxDelT: 42.00
```

7.4.3 Absolute to Relative Conversion

As mentioned before, the SOUs provide us absolute orientation of the limbs, and to use for our visualization purpose, we require relative orientation of the data, which can be found by subtracting the parents' absolute orientation from children's absolute orientation, according to the relationship provided in table 1.

Based on how we have designed the placement of our SOUs, there are two stages in which we can convert the absolute data to relative. First of all, each SOU provides us orientation from two nodes or MARG sensors. Now, the two nodes or MARG sensors in a single SOU are inherently in parent child relationship. For instance, in the nodes 'N1_0' and 'N1_1', 'N1_1' is the child of 'N1_0'. And in the data, we would always receive each of the entries of 'N1' SOU with a single time stamp, where *Yaw1*, *Pitch1* and *Roll1* denotes the orientation of 'N1_0' and *Yaw2*, *Pitch2* and *Roll2* denotes the orientation of 'N1_1', according to the format presented in table 4. In these cases, after selecting a view with only

‘N1’ SOU, we can perform vector subtractions as shown in equation 14, 15 and 16.

$$\overrightarrow{Yaw2} = \overrightarrow{Yaw2} - \overrightarrow{Yaw1} \quad (14)$$

$$\overrightarrow{Pitch2} = \overrightarrow{Pitch2} - \overrightarrow{Pitch1} \quad (15)$$

$$\overrightarrow{Roll2} = \overrightarrow{Roll2} - \overrightarrow{Roll1} \quad (16)$$

However, for the relations where the parent and child do not belong to the same SOU, the process is a bit complicated than the previous one and a sort and linear scan is required to perform those updates. For instance, where ‘N0_1’ is the parent of ‘N1_0’. It can be done using the pseudo code given in listing 1. ‘//’ indicates comment in the listings.

```

1 let, df = dictionary{data}
2 let, lastVal = dictionary{}
3 sort df based on df[time stamps]
4 for each entry in df:
5     if entry[node1] is a parent:
6         last value[entry[node1].id] = entry[node1].value
7     if entry[node2] is a parent:
8         last value[entry[node2].id] = entry[node2].value
9     if entry[node2] is a child:
10        // Only 2nd node of a SOU can be child
11        entry[node2].yaw -= lastVal[entry[node2].parent].yaw
12        entry[node2].pitch -= lastVal[entry[node2].parent].pitch
13        entry[node2].roll -= lastVal[entry[node2].parent].roll

```

Listing 1: Non SOU Absolute to Relative Conversion

These two approaches are merged together to get the final absolute to relative converted data.

7.4.4 Cartesian Conversion and Smoothing

According to *Approach* section, subsection 6.5, we considered our linear values as polar coordinates, where the radius is 1 and the different linear values are the angles. Then following the Cartesian conversion, we get the Cartesian coordinates of the angle values.

For our work, we have 3 axes values for each data point, X, Y and Z. For each of the 3 axes, we have 2 coordinate values after conversion, adding up to 6 values in total. Our whole dataset would provide a stream of data points which would have 6 values at each point.

For smoothing operation, we isolate each of the X, Y and Z axes. Then we get the Cartesian coordinate values of each axes, smooth out the time sorted two coordinate values throughout the whole signal and then revert back to the axes values from the coordinate. We perform it for all the axes.

The smoothing approach depends on the use case and flexibility. In our work, we implemented multiple filters built focusing specifically towards our use case, which is modeling the movement of the body. We have used Gaussian filters and frequency domain filters. We did not use Kalman or any other predictive filters due to lack of required amount of data.

Our first approach was using the Gaussian filter. This filter is a global filter, which needed a good amount of tweaking of parameters to make it work.

