Otto-von-Guericke-University Magdeburg
Faculty of Electrical Engineering and Information Technology

Institute for Medical Engineering
Chair for Catheter Technologies and Image Guided Therapies

Research Track Project

IMU-Based Measurement System for Kinematic Estimation of Knee Joint Motion

Review Date: 30th of November 2019

Head of Chair
Catheter Technologies
Prof. Michael Friebe

Submitted by
Mishuk Mitra (221621)

Supervised by
Thomas Suehn

Objective of the Study

‘‘The aim of this research is setting up kinematic estimation system (hardware development) and evaluation of a system for kinematic estimation of the knee joint motion, based on two inertial measurement units (IMU). The project work is concerned to establish i2c communication between raspberry pi and sensor (xsens) by python programming and measuring the inertial measurement motion data for the estimation of knee joint motion.’’

Table of Contents

Introduction

Knee physiology

The knee is the largest joint in the human's body and one of the most complex. It has lateral and medial meniscus which absorbs shock and stable the joint. Anterior and posterior ligaments prevent displacement of the tibia. Knee generates motion because of the flexion and extension of leg. There are three different types of motion take place for three different types of movement - Anterior posterior gliding movement, Rolling hinge like movement and Rotation of lower leg to its long axis [1]. Measurement of the vibrations or motions produced by a joint as its moves that utilizes to identify disorders and functional pathologies.

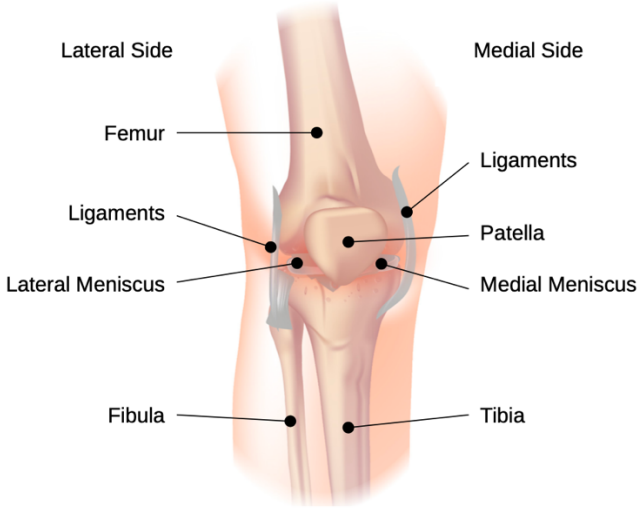

Figure 1: Anatomy of knee

Motivation

Due to the complexity of knee joint, it is the most vulnerable part of human body. Most of the knee injuries including strain, fracture, dislocation, meniscus injury, smirch is sport related. There are several methods to diagnose the knee injuries. Common non-invasive methods for the examination of the knee joint are palpation and auscultation. In contrast, arthroscopy is a minimally-invasive procedure used for the evaluation and treatment of the joint. It gives insight to the inner structure of the joint but comes with the two major drawbacks of invasiveness and little information about the functional behaviour of the joint in motion. To overcome the limitations of current examination methods, measurement of the acoustic signals of the knee joint during specific movement is a promising approach. This method is called Vibroarthrography (VAG) utilizes the occurring acoustic signals to identify disorders and functional pathologies and could serve as a non-invasive diagnostic tool. Another main advantage of the VAG is no need to expose radiation to the patient. For the analysis of the

acquired sound signals with respect to the osteokinematics, the assessment of the health status of the joint and the reproducibility of the measurements, it is crucial to acquire information regarding the particular knee joint angle.

To estimate the knee joint kinematics in a convenient way and allow the integration of the VAG and joint angle measurement system in a wearable device, an approach based on inertial measurement data will be investigated. Using an automated monitoring tool, it would be possible to estimate activity level of knee, the status of knee health and the intensity of training. It could also be used to medical diagnoses purpose of knee at the hospital.

State of the Art

Knee joint kinematic estimation

Vibroarthrography is the another swiftly growing technique for knee joint kinematic estimation. It has the potential to be the most prominent and reliable diagnostic tool in the future because it is an inexpensive technique and totally radiation free. Knee joint is a particular kind of joint of patella, tibia, femur and some ligaments which produce vibrations because of the relative movement and motion. These vibrations propagate through the surrounding tissues and produces VAG signals. So, it is certain that producing signal will be different for the different relative motion. And by detecting the motions for every possible angle we can measure kinematic estimation for knee joint motions [2][3][4]. And by analysing this motion we can estimate the activity level of knee as a such potential diagnostic tool.

Inertial Measurement Unit (IMU)

Inertial Measurement Unit refers the name inertial sensing technology. It is an embedded system of motion sensors. In medical system, inertial sensors are commonly used for measuring linear acceleration, rate of turn / angular velocity, magnetic field of an object in the three-dimensional coordinate system [5]. Some commercially available IMU devices uses algorithm of Euler angles to estimate quaternion which represent orientation and rotation of the object with a fixed field. Although the existence of ferromagnetic material in the IMU for measuring magnetic field may occur some disturbances of the orientation accuracy [6]. In this context, the measurement system provides some IMU data needed to implement a kinematics estimation method for knee joint axis and position identification.

Optical measurement system

Optical measurement system uses optical sensor which translate quantity of light rays to electrical signal which can be used as an inertial measurement unit. There is a light source which produces light which is connected to photo detector through feed and return optical fiber and measurement device. Depending on light there is a change of output power detected by photodetector acts as a sensing device both external and internal part. External part of the sensor trans-receive quantity of rays while internal part can sense change in direction, pressure, temperature etc. That's why it could be used as motion sensing technology in medical system which is used for motion tracking of human body part, plays a great role for medical diagnosis. This reliable lightweight measurement system produces high sensitivity and accuracy. Nevertheless, ambient light interference can affect the measurement which limited the dynamic range and arise some questions in terms of limited long-term stability [9].

Figure 2: Optical fiber sensors

Goniometric measurement system

Goniometric is an old technique which uses a Goniometer to measure knee joint angle. It consists of three part. One part moves with the flexion and extension of knee while other part remains stationary. Third circular part contains measuring scale (fulcrum) which measures in degree. The main challenge of using this technic is positioning. Inexact positioning results inaccurate result. So, using this system is time consuming and has less efficient [12].

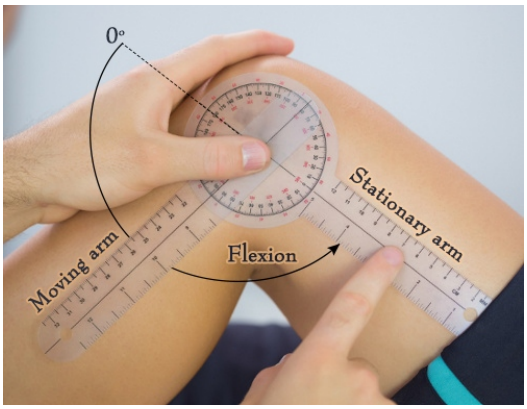

Figure 3: Measuring motion of knee flexion-extension using Goniometer

Xsens as IMU

The sensors (Accelerometer, Gyroscope, Magnetometer and quaternion) used to detect the motion are embedded into a system called Inertial Measurement Unit (IMU). These sensors are widely used in robotics. Now a days many microelectronic mechanical systems (MEM's) based IMU sensors are used to measure kinematic estimation. In this project we used xsens MTi 1-series [3] as a motion tracking technology for kinematic estimation. It is a MEMS based inertial measurement unit which is an excellent choice for use in high-volume applications and with industrial grade accuracy [7]. It also comes with a development kit which always gives a special diversity to work with.

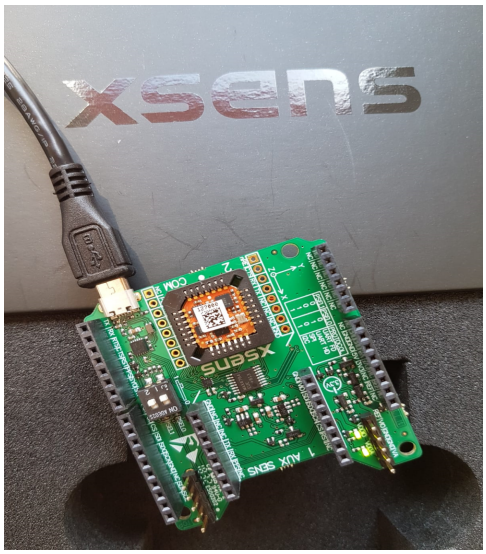

Figure 4: Xsens MTi 1-series motion Sensor (Development kit) (Xsens, 2019)

Methodology

Scope of the proposed research track project is the hardware development and evaluation of a system for kinematic estimation of the knee joint motion, based on two inertial measurement units (IMU). The two sensors will be attached to each side of the knee joint and integrated into a wearable device (e.g. knee bandage). The used attachment method should allow an arbitrary mounting position of the sensors (with respect to the alignment along the joint axes). The plausibility and accuracy of the acquired / recorded IMU data will be checked. Some straps are used for amalgamate of the sensors and tightening enough with the knee joint for robustness performance.

System design

Since it severe to attach the sensor on the leg for getting the accurate coordinate axes of knee joint axis, here two inertial measurement units have been used and then data will be transformed according to joint related coordinate systems [8] weather two coordinate axes associate together or one axes is longitudinal to other. Then transforming both coordinate axes enable us to measure a joint motion axes though accuracy of this technique is sometimes confined. Here, two xsens sensors have been used as slave with the raspberry pi zero which acts as a master device and they are intercommunicated by I2c interfacing. And for this communication we configured two i2c addresses for the two sensor by enabling different address port (A0,A1,A2) [6]. Here two push buttons and two LED's also used to perform the operation as desired.

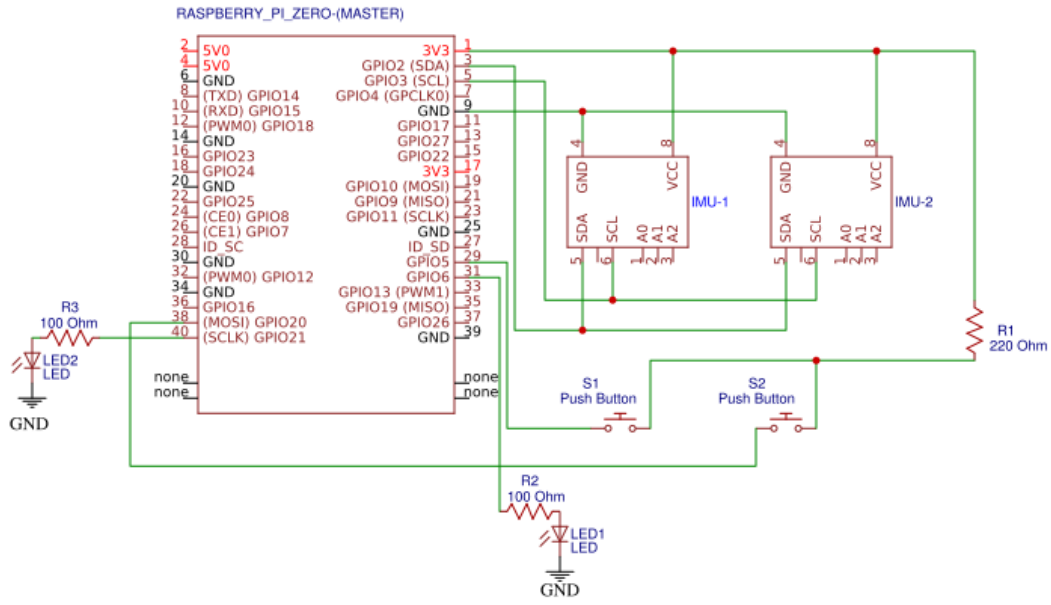

Figure 5: Circuit diagram for I2C communication between Raspberry Pi zero and two IMU sensors

Once master get the slave addresses, we set some python programming to communicate and provide some live motion data.

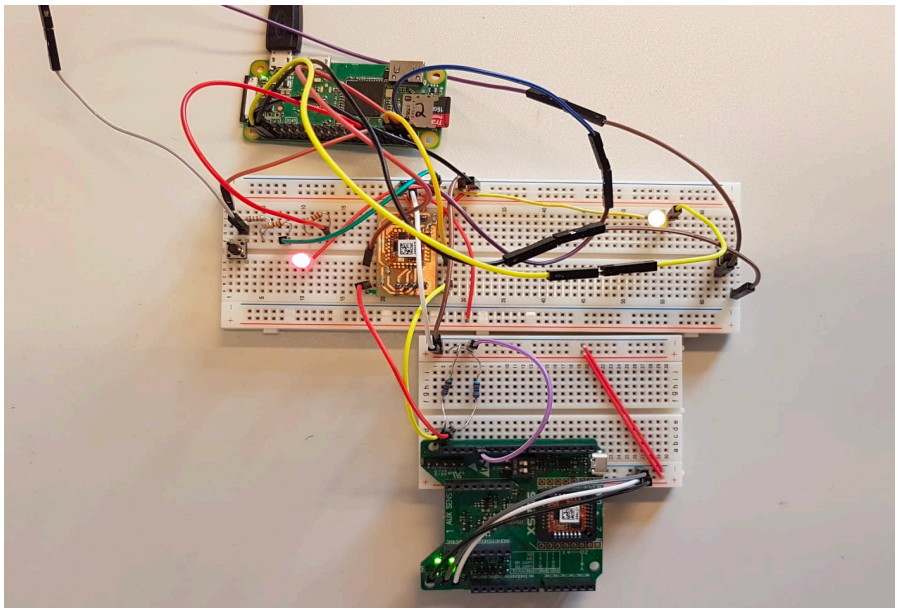

Figure 6: Physical I2C connection of raspberry pi zero (Master) and two IMU sensors (slaves)

The system connection designed in a way such that yellow LED-1 will blink when the device is turned on and it ensures the device is configured and ready to measure. By pressing the left push button-2 the system goes to measurement state and it start measuring with the blinking of red LED-2. Second push will make the LED-2 turn off indicates the system stops measuring. Each time of measuring data will be saved in different excel sheet which enables to identify for different joint axes motion data for different time interval. And at last but not the least by holding the push button-1 makes the system safety shutdown.