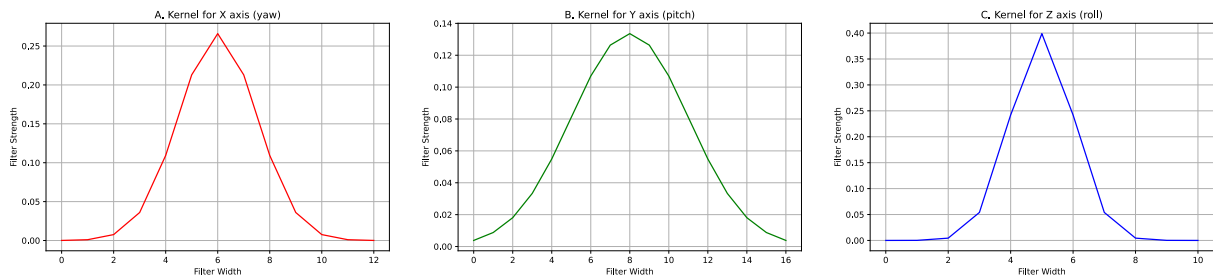


Figure 10: Different Gaussian kernels for X, Y and Z axes

There are three different Gaussian filters used for X, Y and Z axes. Because different axis components of the same signal had different characteristics, we had to specify filters with different parameters which would work best. Gaussian filter worked sufficiently well as per our expectations. An example implementation of Gaussian filter is shown in figure 11.

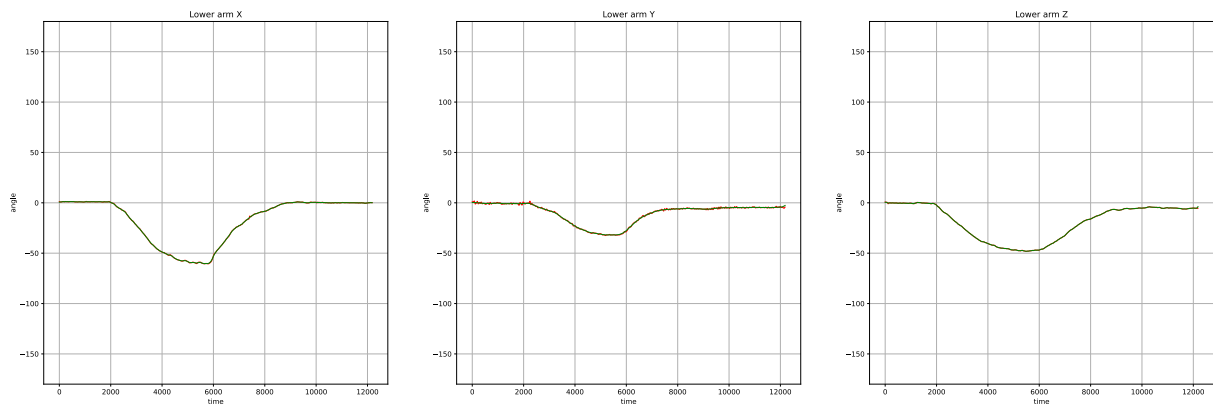


Figure 11: Smoothed signal after applying Gaussian smoothing

In figure 11, the red line is the actual signal and the green line is the smoothed signal which is su-

perimposed on the red one. Parts of the red line that are visible are the noises that have been discarded, and we receive the smoothed outputs.

Our second approach was to use the frequency domain filter. Inclusion of more sensors resulted in longer complex outputs. In this approach, we used FFT to transform a temporal signal to a frequency domain signal. The idea is, in frequency domain, the actual signal should belong to the most dominating frequency levels i.e. have higher amplitude responses in those frequencies. Since noises are signal elements that are random and infrequent, and adds to the original signal to make it uninterpretable, the frequency domain response of noises would be very low with respect to the original signal. The responses would also be spread out among different frequency levels. We would consider a threshold to allow the frequency responses above the threshold while suppressing or discarding the responses below the threshold. The parameters can even be different along different axes.

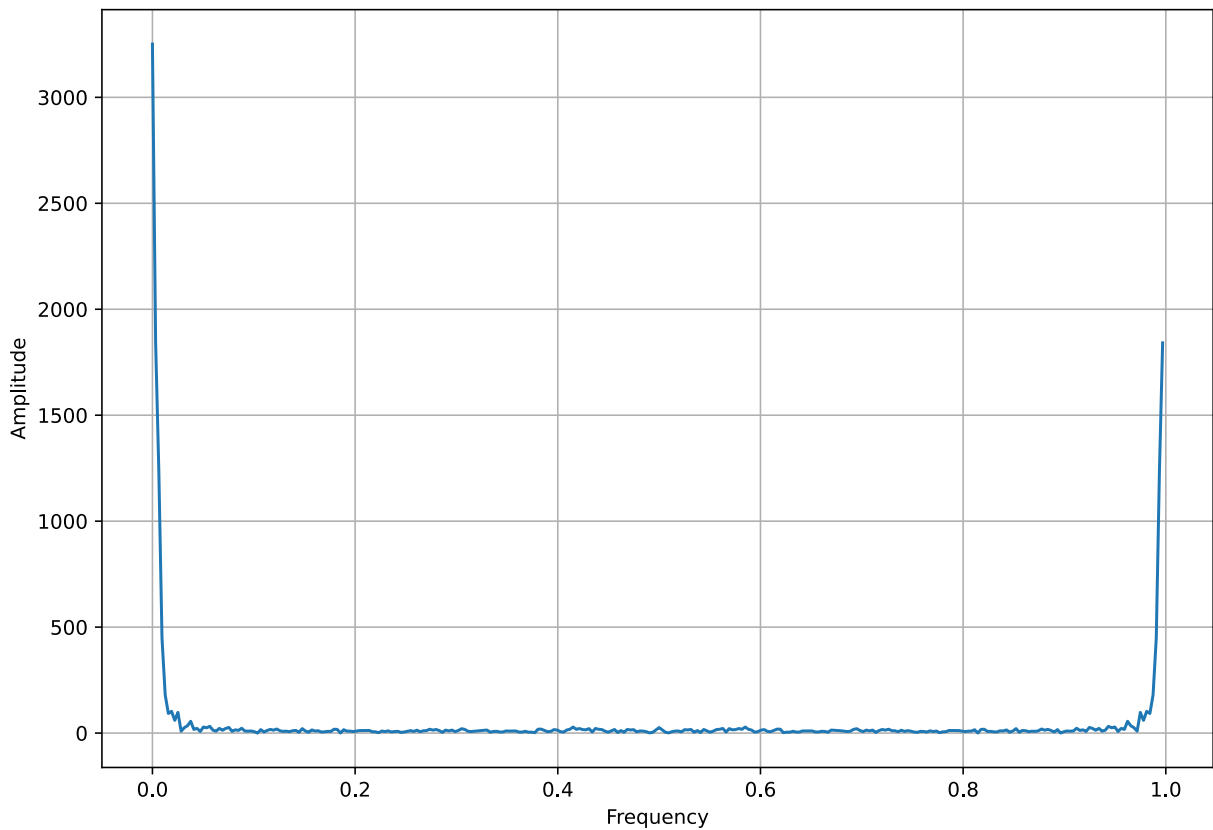


Figure 12: Frequency Domain representation of a signal

According to the frequency domain representation (figure 12) of the signal, the most dominant frequencies with higher amplitudes are the actual signals while almost all the frequency responses with very low amplitudes are the noises.

In our implementation, we have set the threshold to 50 percent, that is all the signals above that threshold would be allowed. The signals below that threshold would be reduced to 30 percent of the original value. The suppressing, instead of discarding is done so that we do not smooth the signal too much. We could also set another threshold above which we would suppress, and would discard the signals below. While this could give more control, it depends totally on the signal itself.