Device Design

The equipment's are used to design the IMU: two xsens motion sensor, one raspberry pi, one lipo battery to power the device, one power management unit, two push-up buttons and two LED's. Here sensors are designed for i2c interfacing and connected to the raspberry pi for i2c communication. The finalized IMU prototype for measuring the real time motion data is given below.

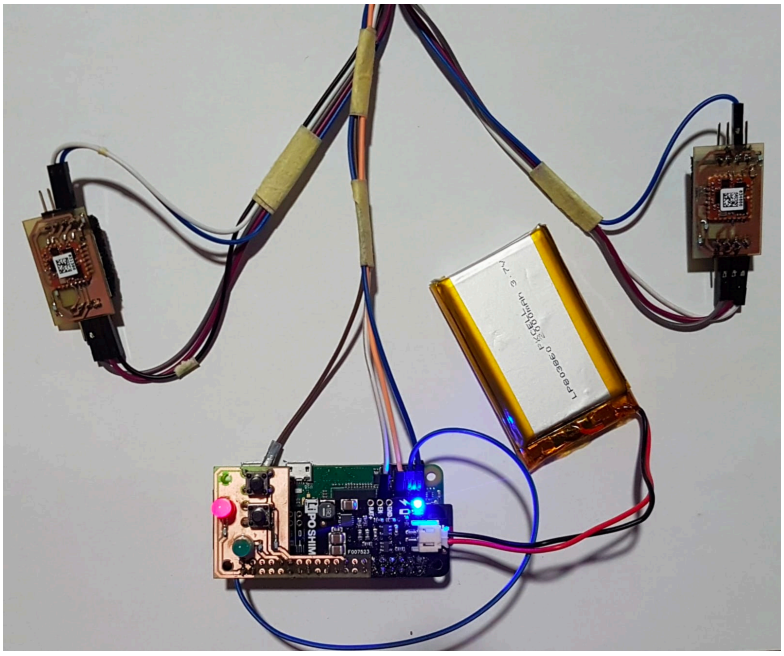

Figure 7: Final prototype design for measurement

Communication

For I2C communication between pi and xsens and measuring live motion data need some python programming. When slaves are connected to the master, it has different addresses for both the sensors. Then we have to initialize and configure the sensors and put the sensor in the measurement state. We are working with raspberry pi zero and maximum i2c speed of pi is 400Kbps. By default, xsens sensor gives values 100 time in 1 second and here for reading values with 100Hz we need to set maximum speed for i2c at 400Kbps. For working with i2c we created an i2c class. Because in unix system everything's are files conception. SMbus cannot read and write more than 32 bytes, but we need to read and write more than 32 bytes. After creating i2c class, when we open file and say to the system that this file is a i2c device and at unix system constant i2c slave is 0x0703. Here making some functions to read, write, create, close etc.

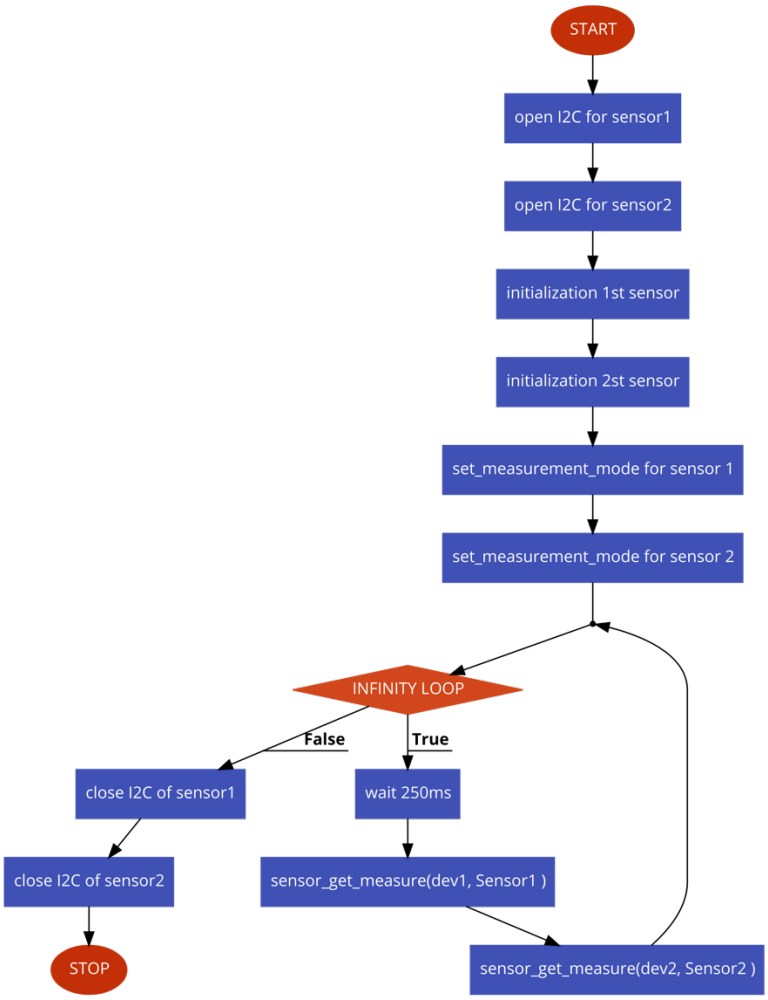

Figure 8: Flow chart for measuring live motion data for python programming

we have to import some python libraries for reading, writing, making time delay for sleep and controlling some essential parameters. At first, selecting the i2c sensor addresses to initialize the sensor and put the sensor to configure state. Secondly, set the config mode for desired data output. After configuration put the sensors to measure mode through an infinity loop. If its false program stops and close the i2c sensors and if it is true then wait a while and sensors get measured. And we will have the desired live data output.

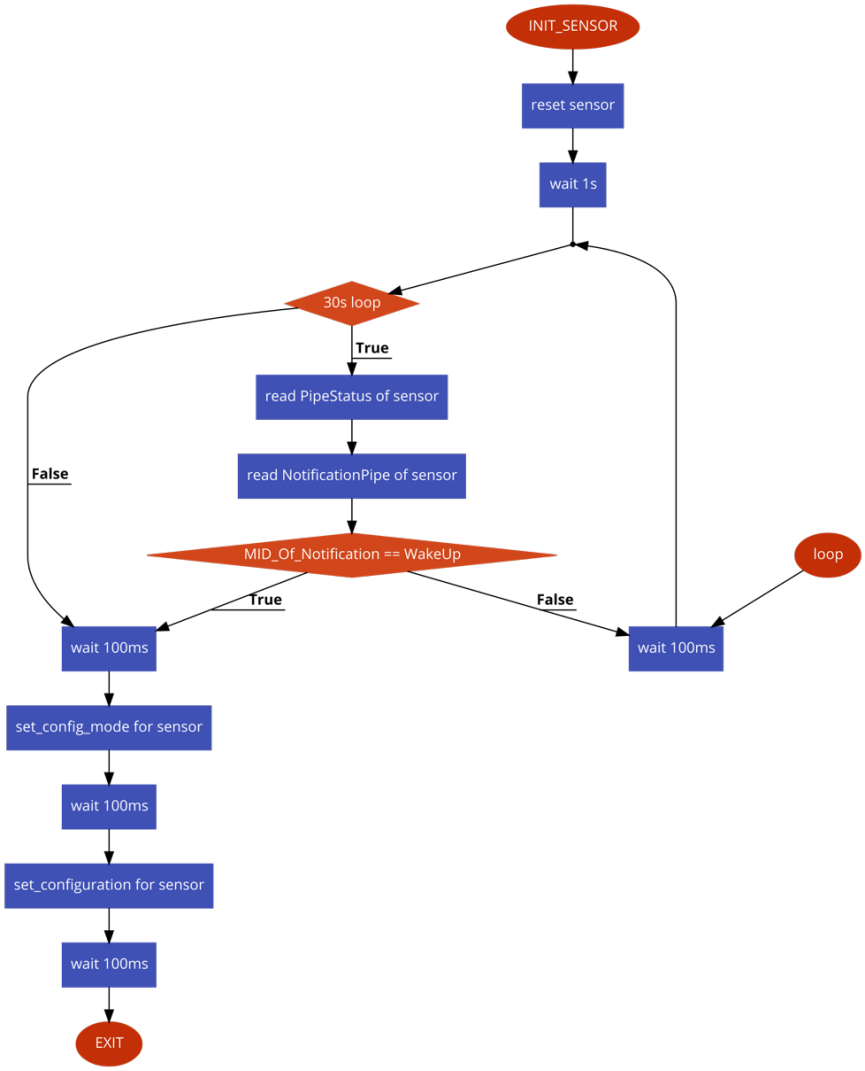

Figure 9: Flow chart to initialization of the sensors for python programming

I2C always use address opcode instead BID. when we use write function first byte set from op -code. For initialization, first we send 'Reset (0x40)' [6] to the sensor. After resetting, we wait for wake up because sensor needs time after reset, wait 1 second. In this loop we must read notifications from sensor. when we catch 62 (0x3E) (Wakeup) - it means that sensor was reset and boot. 300 in the loop and in the end time sleep 0.1s (300*0.1 =30s). Max time to wake up for the loop will be 30s. But we really go out from this loop faster when catch Wakeup and then break. we work with Pipe Status (4) to know how many bytes we must read now we read 4 bytes to know the length of massages. 1st and 2nd bytes are length of Notification Pipe and 3th and 4th bytes are length of Measurement Pipe. Read Notification Pipe, if it 0x3E = Wakeup [6] then go out from this loop. Then put sensor to Configure state and set configuration for sensor for Accelerometer, Gyroscope, Magnetometer and Quaternion refreshing 1 time in 1 second.

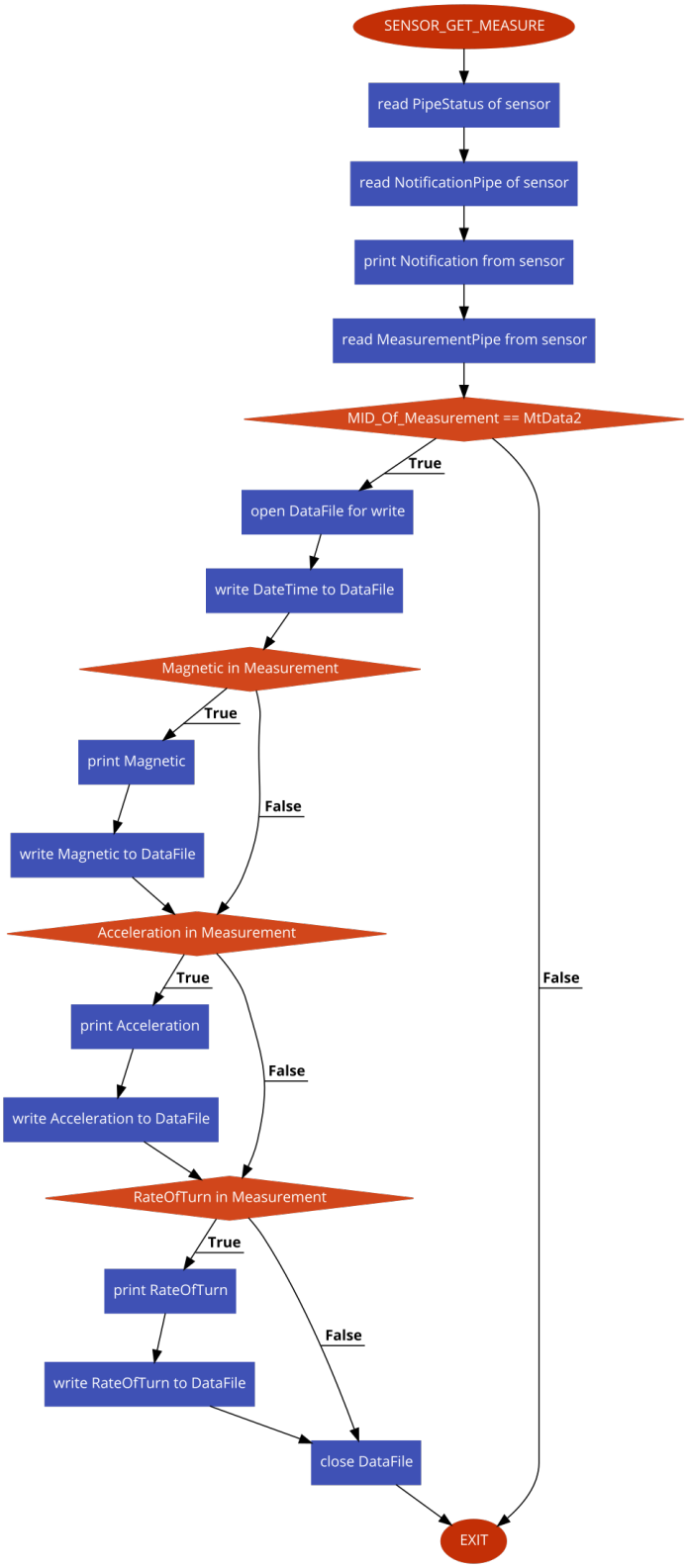

Figure 10: Flow chart to set up the measurement mode of the sensors for python programming

we work with Pipe Status (4) to know how many bytes we must read. Now we read 4 bytes to know the length of massages. Then read Notification Pipe and Measurement Pipe and make checksum of message and check it. According to the xsens documentation MtData2 is MID 0x36 [6]. If the Checksum is correct with the MtData2 then save all the values to .csv file.