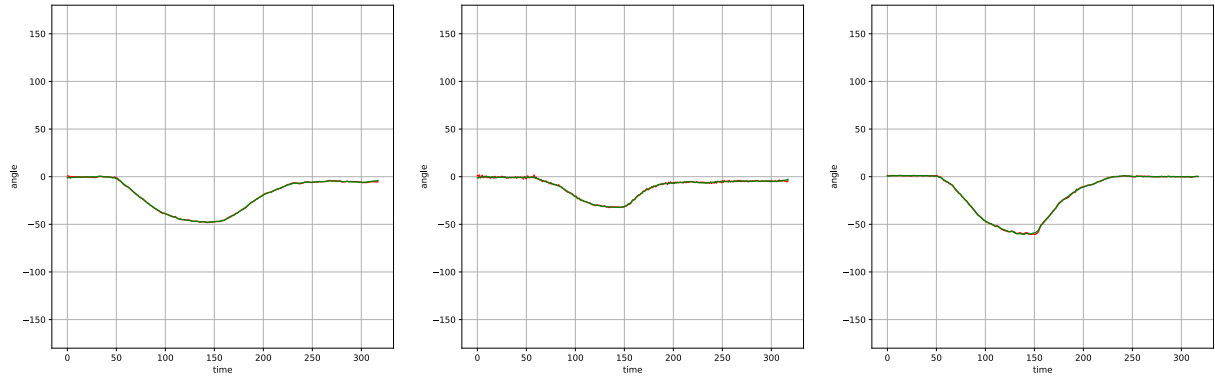


Figure 13: Smoothed signal after applying Frequency Domain smoothing

In figure 13, like figure 11, the smoothed green line is superimposed on the actual red line. Here this filter also performs well. But as we are discarding or suppressing non-dominant signals below a certain threshold, there is a chance that some voluntary vibrations, which are not frequent might also be discarded. To check the filter, we run it over some arbitrary rapid moving sensor readings.

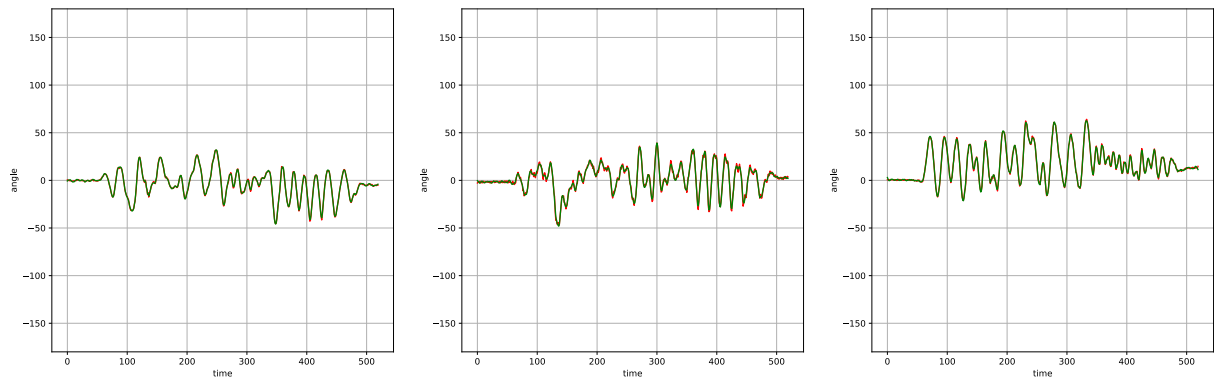


Figure 14: Smoothed signal after applying Frequency Domain smoothing (on rapid moving sensor readings)

In figure 14, the filter achieved expected smoothed output, which indicates the correctness of this filter. Although it is not the filter itself, rather the parameter tuning that can make it work properly. In contrast to this filter, its spatial domain representation is a Gaussian-like filter with corresponding parameters, but still this approach is more flexible in a sense that it gives us more control and understanding of the process.

These two filters are used in our implementation because it suited our use case. To generate each of these outputs, we had to tweak the parameters manually and also had to counteract the drawbacks mentioned in the *Approach* section. Once we have accumulated more data, we can apply machine learning to make the parameters adapt to different types of data. Other than that, the best approach would be to choose the best possible filter and manually or conditionally tweak its parameters.

7.5 Mapping, Discontinuity Solution and Visualization in Blender

After the post processing of data has been completed, it is now ready to be transferred into Blender. We have used Python scripting internal to Blender to transfer and visualize the data.

Blender has its own Python library or module called *bpy*, which can basically mimic every action that a user can manually do in Blender via a Python script with other added functionalities. Blender comes with its own Python 3 runtime and interpreter.

The processed data needs to be mapped with a rigged model in order to visualize it. In our experiment, we have used a model rigged using *Adobe Mixamo*, which provides an AI based auto human rigging system. However, it is best if the model can be purpose built and manually rigged properly so that the calibration pose aligns with the initial pose properly, and each limb is designed and scaled based on the user who is providing input.

To map the data of the sensors to each limb of the model, two things need to be specified. Which sensor or node is mapped to which limb and how the *yaw*, *pitch* and *roll* of the sensors are mapped to model limb orientation, which is in our case, Euler XYZ angles. For this purpose, to keep the system dynamic, we have used a *JSON* script to specify these relationships and used Blender's Python scripting to read and transfer data to the model accordingly. A JSON script for mentioning the relation or link is given in listing 2. The # before yaw, pitch or roll can either be '+' or '-', which denotes the relationship direction between the Euler angle of Blender and our angle. Negative symbol denotes an inverse rotational direction.

```

1 ...
2 "SOU ID" :
3     {
4         "MPU1" :
5         {
6             "target" : "joint or limb name",

```

```

7         "X" : "# yaw 1/2 or pitch1/2 or roll1/2",
8         "Y" : "# yaw 1/2 or pitch1/2 or roll1/2",
9         "Z" : "# yaw 1/2 or pitch1/2 or roll1/2"
10    },
11    "MPU2" :
12    {
13        "target" : "joint or limb name",
14        "X" : "# yaw 1/2 or pitch1/2 or roll1/2",
15        "Y" : "# yaw 1/2 or pitch1/2 or roll1/2",
16        "Z" : "# yaw 1/2 or pitch1/2 or roll1/2"
17    }
18 }
19 ...

```

Listing 2: Example Blender Link Script

Once the relationship is specified, the data can now be transferred. First, the time stamps of each data entry is converted into Blender's target frame using equation 12 and 13 and then the Euler angles are key framed for the target limbs in the target frame number using *bpy* methods.

After the data is key framed animated, there still remains the *Euler Discontinuity* problem as stated before. To solve this, we have used Blender's own *Euler Discontinuity Filter*. It can either be called from the script or be executed manually for the animation curve by going into Blender's *Graph Editor*.

After these steps, the data transferring procedure is complete and it can be used for any desired purpose. In our case, we visualized it directly inside Blender and render out an animation into a video using *Eevee* render engine provided with Blender.

8 Evaluation

8.1 Error Measurement and Error Propagation

The ideal method for measuring IMU or MARG sensor array errors is to create a mechanical or vision based rig and test the sensors against precise, predetermined motions[34]. Two types of errors are mainly measured.

1. **Static Error:** When the sensor motion is tested against a reference motion oscillating at *less* than $5^\circ/s$.
2. **Dynamic Error:** When the sensor motion is tested against a reference motion oscillating at *greater* than $5^\circ/s$.

Error for each individual axis of sensor output is measured independently since there are differences between the readings of accelerometer, gyroscope and magnetometer.

Due to resource and time constraints and Covid pandemic during the time of this experiment, physical error measurement procedures could not be carried out. Error data from the original report of Sebastian Madgwick are used instead as the base of our evaluation [5].

8.1.1 Per Sensor Error Measurement

Both static and dynamic errors per axis are given in table 5 and 6 respectively.