Measurement Design and setup

Flexion and Extension of knee movement take place in the upper part of the meniscus while femur and tibia glides over it because of movement. During flexion transverse axis moves back and downwards and during extension knee moves to reverse direction and go to transverse axis.

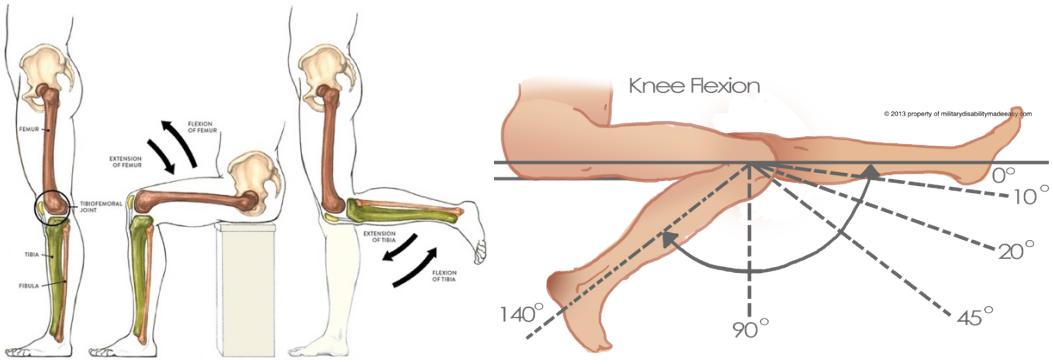

Figure 11: 1) Possible Flexion-Extension of knee joint [13]. 2) Range of knee joint motion [14]

When no movement occurs means knee is at transverse plane, there is no flexion occurs so movement is 0°. When knee is flexed and fully bent maximum possible movement is 140°. For sports most functional movement up to 120°. So, physical ability mostly depends on according to the rotation of movement. For this measurement we used two sensors, one is on the upper leg to track the movement of femur and other is on the lower knee for the movement of tibia during the flexion and extension. Keeping the leg in transverse plane, second sensor mostly track the flexion-extension of tibial movement respect to the first sensor. On the other hand, for the femoral movement upper sensor tracks the femur movement respect to tibia. After measuring we get three axis data which varies according to the movement of knee for both two sensors. Then plot these data to identify the changes of axis during movement of precession. The designed inertial measurement unit and its locations of placement is given below, while experiments were performed in real time.

Figure 12: Sensors placement locations and measurement procedure

Results

To compare our result of both upper and lower sensors initially we took into consideration the standard output of the xsens kit as a reference with the help of MT software suite. The software suite usually gives the real time graphical representation of movement for the motion sensor including Accelerometer, Angular velocity, Magnetic field, Quaternion etc. In the following section we plotted the individual experimental data for flexion and extension for the both sensors and compare these plots with the standard output of the sensor which is shown in figure 13.

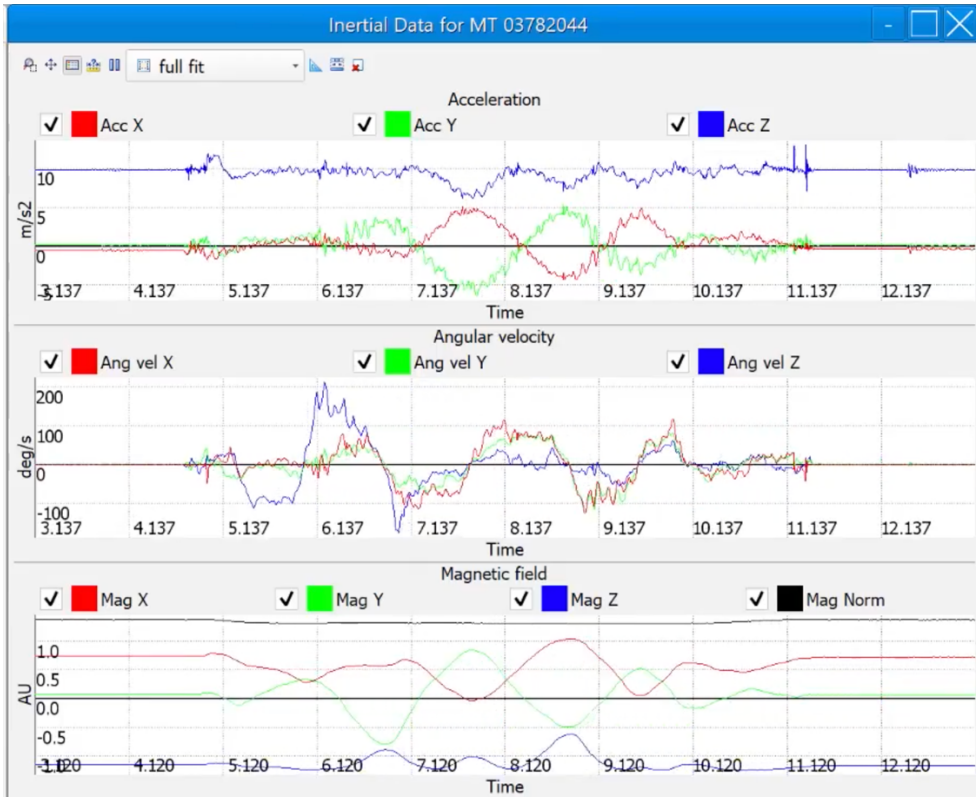

Figure 13: Graphical standard output of MT software suite for motion tracking [6]

Experimental Result

Acceleration

Acceleration is measured by the rate of change of velocity of the knee movement with respect to time in second. Acceleration is a vector quantity where every axis has a velocity. According to the flexion and extension every axis changes its direction with respect to time and the IMU system sense these movements for the both sensors to measure the motion for individual knee parts for kinematic estimation. For measuring acceleration one axis always gain gravitational force (g) because of gravity.

Table 1: Rate of change of velocity for both sensors

	Upper Sensor (m/s2)				Lower sensor (m/s2)		
Time (s)	X	Y	Z		X	Y	Z
1	-9,705818176	-1,283555865	-0,97163415		-9,643976212	-1,646699786	0,034210924
2	-10,0575428	-0,972580254	0,526682436		-9,141031265	-1,811846495	-2,709074736
3	-9,324007034	-1,452648044	3,192214012		-1,893517733	-3,519210339	-8,935698509
4	-9,262457848	-1,149349332	3,328930616		-5,598875523	-3,204815865	-7,325954914
5	-9,314162254	-0,372737616	2,625941277		-9,141348839	-2,702304363	-2,653731585
6	-9,735490799	-0,876433134	1,606188536		-9,537993431	-1,637293577	1,083605647
7	-9,630634308	-0,93140322	2,596264124		-6,787938118	-2,71824646	-6,602828026
8	-9,135177612	-0,514238238	2,952707767		-1,346960545	-3,136825085	-9,284866333
9	-9,270467758	-1,045334935	3,125994921		-7,007166862	-2,838220835	-6,481112003
10	-9,523771286	-0,764781773	2,231315851		-9,289283752	-2,27294445	0,506406188
11	-9,783691406	-0,840683281	1,219012976		-9,484154701	-2,53016901	0,864869237
12	-9,500424385	-1,179095507	2,423717976		-5,852385521	-3,278666019	-7,436709881
13	-9,373953819	-1,253314734	3,152737141		-2,47256422	-3,562229156	-8,860964775
14	-9,475522995	-1,085405469	2,53704834		-8,792198181	-3,125797987	-4,186294079
15	-9,666213036	-0,871023238	1,488176107		-9,474010468	-1,886582375	1,563883901
16	-9,912669182	-0,959118843	1,789016247		-8,180336952	-2,82002759	-5,139484406
17	-9,279493332	-1,291783929	2,978827953		-2,704292297	-3,465852261	-8,830625534
18	-9,124192238	-0,706593454	3,067784309		-8,096886635	-3,236749411	-5,061518669
19	-9,710693359	-0,759939253	1,890487075		-9,368823051	-1,89155674	1,56647265
20	-9,621582031	-1,103065014	3,042118549		-7,580892563	-2,948143959	-5,762324333
21	-9,209636688	-1,627889276	3,667690277		-1,662340164	-3,075443029	-9,154450417
22	-9,35233593	-0,458295792	3,041507721		-9,338891983	-2,37146759	-2,238464594
23	-9,761161804	-0,823282361	0,594999075		-9,70882988	-1,522434831	0,260669857

In the table, shows the measured acceleration data because of knee flexion and extension for both upper and lower sensors and it shows some changes because of relative motion.

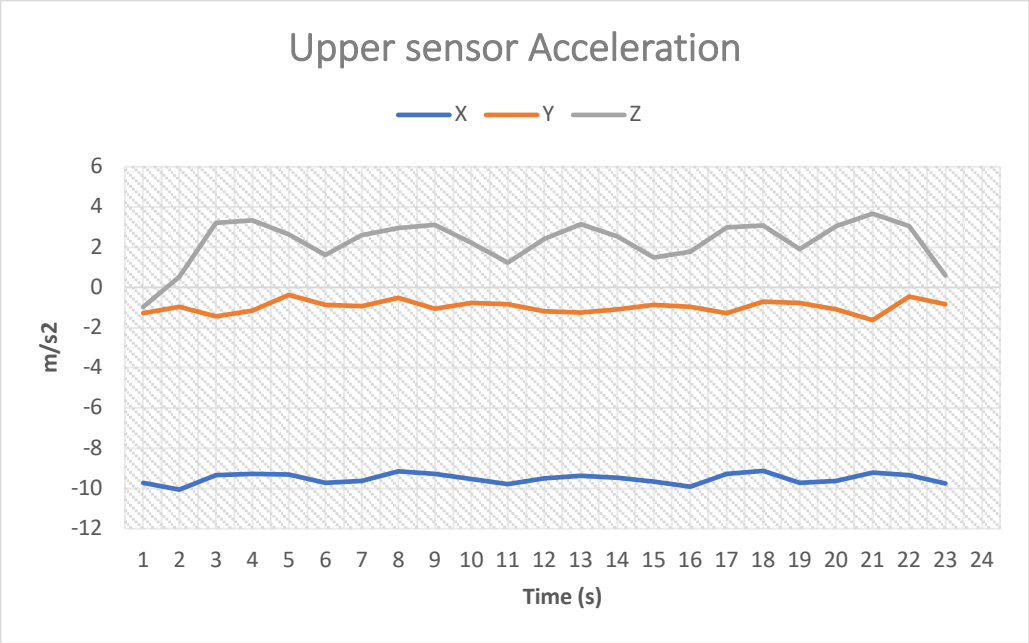

Figure 14: Rate of change of velocity of relative motion for the upper sensor

Here, x-axis shows the gravitational force and remains constant most of the time period because there is no movement during the tibial flexion and extension.

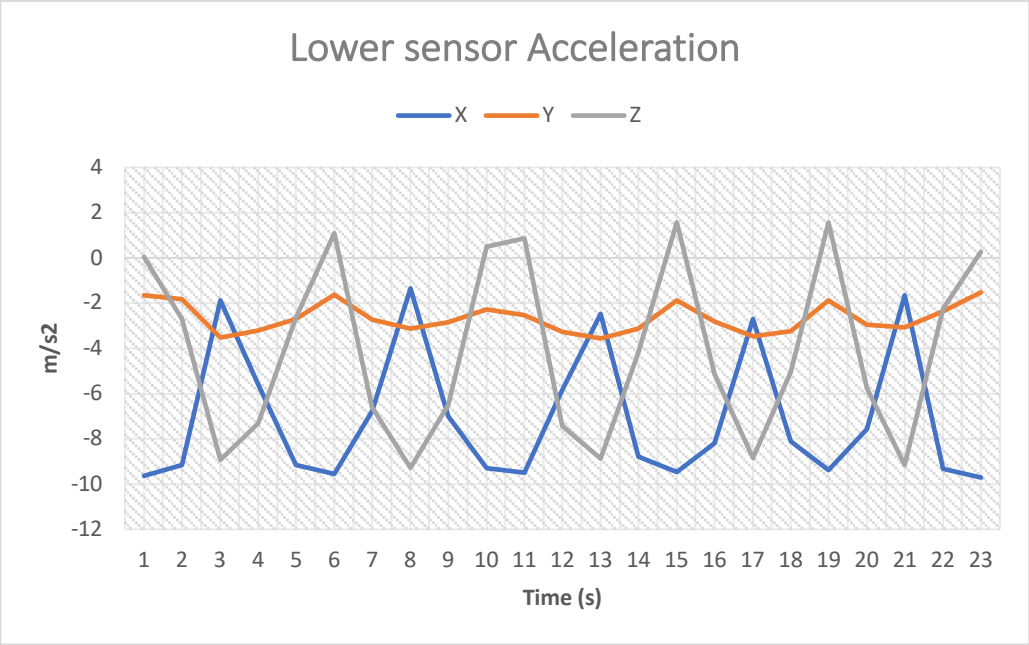

Figure 15: Rate of change of velocity of relative motion for the lower sensor

On the other hand, x-axis shows some changes for the tibial movement, which causes relative changes in the z-axis. In some position, x-axis tries to get z axis and vice versa at the same time position. Where there is no movement on the y axis only rotation for the one directional movement which remains constant.