Axis	Madgwick Filter	Kalman Filter
Pitch (X)	0.581°	0.789°
Roll (Y)	0.502°	0.819°
Yaw (Z)	1.073°	1.150°

Table 5: Static Error per Axis

Axis	Madgwick Filter	Kalman Filter
Pitch (X)	0.625°	0.769°
Roll (Y)	0.668°	0.847°
Yaw (Z)	1.110°	1.344°

Table 6: Dynamic Error per Axis

From table 5 and 6, it can be seen that Madgwick filter produces less error, both static and dynamic, for all the axes.

8.1.2 Error Propagation

In our system, orientation for each of the limbs is calculated with the help of chain subtraction as shown in the *Approach* section, subsection 6.4, which is an additive linear process. Measurement error of any node can be calculated using equation (17).

$$\overrightarrow{Propagated\ Error} = \overrightarrow{Sensor\ Axis\ Error} \times Chain\ Length \quad (17)$$

The largest node chains in our experiment have 4 nodes, resulting in maximum static error of [2.324, 2.008, 4.292] ° and dynamic error of [2.500, 2.672, 4.440] °, where the elements of the vectors are *pitch*, *roll* and *yaw* respectively. All the chain lengths are calculated from node 'N0_0'. For instance 'N1_1' has chain length of 4 where the chain consists of 'N0_0', 'N0_1', 'N1_0' and 'N1_1' in order.

Both static and dynamic errors for all the nodes along with respective chain lengths are shown in table 7. Both dynamic and static errors are measured in unit degree and the order of elements in the vectors are *pitch*, *roll* and *yaw* respectively.

Node	Chain Length	Static Error (°)	Dynamic Error (°)
N0_0	1	[0.581, 0.502, 1.073]	[0.625, 0.668, 1.110]
N0_1	2	[1.162, 1.004, 2.146]	[1.250, 1.336, 2.220]
N1_0	3	[1.743, 1.506, 3.219]	[1.875, 2.004, 3.330]
N1_1	4	[2.324, 2.008, 4.292]	[2.500, 2.672, 4.440]
N2_0	3	[1.743, 1.506, 3.219]	[1.875, 2.004, 3.330]
N2_1	4	[2.324, 2.008, 4.292]	[2.500, 2.672, 4.440]
N3_0	2	[1.162, 1.004, 2.146]	[1.250, 1.336, 2.220]
N3_1	3	[1.743, 1.506, 3.219]	[1.875, 2.004, 3.330]
N4_0	2	[1.162, 1.004, 2.146]	[1.250, 1.336, 2.220]
N4_1	3	[1.743, 1.506, 3.219]	[1.875, 2.004, 3.330]
N5_0	3	[1.743, 1.506, 3.219]	[1.875, 2.004, 3.330]
N5_1	4	[2.324, 2.008, 4.292]	[2.500, 2.672, 4.440]

Table 7: Per Node Propagated Error

8.2 Cost

The whole system was developed and built using commercially available hardware and open source software. The total cost is only dependent on the purchased hardware and components which cost around *BDT* 30,000.

9 Conclusion and Future Work

In our report, we have proposed a sensor based human posture recognition method that is both cost effective and suitable for capturing motion data without any kind of mobility or environmental constraints. The proposed system is not only restricted to the motion of human beings and can be extended to capture the motion of machines, animals and other moving objects. The system is also highly modular, making it an easily scalable solution. The method proposed here works as a basic framework for capturing motion data using commercially available hardware which can further be extended and purpose built for specific applications. The proposed method is not restricted to the specific hardware that we have used and can be implemented using any similar available hardware, while allowing mix matching as well, if the application demands.

There is still room for more testing and extension within this proposed system. A formal testing framework needs to be established and the proposed system needs to be measured against either a control

or similar motion capturing systems to find out its effectiveness. There are many layers or modules within the proposed system that can be modified and altered to change the outcome or performance of the system. The whole system can be kept in quaternion format to eliminate many of the problems faced within the processing pipeline. However, use of quaternion is not common to the mass and might prove to be uninterpretable for many users. The smoothing filter used to smooth out the data and suppress noise is highly purpose dependent as mentioned earlier. Different types of filters can be used to test out the effectiveness of the data smoothing. The system can be tested extensively both by building a standard machine based automated approach or by taking readings of the same motion using our proposed system and a vision based system, such as *Microsoft Kinect*, *PSVR* or *Intel RealSense* depth camera or sensor based system, such as *Rokoko* motion capture system. These two approaches provide two different paradigms of testing. Finally, use cases of our proposed system are yet to be analyzed practically. We believe, from machine learning to computer graphics, any place where a vision based motion capture system is in use, our system can be a possible and effective substitution. Moreover, one of the main goals of our approach is to provide a cost effective solution for motion capture which should be widely accessible and easily buildable. It can only be proven effective when proper experiments are carried out to prove the effectiveness, accuracy and viability of the system and its use cases.

References

- [1] M. J. M. Vasconcelos and J. M. R. Tavares, “Human motion analysis: methodologies and applications,” *CMBBE 2008*, 2008.
- [2] A. N. Mohamed, “A novice guide towards human motion analysis and understanding,” *arXiv preprint arXiv:1509.01074*, 2015.
- [3] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, vol. 82, pp. 35–45, 03 1960.
- [4] R. Mahony, T. Hamel, and J.-M. Pflimlin, “Nonlinear complementary filters on the special orthogonal group,” *IEEE Transactions on automatic control*, vol. 53, no. 5, pp. 1203–1218, 2008.
- [5] S. Madgwick, “An efficient orientation filter for inertial and inertial/magnetic sensor arrays,” *Report x-io and University of Bristol (UK)*, vol. 25, pp. 113–118, 2010.
- [6] R. Rosales and S. Sclaroff, “Inferring body pose without tracking body parts,” in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, vol. 2, pp. 721–727 vol.2, 2000.
- [7] O. Ozturk, Toshihiko Yamasaki, and Kiyoharu Aizawa, “Tracking of humans and estimation of body/head orientation from top-view single camera for visual focus of attention analysis,” in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pp. 1020–1027, 2009.
- [8] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, “Real-time human pose recognition in parts from single depth images,” in *CVPR 2011*, pp. 1297–1304, 2011.
- [9] D. F. Glas, T. Miyashita, H. Ishiguro, and N. Hagita, “Laser tracking of human body motion using adaptive shape modeling,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 602–608, 2007.
- [10] G. Glonek and A. Wojciechowski, “Hybrid orientation based human limbs motion tracking method,” *Sensors*, vol. 17, no. 12, 2017.
- [11] J. Yin, D. Zhu, S. Min, and Z. Wang, “Depth maps restoration for human using realsense,” *IEEE Access*, vol. PP, pp. 1–1, 08 2019.

- [12] F. Jiang, X. Yang, and L. Feng, “Real-time full-body motion reconstruction and recognition for off-the-shelf vr devices,” in *Proceedings of the 15th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry-Volume 1*, pp. 309–318, 2016.
- [13] E. R. Bachmann, I. Duman, U. Y. Usta, R. B. McGhee, X. P. Yun, and M. J. Zyda, “Orientation tracking for humans and robots using inertial sensors,” in *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA’99 (Cat. No.99EX375)*, pp. 187–194, 1999.
- [14] E. R. Bachmann, Xiaoping Yun, and R. B. McGhee, “Sourceless tracking of human posture using small inertial/magnetic sensors,” in *Proceedings 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation. Computational Intelligence in Robotics and Automation for the New Millennium (Cat. No.03EX694)*, vol. 2, pp. 822–829 vol.2, 2003.
- [15] Xiaoping Yun, M. Lizarraga, E. R. Bachmann, and R. B. McGhee, “An improved quaternion-based kalman filter for real-time tracking of rigid body orientation,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, pp. 1074–1079 vol.2, 2003.
- [16] Z. Zhang, X. Meng, and J. Wu, “Quaternion-based kalman filter with vector selection for accurate orientation tracking,” *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 10, pp. 2817–2824, 2012.
- [17] X. Yun and E. R. Bachmann, “Design, implementation, and experimental results of a quaternion-based kalman filter for human body motion tracking,” *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1216–1227, 2006.