Accuracy check for Acceleration:

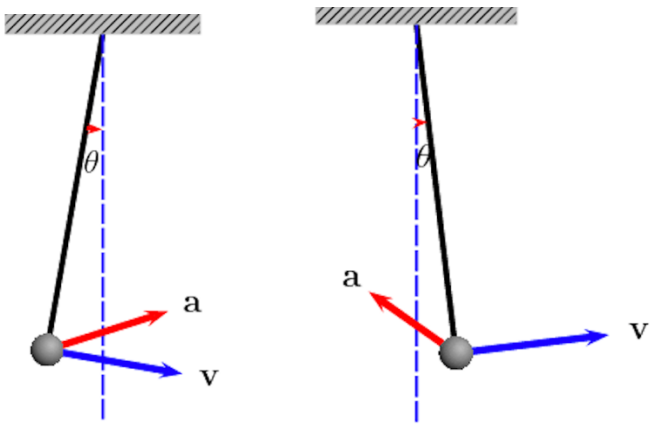

Figure 16: Rate of change of velocity of oscillating pendulum [15]

For one directional movement, rate of change of velocity occurs because of the relative movement of the two axes where another axis remains constant. The increase of x-axis causes the decrease of the z-axis and vice versa. In the figure 15, x-axis moves up because of the relative downward movement of z-axis which validates the results according to figure 16 and gives similar data output comparing to standard output from figure 13. Where there is no directional changes of y axis because its keeps rotating in its own axis, so its gives constant IMU data over the time period.

Angular Velocity

Gyroscope sensor measure the orientation and angular velocity, where how fast the knee moves and changes its angle position with respect to time measured in degree per second.

Table 2: Angular velocity in degree per second for the both sensors

	Upper Sensor (deg/s)				Lower sensor (deg/s)		
Time (s)	X	Y	Z		X	Y	Z
1	-0,004216284	-0,011233986	0,00271406		-0,007427112	0,000917353	-0,003706291
2	-0,075470395	-0,567862511	0,052457195		-0,121385939	1,160041332	-0,161482587
3	-0,174023256	-0,067175575	0,001763739		-0,206023663	0,774009943	-0,110023029
4	0,104656264	-0,008866937	-0,000459328		0,11466606	-0,780587971	0,177081972
5	-0,072894253	0,043563738	-0,01610145		-0,07463745	-0,67109108	-0,00225157
6	-0,08174517	0,137858272	-0,044808913		0,048332125	-0,341645807	0,010634031
7	0,098277442	-0,047063455	-0,058305647		0,116438441	0,628936052	-0,077115513
8	0,048668128	-0,109390914	0,077467218		-0,163494825	0,19232209	0,017263932
9	0,228141278	0,051764037	0,024366755		0,062209398	-0,890549362	0,240164906
10	0,086867638	0,22809343	-0,044622652		-0,096000373	-0,015566872	-0,037734959
11	0,027642589	-0,108941667	0,09853673		-0,047486581	1,230751157	-0,202514052
12	-0,043042377	-0,160054162	0,015999937		-0,021929307	0,340866268	-0,034667939
13	0,025799686	0,090168595	0,034089576		0,066938519	-0,483318597	0,108055338
14	-0,001737103	0,15388079	0,009475837		0,120541178	-0,729034722	0,130523473
15	-0,044715423	0,274477422	-0,087996237		-0,050390169	-0,323690325	0,074596554
16	-0,072942331	-0,325475782	0,005625378		0,153284237	1,237445354	-0,099849239
17	-0,038969148	-0,029533733	-0,00466928		-0,053921357	0,272145778	-0,042320803
18	0,091945574	0,108569294	0,030796418		-0,073969714	-1,04643786	0,132043332
19	0,012741052	-0,012020768	0,003925525		0,031543706	-0,002124533	-0,005245582
20	-0,100384295	-0,176715538	0,003430434		0,08293239	0,661976099	-0,087121665
21	0,040938891	-0,106796645	-0,086636841		0,04522673	-0,393162221	0,053392947
22	0,037392616	0,356394231	0,076198243		0,224345341	-0,834404051	0,091241032
23	-0,004216284	-0,011233986	0,00271406		-0,076211989	0,031607423	0,063447103

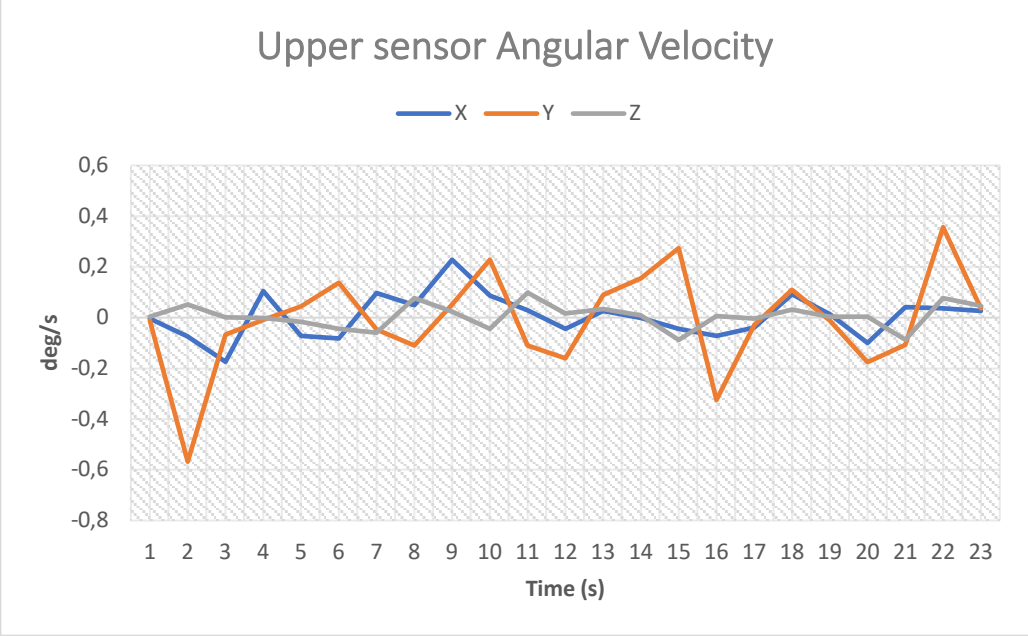

Figure 17: Angular velocity plot of the knee movement for the stationary upper sensor

From our measurement data we plotted angular velocity for both part of knee. From the both sensors it is clear that y-axis revolves according to the knee movement where lower sensor quite faster than upper sensor and that is because of tibial movement.

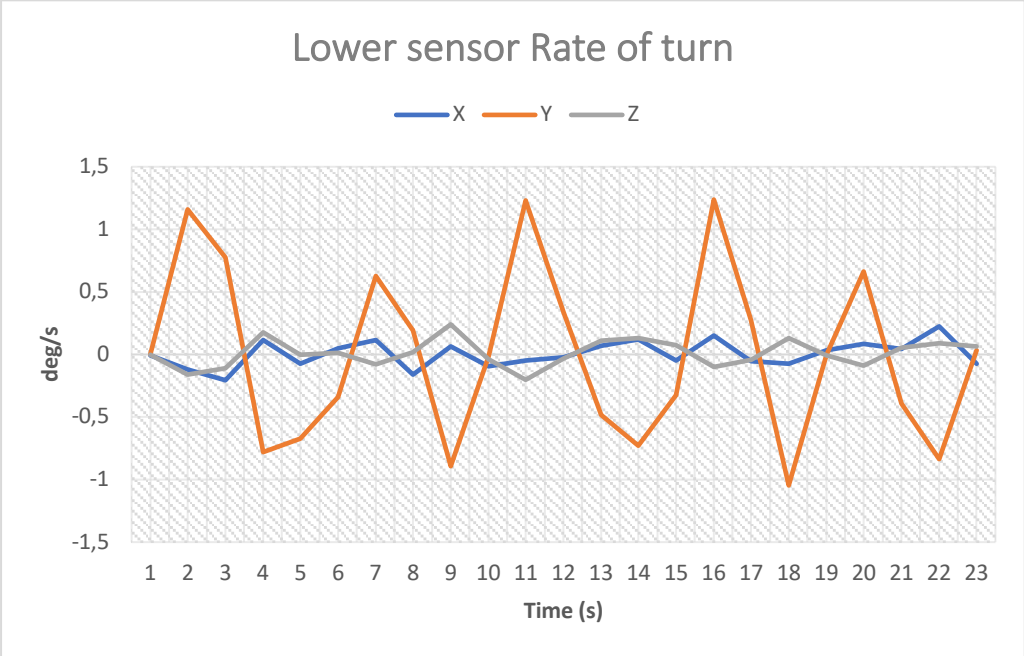

Figure 18: Angular velocity plot of the knee movement for the lower sensor

Accuracy check of Gyroscope:

According to our measured data, there is a spin angular velocity of y-axis because of flexion and extension of knee joint which proves the formula and gives the similar kind of graphical representation comparing with standard output data from figure 13.

Magnetic field

Our IMU system uses xsens MTI-series1 motion sensor which is a MEMS device integrated with Magnetometer sensor can detect and measure magnetic field. Mechanical motion produces magnetic field and measured the strength by the systems are given below during tibial movement.

Table 3: Measurement of magnetic field for every axis

	Upper Sensor (AU)				Lower sensor (AU)		
Time (s)	X	Y	Z		X	Y	Z
1	1,050881386	-0,219652891	-0,053301573		1,110633373	-0,25250864	-0,09614706
2	1,066551924	-0,229391813	-0,163583994		1,084573269	-0,179952621	0,115450144
3	0,979946852	-0,2276721	-0,542315006		0,243028164	-0,056714773	0,933563709
4	0,958115816	-0,249314308	-0,548022032		0,708705664	-0,082001686	0,721254826
5	0,987616777	-0,297688723	-0,460650682		1,05913496	-0,190265894	0,171177149
6	1,031403065	-0,282306671	-0,337555885		1,065992594	-0,248798847	-0,262148142
7	1,006156683	-0,261657238	-0,43081665		0,830579996	-0,096191883	0,62759161
8	0,985358715	-0,226287365	-0,539824486		0,315591335	-0,078784466	0,915380239
9	0,973848104	-0,249331474	-0,512713194		0,832830906	-0,105478525	0,598048925
10	1,01531148	-0,266626358	-0,381108522		1,072120905	-0,233738184	-0,279533625
11	1,043632507	-0,25670886	-0,2809484		1,078557014	-0,214351892	-0,258817196
12	0,998074055	-0,241340876	-0,458890915		0,739435196	-0,081650734	0,702436686
13	0,975968838	-0,230007887	-0,548311234		0,389580488	-0,078896046	0,907711029

14	1,000419617	-0,267638445	-0,45977211		0,98778367	-0,120120049	0,361601591
15	1,026404381	-0,260033846	-0,321280241		1,071913481	-0,229783297	-0,300307751
16	1,021170855	-0,261362553	-0,377568483		0,96293354	-0,118255615	0,437166929
17	0,986746311	-0,215162039	-0,534191847		0,400319338	-0,013077021	0,893182039
18	0,97550416	-0,255252838	-0,506327868		0,967612267	-0,119286299	0,410760164
19	1,031582117	-0,252860308	-0,361545801		1,072302341	-0,223909378	-0,307689667
20	1,000243425	-0,260772943	-0,464654922		0,894380569	-0,096392393	0,520874977
21	0,95238924	-0,227027893	-0,580560207		0,347219467	-0,042292356	0,913001776
22	0,983367443	-0,287794352	-0,472449303		1,074638844	-0,148941994	0,123281717
23	1,046165466	-0,28716588	-0,207668304		1,092815638	-0,232452869	-0,156083584

This IMU measures the magnetic field strength according to the atomic number. Here measured atomic numbers are plotted during the time of movement to estimate the field strength during tibial movement.

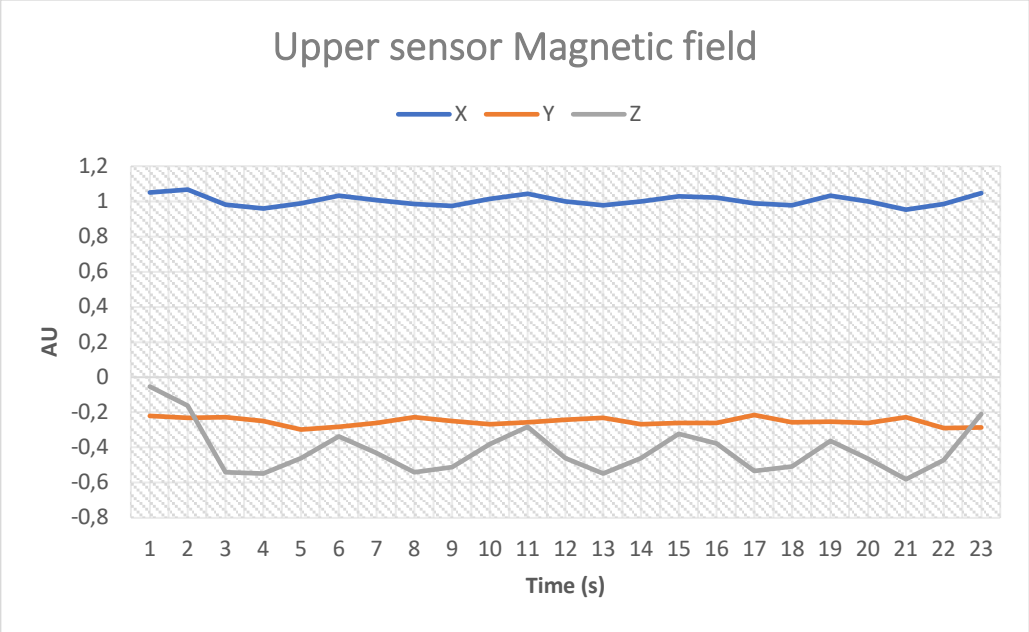

Figure 19: Magnetic field measured by the upper sensor

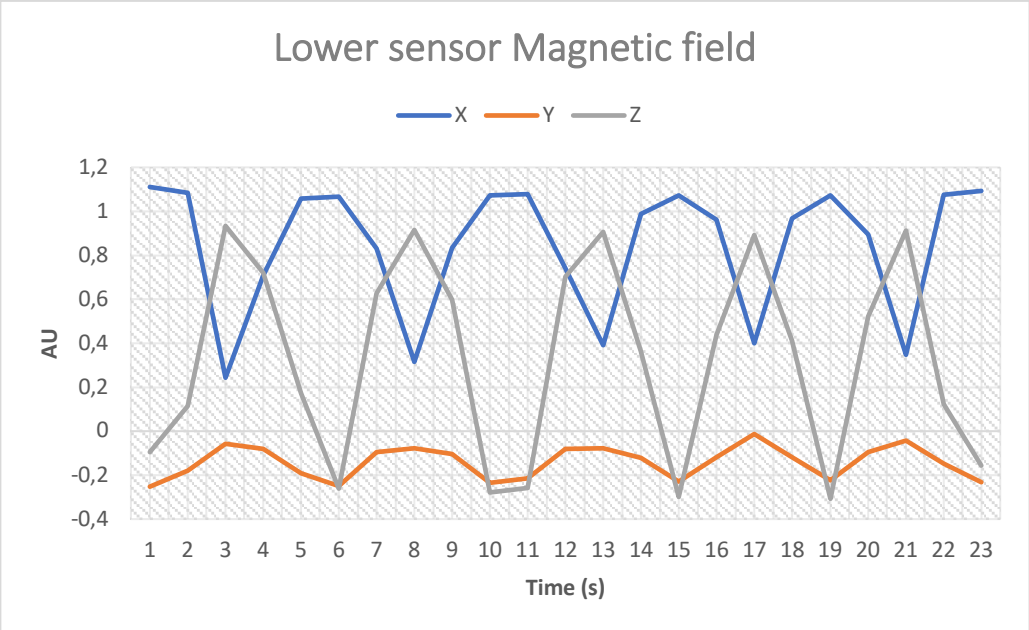

Figure 20: Magnetic field measured by the lower sensor

Accuracy check of Magnetic field strength measurement:

During the tibial flexion and extension almost no movement of femoral part. So, magnetic field strength during this movement almost constant. On the other hand, there is a change in magnetic field strength for the movement of lower knee movement. Figure 20 clearly shows the changes for x and z axis movement changes the strength of magnetic field. SO our IMU system gives the similar data comparing with the reference output from figure 13.

Quaternion

Quaternion is a three-dimensional rotation, that means axis can can rotate in three dimensions around a given force. So, it will have four vectors. SO, it represents axis rotaion for a given pole.

Table 4: Quaternion measurement data for both sensors

	Upper Sensor					Lower sensor			
Time (s)	X	Y	Z	W		X	Y	Z	W
1	0,562452555	0,316357255	0,668759167	-0,369224191		0,696229398	-0,00368107	0,708592772	-0,114662357
2	0,599300385	0,305988282	0,625504792	-0,394910455		0,61152935	-0,012666592	0,778760552	-0,139297009
3	0,717684627	0,198420078	0,544941366	-0,385483086		0,070386298	-0,016436748	0,982660532	-0,170744941
4	0,72393328	0,209261417	0,536456823	-0,379927009		0,316378117	-0,012187055	0,933474541	-0,168468237
5	0,698473692	0,261324286	0,53434515	-0,397894114		0,59751451	0,006125438	0,78811866	-0,147676259
6	0,660917819	0,276064992	0,583784401	-0,3823241		0,742809653	-0,01837562	0,661206543	-0,103450596
7	0,690858901	0,235313132	0,576876879	-0,366817266		0,374092788	-0,034868658	0,913847446	-0,154018417
8	0,720157564	0,192107871	0,554061711	-0,370787561		0,103699192	-0,037692361	0,981475592	-0,156625882
9	0,713435173	0,21814023	0,546085119	-0,381072849		0,39090836	-0,017959343	0,906907558	-0,156162485
10	0,673752844	0,261155248	0,57528317	-0,383281112		0,744251192	-0,029441338	0,659065127	-0,104196057
11	0,644236147	0,270840913	0,613116741	-0,368365377		0,729909837	-0,049229216	0,675502598	-0,092219971
12	0,69848901	0,220198661	0,572422981	-0,368724406		0,321068257	-0,04299302	0,933286846	-0,155056342
13	0,723558784	0,194162861	0,55015862	-0,368902832		0,141685024	-0,038037136	0,976499975	-0,157881767
14	0,698158979	0,237792656	0,556844175	-0,382038206		0,507638514	-0,027800526	0,84798193	-0,149856851
15	0,65358144	0,271431476	0,594609678	-0,381569982		0,75093627	-0,03137017	0,651471198	-0,103422299
16	0,67438674	0,2529338	0,583276689	-0,37552008		0,466634423	-0,041664384	0,871916771	-0,142400131
17	0,714642465	0,194852769	0,551104665	-0,384190261		0,14471589	-0,034691226	0,970710695	-0,188612401
18	0,707923234	0,23210597	0,533267438	-0,400745958		0,488456547	-0,012827993	0,858046532	-0,15812017
19	0,664855659	0,259949356	0,583187401	-0,38766706		0,752505422	-0,027439088	0,647808909	-0,11544013
20	0,700628698	0,231753692	0,556420088	-0,381846219		0,422970176	-0,028947672	0,892237306	-0,155469418
21	0,733607709	0,187409773	0,53310281	-0,377490073		0,121650569	-0,024536248	0,976515055	-0,176117897
22	0,702396452	0,250755161	0,539855421	-0,390278488		0,605644524	-0,022063792	0,781715572	-0,147067323
23	0,623930037	0,303256243	0,622472107	-0,36231941		0,704610527	-0,019625116	0,701198041	-0,107052431

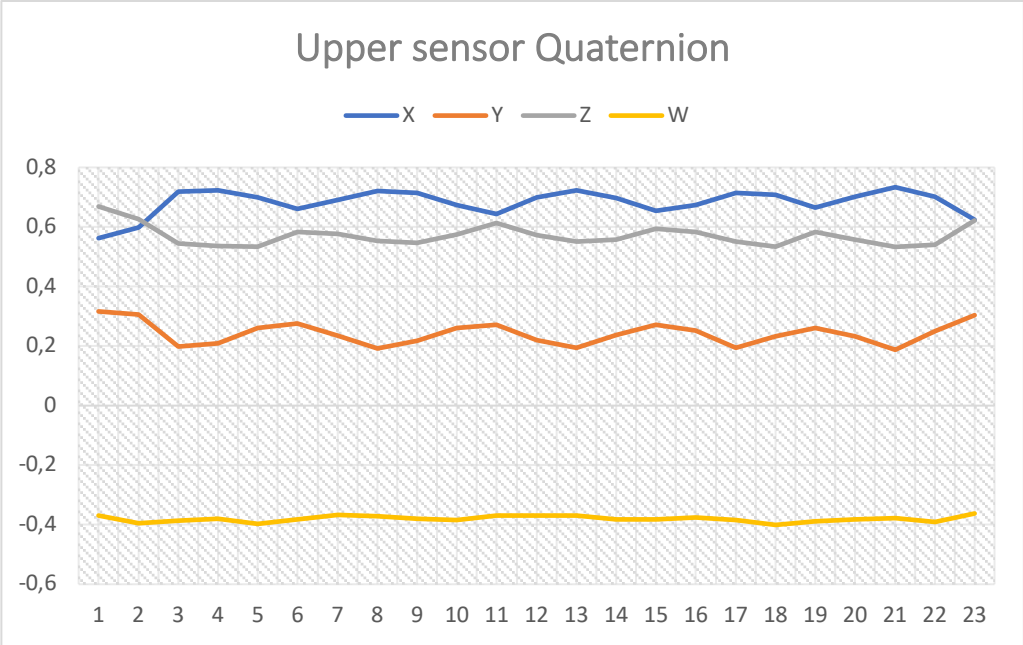

Figure 21: Quaternion data plotted for the sensor

There is no movement in 3-D space because no given pole (no movement of the upper knee joint).

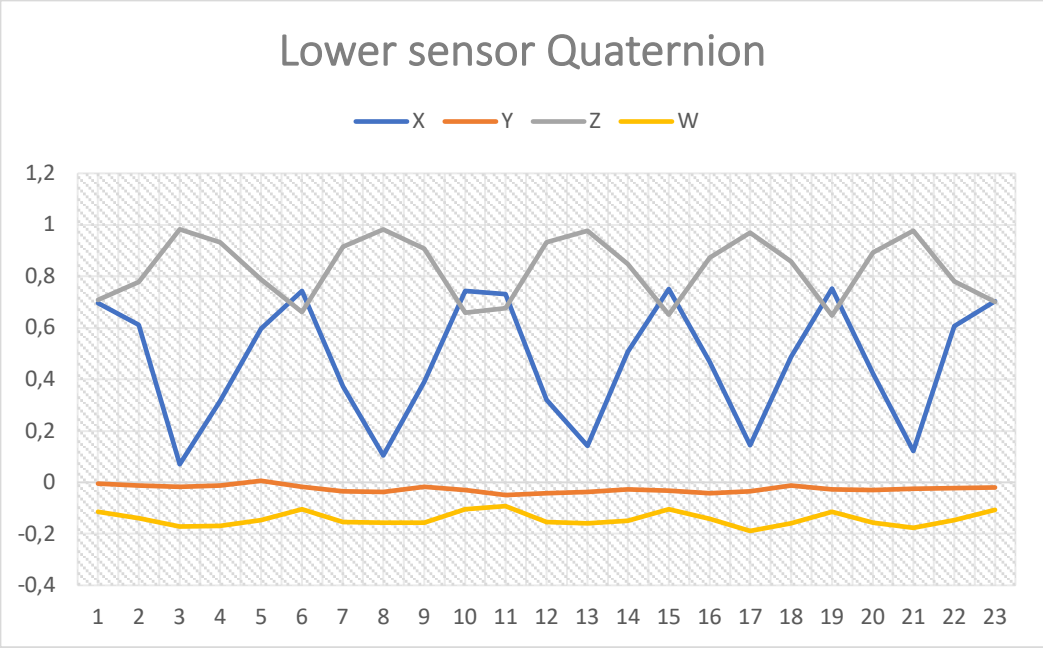

Figure 22: Quaternion data plotted for the lower sensor

Because of the directional flexion and extension, it has only 2-Dimentional movement, one is in the x direction and other in the z direction. And from the tibial movement representation of lower tracking system it shows 2-dimensional movement. Figure 22 shows the rotation in the x and z axis for the tibial knee movement.

Validation

To validate our IMU system output for both upper and lower sensors initially we took into consideration the standard output of the xsens MVN analyzer [16] as a reference. This reference system usually measures and gives the real-time 3D movement for the motion sensor including Accelerometer, Angular velocity, Magnetic field, Quaternion etc. In the following task, we performed the measurement synchronously at a certain amount of time for both the IMU and the reference MVN xsens sensor and plotted the individual experimental data for flexion and extension for both the sensors and compare these plots with the standard output of the sensor which is shown in the figure below.

For this measurement, we set the IMU in a way to get the maximum possible output per second. Here we worked with raspberry pi zero, which performs quietly good at 25Hz, which means it gives 25 data per second which we assumed as the maximum possible outcome. Where our reference system measures data at 240Hz, it means 240 data per second. Though it's quite tough to compare for this big difference, we just took graphical representation into consideration for the validation, because we put these sensors ideally and measured synchronously. So, it must show a similar graphical representation.

On the x-axis, we plotted the number of data measured for the specific time of measurement, because it's quite messy to plot the time axis for the huge amount of data per second. The duration of the measurement for each sensor was around 63 second. For our IMU data representation x-axis shows around 1561 data at 25 Hz which represents the duration of 63 seconds. And for the reference system shows around 15000 data at 240Hz means also represents the same recorded time.