- [18] D. Roetenberg, P. J. Slycke, and P. H. Veltink, “Ambulatory position and orientation tracking fusing magnetic and inertial sensing,” *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 5, pp. 883–890, 2007.
- [19] J. K. Lee and E. J. Park, “A minimum-order kalman filter for ambulatory real-time human body orientation tracking,” in *2009 IEEE International Conference on Robotics and Automation*, pp. 3565–3570, 2009.
- [20] G. Šeketa, D. Džaja, S. Žulj, L. Celić, I. Lacković, and R. Magjarević, “Real-time evaluation of repetitive physical exercise using orientation estimation from inertial and magnetic sensors,” in *First European Biomedical Engineering Conference for Young Investigators (Á. Jobbágy, ed.)*, (Singapore), pp. 11–15, Springer Singapore, 2015.

- [21] H. Ahmed and M. Tahir, "Improving the accuracy of human body orientation estimation with wearable imu sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 66, no. 3, pp. 535–542, 2017.
- [22] S. Bakhshi, M. H. Mahoor, and B. S. Davidson, "Development of a body joint angle measurement system using imu sensors," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 6923–6926, 2011.
- [23] H. J. Luinge and P. H. Veltink, "Measuring orientation of human body segments using miniature gyroscopes and accelerometers," *Medical and Biological Engineering and computing*, vol. 43, no. 2, pp. 273–282, 2005.
- [24] S. Zihajehzadeh, D. Loh, M. Lee, R. Hoskinson, and E. J. Park, "A cascaded two-step kalman filter for estimation of human body segment orientation using mems-imu," in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 6270–6273, 2014.
- [25] H. Zhao and Z. Wang, "Motion measurement using inertial sensors, ultrasonic sensors, and magnetometers with extended kalman filter for data fusion," *IEEE Sensors Journal*, vol. 12, no. 5, pp. 943–953, 2012.
- [26] J. Carberry, G. Hinchly, J. Buckerfield, E. Tayler, T. Burton, S. Madgwick, and R. Vaidyanathan, "Parametric design of an active ankle foot orthosis with passive compliance," in *2011 24th International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 1–6, 2011.
- [27] G. D. Voinea and G. Mogan, "Development of a wearable scoliosis monitoring system using inertial sensors," in *Advanced Research in Aerospace Engineering, Robotics, Manufacturing Systems, Mechanical Engineering and Biomedicine*, vol. 811 of *Applied Mechanics and Materials*, pp. 353–358, Trans Tech Publications Ltd, 12 2015.
- [28] R. Paul, *Robot Manipulators: Mathematics, Programming, and Control : the Computer Control of Robot Manipulators*. Artificial Intelligence Series, MIT Press, 1981.
- [29] J. M. McCarthy, *Introduction to Theoretical Kinematics*. Cambridge, MA, USA: MIT Press, 1990.
- [30] S. Mansoor, U. I. Bhatti, A. I. Bhatti, and S. M. D. Ali, "Improved attitude determination by compensation of gyroscopic drift by use of accelerometers and magnetometers," *Measurement*, vol. 131, pp. 582–589, 2019.
- [31] J. J. Craig, "Introduction to robotics mechanics and control third edn," 2005.

- [32] V. Mansur, S. Reddy, S. R., and R. Sujatha, "Deploying complementary filter to avert gimbal lock in drones using quaternion angles," in *2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*, pp. 751–756, 2020.
- [33] J. M. Cooke, M. J. Zyda, D. R. Pratt, and R. B. McGhee, "Npsnet: Flight simulation dynamic modeling using quaternions," *Presence: Teleoperators & Virtual Environments*, vol. 1, no. 4, pp. 404–420, 1992.
- [34] A. Godwin, M. Agnew, and J. Stevenson, "Accuracy of inertial motion sensors in static, quasistatic, and complex dynamic motion," *Journal of biomechanical engineering*, vol. 131, p. 114501, 11 2009.