Acceleration

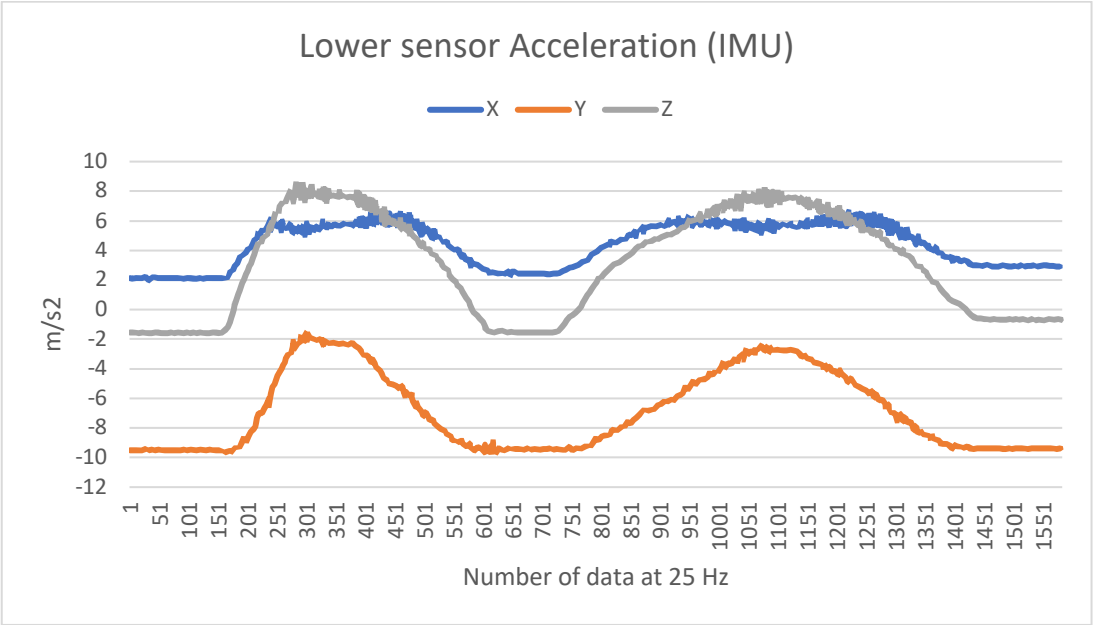

Figure 23: Rate of change of velocity of relative motion for the IMU lower sensor

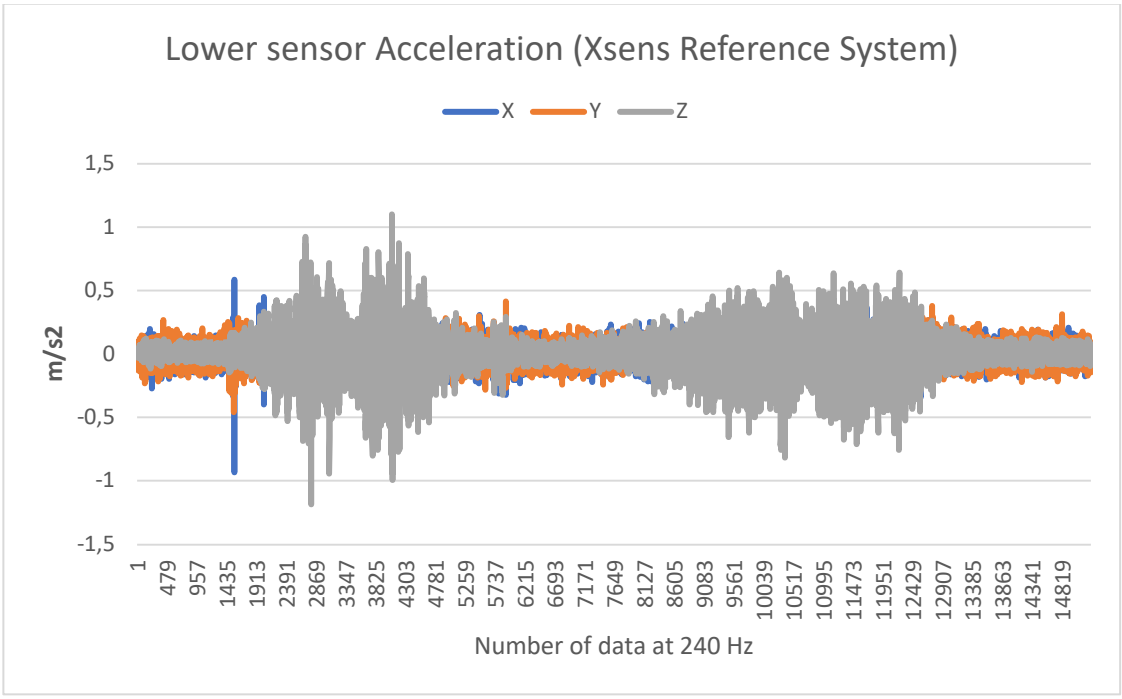

Figure 24: Rate of change of velocity of relative motion for the MVN lower sensor

Lower sensor must show some changes in the axis for the tibial movement though it's difficult to compare between them for the huge data difference per second, its clear from the graph that both sensors shows same graphical representation.

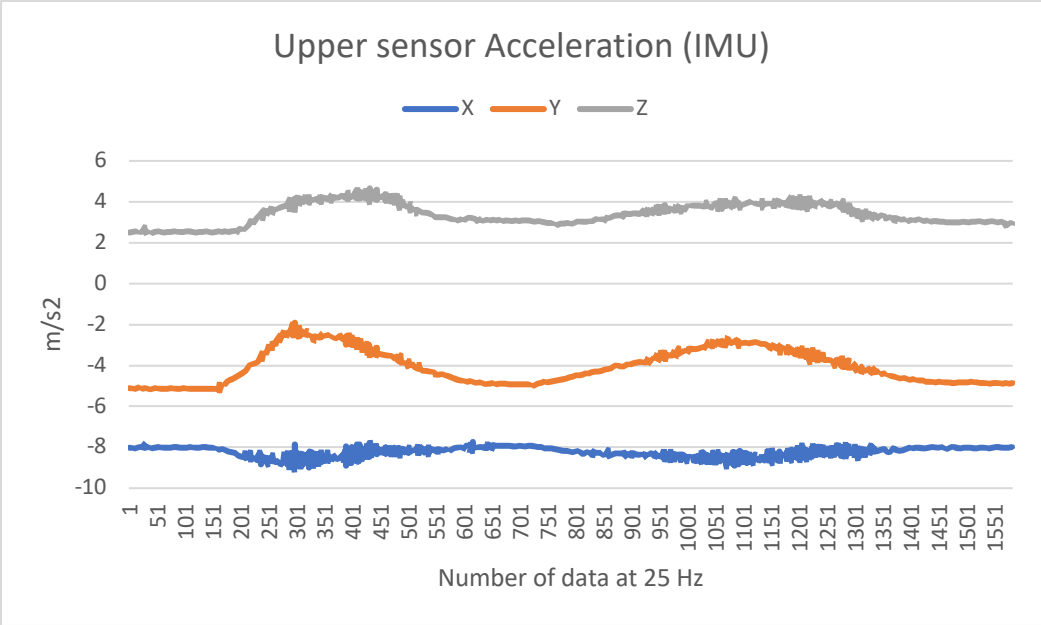

Figure 25: Rate of change of velocity of relative motion for the IMU upper sensor

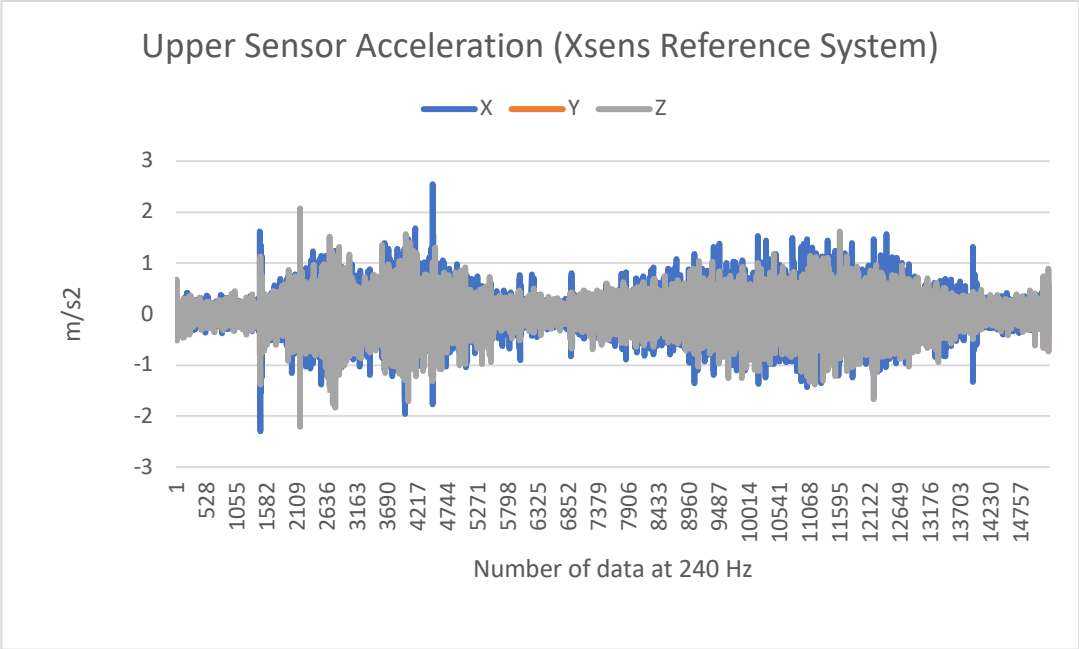

Figure 26: Rate of change of velocity of relative motion for the MVN upper sensor

On the other hand, the is no movement in the upper sensor for the tibial 90° movement, because of the movement the sensor shows some noises for the both cases.

Angular Velocity

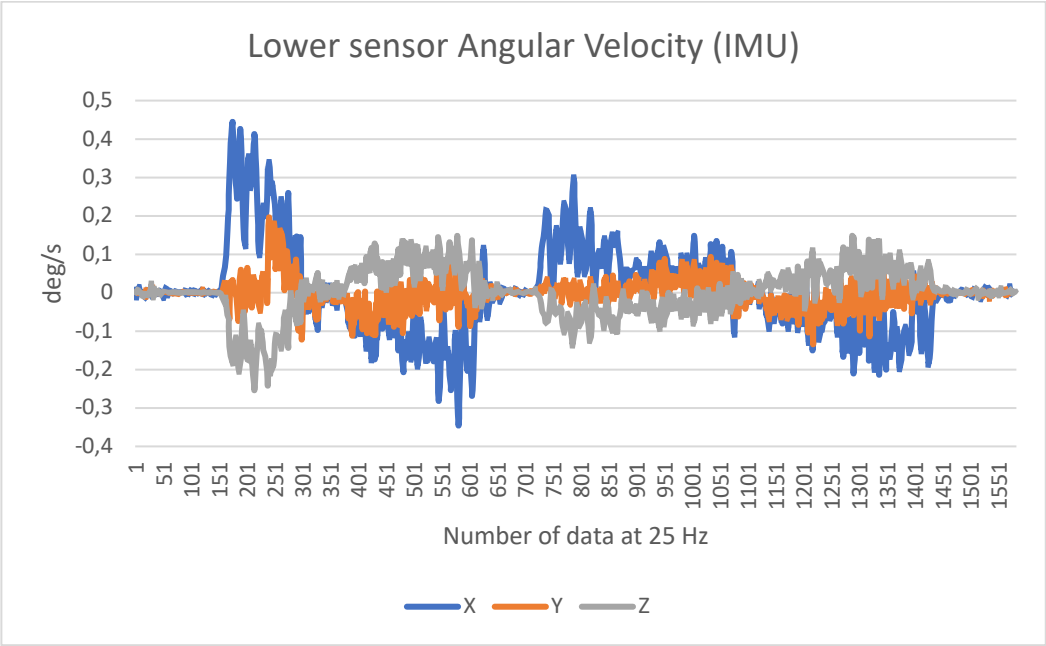

Figure 27: Angular velocity plot of the knee movement for the IMU lower sensor

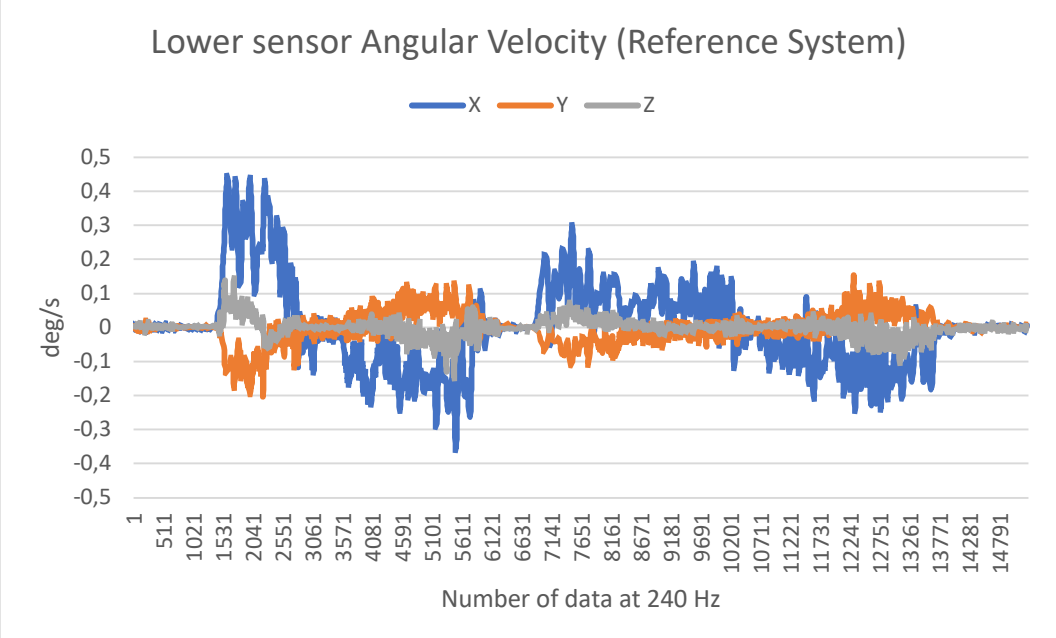

Figure 28: Angular velocity plot of the knee movement for the MVN lower sensor

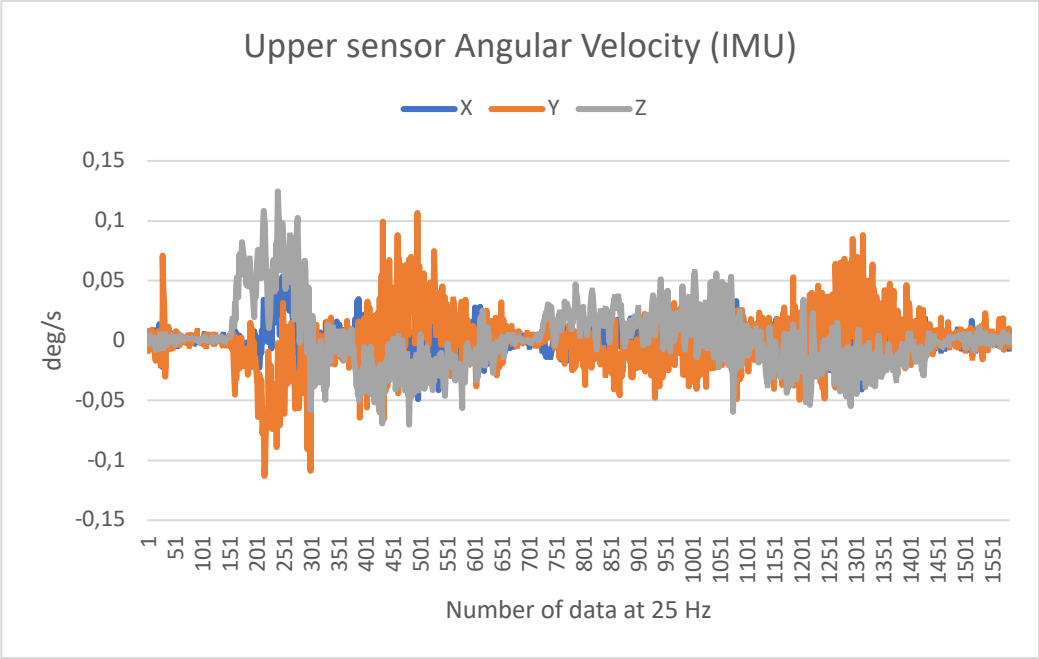

Figure 29: Angular velocity plot of the knee movement for the IMU upper sensor

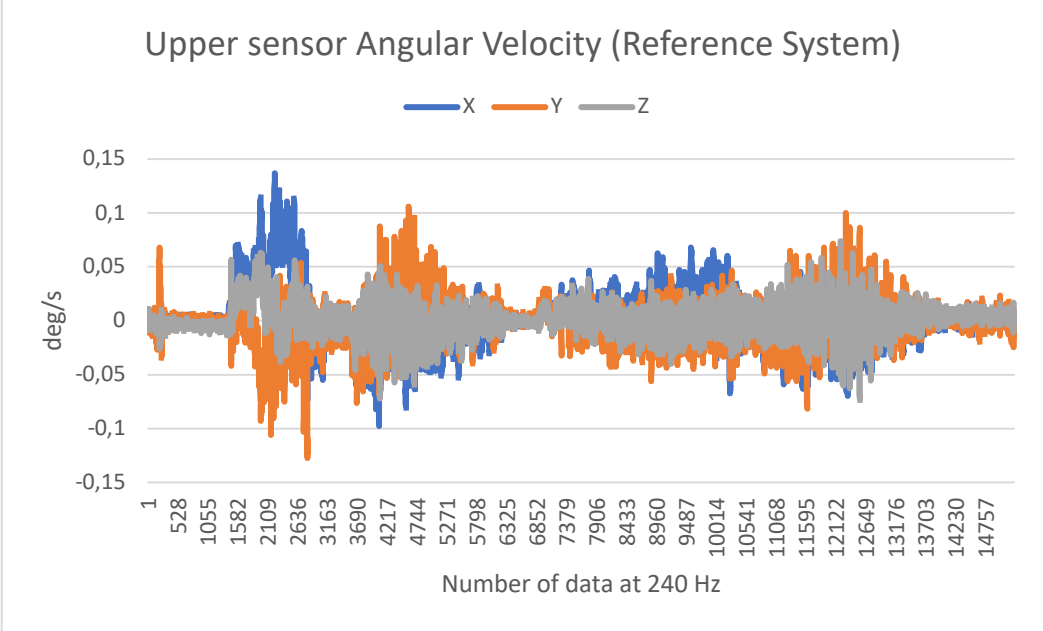

Figure 30: Angular velocity plot of the knee movement for the MVN upper sensor

For both the upper and lower system IMU shows almost the same representation as the MVN analyzer which considers as a reference system for the specific measurement.

Magnetic field

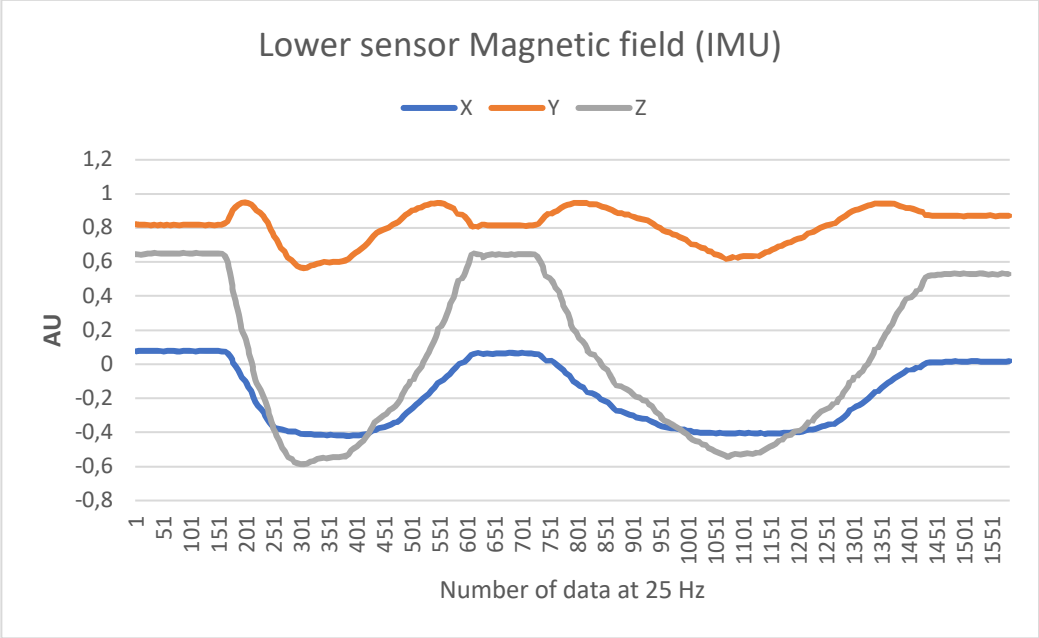

Figure 31: Magnetic field measured by the IMU lower sensor

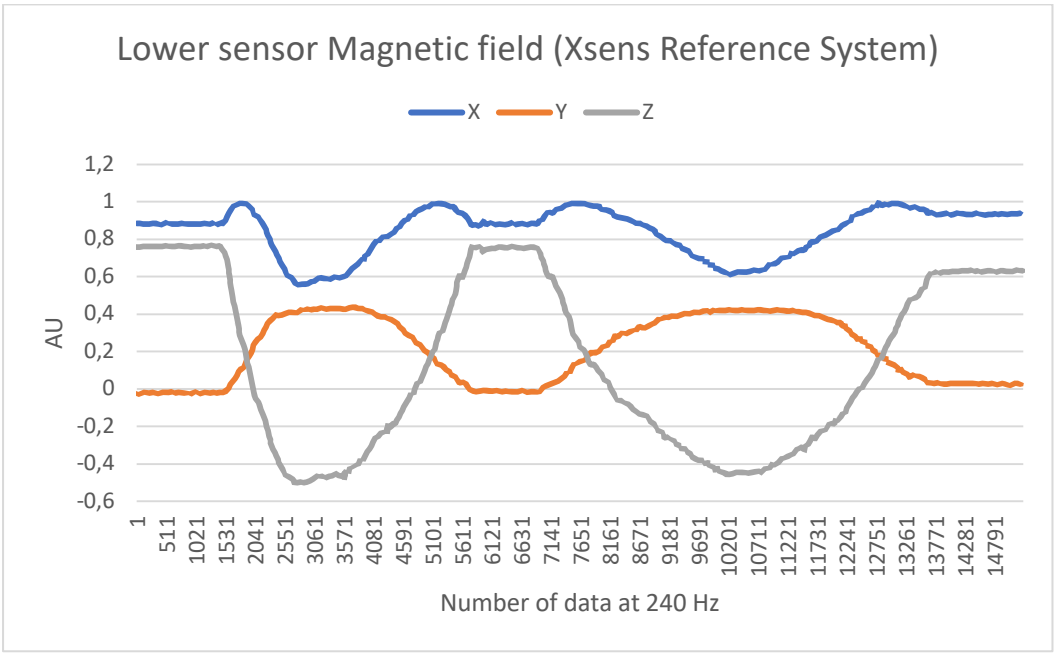

Figure 32: Magnetic field measured by the MVN lower sensor

Here also for the tibial movement our IMU lower shows the same magnetic field changes as the reference MVN analyzer does.

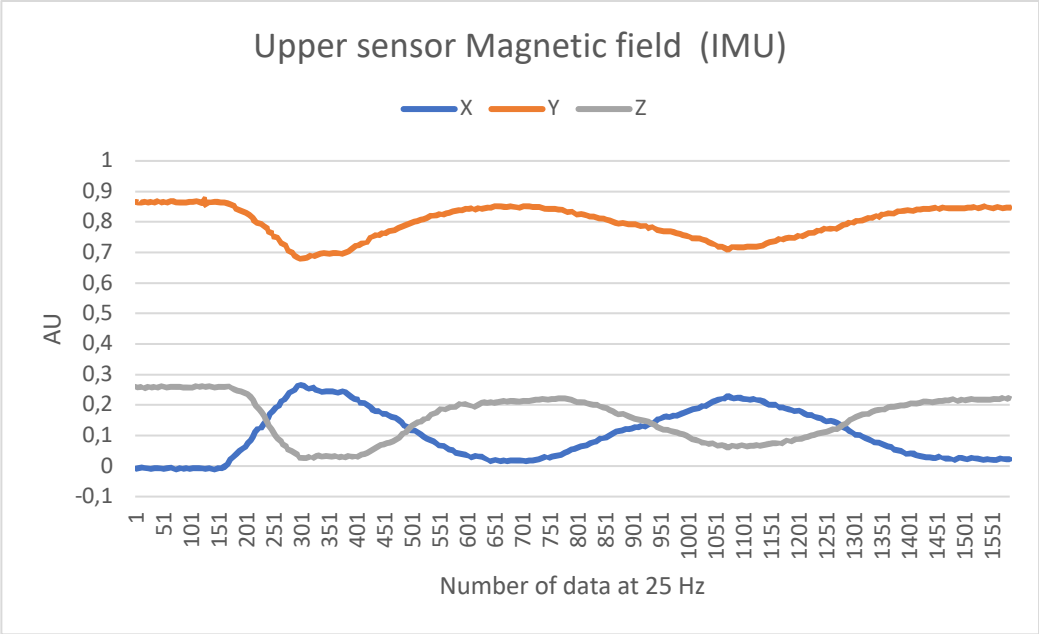

Figure 33: Magnetic field measured by the IMU upper sensor

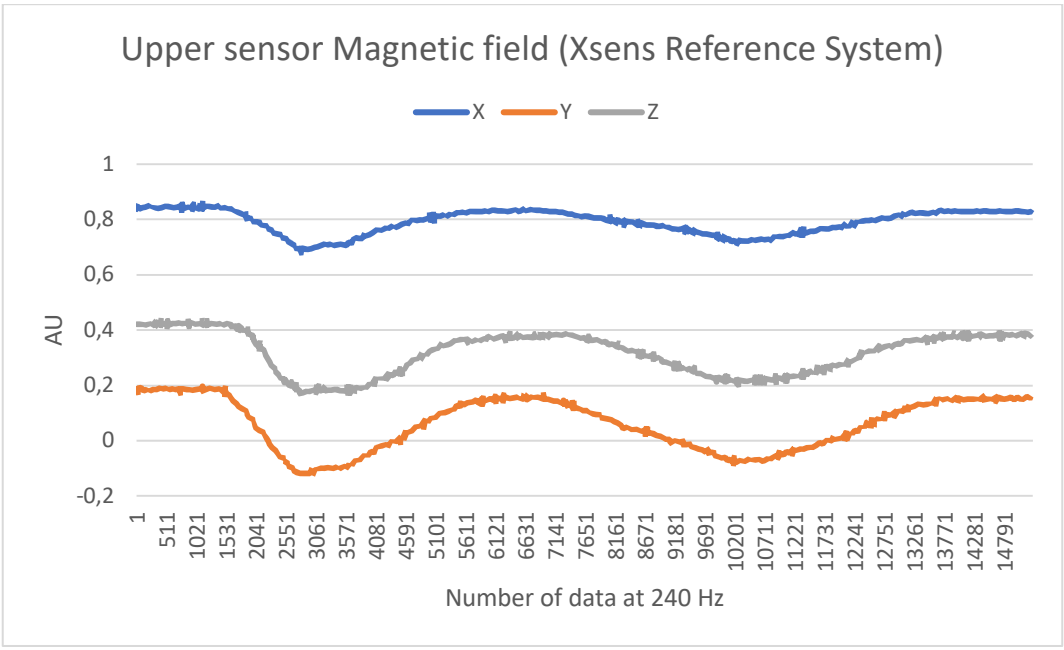

Figure 34: Magnetic field measured by the MVN upper sensor

IMU upper sensor shows almost no changes in the magnetic field for this specific movement and showing the same representation for the reference system it validates our result.

Discussion and Conclusion

In this project we developed an inertial measurement unit for the knee joint motion estimation. For this development we choose two xsens sensors for the motion tracking and measurement. Sensors are interfaced with raspberry pi for i2c commutation. We also developed a python code to communicate with sensors to track and measure knee joint motion data and described the communication procedure by some flow charts according to the xsens datasheet. Integration of hardware are designed in such a way to perform practically with the switches and LED notifications. After the PCB design we integrated the hardware and made it prepare for knee joint motion measurement. There are three different possibilities to perform and measure knee motion because three different ways of movement. Here we measured the knee motion only for flexion-extension of knee movement. After that we plotted the IMU data and performed validation check. According to the movement of knee sensors gives accurate data. By comparing data between two sensors, knee joint motion tracking is performed which is useful for medical diagnoses purpose. Future prospects of this IMU device to use as automated monitoring tool such as activity level of knee, the status of knee health and the intensity of training.

Future work

Evaluation of the system for kinemetric estimation:

- Appropriate calibration method proposes
- Final evaluation of accuracy

References

1. Arthritis Foundation . (n.d.). *Arthritis Foundation* . Retrieved from arthritis.org: https://www.arthritis.org/about-arthritis/where-it-hurts/knee-pain/diagnosis/knee-mri.php

2. W. G. Kernohan, D. E. (1990). Vibration arthrometry. *cta Orthopaedica Scandinavica*, 70–79.

3. C. Cescon, P. M. (2008). Longitudinal and transverse propagation of surface mechanomyographic waves generated by single motor unit activity. *Medical & biological engineering & computing,*, 871.

4. Rangayyan, Y. T. (1994). Adaptive cancellation of muscle contraction inter- ference in vibroarthrographic signals. *IEEE Transactions on Biomedical Engineering*, 181–191.

5. Bachmann, E., Yun, X., & Brumfield, A. (2007). Limitations of Attitude Estimation Algorithms for Inertial/Magnetic Sensor Modules. *IEEE Robot. Autom. Mag.*, 76–87.

6. Xsens. Retrieved from https://www.xsens.com/products/mti-1-series/

7. Fields, K. B. (2011). Running injuries-changing trends and demographics. *Current sports medicine reports*, 299–303.

8. Kapur, R. A. (2016). Acoustic emission in orthopaedics: A state of the art review. *ournal of biomechanics*, 4065–4072.

9. Aleksandra Lobnik, M. T. (n.d.). *Optical Chemical Sensors: Design and Applications* . Slovenia : Universitiy of Maribor, Faculty of Mechanical Engineering, Institute for Environmental Protection and Sensors,.

10. Vijay Laxmi Kalyani, V. S. (October 2016). Optical Sensors And Their Use In Medical Field.

11. injurymap. (2019, Aug. 12). Retrieved from https://www.injurymap.com/diagnoses/osteoarthritis-of-the-knee

12. Retrieved from https://www.slideshare.net/StephanvanBreenenCli/occupational-therapy-goniometry-measurement-range-of-motion

13. Doctor. (n.d.). Retrieved from https://doctorlib.info/anatomy/classic-human-anatomy-motion/3.html

14. Retrieved from https://boneandspine.com/knee-range-of-motion/

15. https://commons.wikimedia.org/wiki/File:Oscillating_pendulum.gif#/media/File:Oscillating_pendulum.gif

16. xsens. *MVN Analyze*. Retrieved from xsens: https://www.xsens.com/products/mvn-analyze

Appendix

Python code:

```
# I2C communication between pi and xsens & measuring live motion data using Push button and LED
#
# References:
# https://cdn2.hubspot.net/hubfs/3446270/Downloads/Manuals/Hardware_Integration_Manual_MTi_1-series.pdf                    (link:1)
# https://cdn2.hubspot.net/hubfs/3446270/Downloads/Leaflets/MTi-1-series-datasheet.pdf                                    (link:2)
# https://1drv.ms/b/s!AqtCaRJTR76vymVlSyjQYkbMKXnQ?e=ob6zoP         (link:3)
# https://base.xsens.com/hc/en-us/articles/115002856865-Best-practices-I2C-and-SPI-for-MTi-1-series?mobile_site=true        (link:4)
# https://1drv.ms/b/s!AqtCaRJTR76vsw1RJycZP2z9RCGO?e=eQnXgK         (link:5)
#
# Max i2c speed of pi 400Kbps. We want to read values with 100hz (by default sensor give values 100 times in 1 seconds), then we need max speed for i2c = 400Kbps
# We need setup raspberry to use i2c with speed 400 Kbps.
#  1. open lxterminal
#  2. type: sudo nano /boot/config.txt
#  3. add line: i2c_arm_baudrate=400000
#  4. save and close file: ctrl+x, y, enter
#
# To run this program type in lxterminal: "sudo python i2c.py"
# Values from sensor will be saved in a file: "/home/pi/data.txt". Use to see values next command: "tail -f /home/pi/data.txt"

import RPi.GPIO  as GPIO        # general purpose input output (Switch: pin 10 (gpio-15) to switch and switch to 240 ohm to 3.3v / LED: pin 12 (gpio-18) to 120 ohm to LED and other end to ground)
import time                     # this is for make dalay (sleep)
import io
import fcntl                    # This module performs file control and I/O control on file descriptors.
import sys
import struct                   # for make float from 4 byte
import os
from datetime import datetime   # we will write time to file
from subprocess import call

ADR1        =   0x6b;                       # address of sensors in i2c bus
ADR2        =   0x29;
I2C_SLAVE   =   0x0703;                 # it is hardcoded in unix system constant. the file used like i2c device in slave mode. (this used in ioctl. from linux/i2c-dev.h)

OPCODE_CONTROLPIPE      =   0x03;  # Used to send control messages to the module (link:5, Page:11)
OPCODE_PIPESTATUS       =   0x04;  # Provides status information for the read pipes (link:5, Page:11)
OPCODE_NOTIFICATIONPIPE =   0x05;  # Used to read non-measurement data: errors acknowledgements and other notifications from the module (link:5, Page:11)
OPCODE_MEASUREMENTPIPE  =   0x06;  # All measurement data generated by the module will be available in the measurement pipe (link:5, Page:11)

MID_GOTOCONFIG          =   0x30;  # MessageID: GoToConfig (link:3, Page:11)
MID_GOTOMEASUREMENT     =   0x10;  # MessageID: GoToMeasurement (link:3, Page:11)
MID_RESET               =   0x40;  #  (link:3, Page:56)
MID_SETCONFIG           =   0xC0;  # Set the current output configuration (link:3, Page:23)
MTData2                 =   54     # (0x36) # Message with one or more output data packets (link:3, Page:59)

PacketCounter           =   0x1020 # (link:3, Page:52)
SampleTimeFine          =   0x1060 # (link:3, Page:52)
Acceleration            =   0x4020 # (link:3, Page:52)
Rate_of_Turn            =   0x8020 # (link:3, Page:52)
Magnetic_Field          =   0xC020 # (link:3, Page:52)
```

```
Quaternion              =   0x2010 # (link:3, Page:52)
Statusword              =   0xE020 # (link:3, Page:52)


##----------
Creating i2c class to read, write, create, close (say to the system that this file is a i2c device) --------
class i2c:                      # create class for working i2c, we work with it like with file, because library SMBus can not read and write array more than 32 bytes, but we need read and write array more than 32 bytes
    def __init__(self, device, bus):
        self.fr = io.open("/dev/i2c-"+str(bus), "rb", buffering=0) # create description for read
        self.fw = io.open("/dev/i2c-"+str(bus), "wb", buffering=0) # create description for write
        fcntl.ioctl(self.fr, I2C_SLAVE, device) # speak that it I2C SLAVE device
        fcntl.ioctl(self.fw, I2C_SLAVE, device) # speak that it I2C SLAVE device
        time.sleep(1)
    def read(self, count):
        return self.fr.read(count)  # count bytes from i2c file
    def write(self, data):
        if type(data) is list:
            data = bytearray(data)
        elif type(data) is str:
            data = _b(data)
        self.fw.write(data)         # read data from i2c file
    def close(self):
        self.fr.close()             # close two i2c files
        self.fw.close()
#--------------------------------------------------------------------------------


##-----------Initialization and configuration----------------------------------------
def sensor_init(d):
        # First: we send to sensor is 'RESET'
    d.write([OPCODE_CONTROLPIPE, MID_RESET, 0, 193])    # Send "reset (0x40)" to ControlPipe. 0 - lenght of parameters. 193 - checksum for this message ....
        # i2c always use address opcode instead BID. when we use write function first byte set from opcode

        # see checksum_calculation code for getting checksum for specific message
        time.sleep(1);              # mti-1 needs time after reset, wait 1 secomd
        # Second: waiting for WakeUp. in this loop we must read notifications from sensor. when we catch 62 (0x3E) (WakeUp) - it means that sensor was reset and boot
    for i in range(300):        # 300 in the loop and in the end time sleep 0.1s (300*0.1 =30s). Max time to wake up for the loop will be 30s.
                                # But we really go out from this loop faster when catch WakeUp and then break. SO any values can be used instead 300
            pipes_len=0     # lenght for notification
            pipem_len=0     # lenght for measures
            try:
                    d.write([OPCODE_PIPESTATUS])        # we work with PipeStatus(4) to know how many bytes we must read (link:5, Page:11)
                    s = d.read(4)   # now we read 4 bytes to know lenght of massages.
                    t = bytearray()     # create new empy array of bytes
                    t.extend(map(ord, s)) # S is string, byt we need bytes
                    pipes_len=t[0]+(t[1]*256); # 1st and 2nd bytes are lenght of  NotificationPipe
                    pipem_len=t[2]+(t[3]*256); # 3th and 4th bytes are lenght of  MeasurementPipe
                    d.write([OPCODE_NOTIFICATIONPIPE])      # we work with NotificationPipe(5) (link:5, Page:11)
                    s = d.read(pipes_len); # read NotificationPipe. S is string, byt we need bytes
                    t = bytearray() # create new emtpy array of bytes
                    t.extend(map(ord, s))   # making it to bytes
                    print(hex(t[0]))    # print MID from notification
            if (t[0]==0x41):
                print("ResetAck catched")
                    if (t[0]==62):      # if it 0x3E = WakeUp (link:3, Page:56). then go out from this loop

                        print("WakeUp catched")
                        break;
                    if (t[0]==0x42):
                print("error: "+hex(t[1])+" "+hex(t[2]))
```

```
106.                 d.write([OPCODE_CONTROLPIPE, MID_RESET, 0, 193])
107.                 time.sleep(0.5);
108.                 d.write([OPCODE_CONTROLPIPE,MID_GOTOCONFIG,0,209])
109.             if (t[0]==0x31):
110.                 print("ConfigModeAck catched")
111.                 break;
112.             #dev.write([OPCODE_CONTROLPIPE ,63,0,194]) # wakeupAck = 63 (0x3F) (link:3, Page:56)
113.             #d.write([OPCODE_MEASUREMENTPIPE])      # we work with MeasurementPipe(6) (link:5, Page:11)
114.                     #s=d.read(pipem_len) # read MeasurementPipe, but we need this Measure, just read
115.             except:
116.                     print("E")
117.             time.sleep(0.1)
118.         time.sleep(0.2)
119.         # Third: Put sensor to Configure state
120.     d.write([OPCODE_CONTROLPIPE,MID_GOTOCONFIG,0,209])  # goto Config mode
121.         time.sleep(0.2)
122.         # Fourth: Set configuration for sensor: Acc, Gyr, Mag and Quaternion refresh 1 time in 1 second (lin
     k:3, Page:52)
123.     d.write([OPCODE_CONTROLPIPE,MID_SETCONFIG,28,0x10,0x20,0xff,0xff,0x10,0x60,0xff,0xff,0x40,0x20,0,1,0x80,
     0x20,0,1,0xc0,0x20,0,1,0x20,0x10,0,1,0xe0,0x20,0xff,0xff,119])
124.         time.sleep(0.2)
125. #-------------------------------------------------------------------------------------
126.
127. ##----------Sensor Get Measure-----------------------------------------------------------
128. def sensor_get_measure(dev, num, write):
129.     s="";
130.     pipes_len=0
131.     pipem_len=0
132.     try:
133.         dev.write([OPCODE_PIPESTATUS])      # we work with PipeStatus(4) to know how many bytes we must read
      (3th link page 11)
134.         s = dev.read(4)     # now we read 4 bytes to know lenght of massages.
135.         t = bytearray() # create new emtpy array of bytes
136.         t.extend(map(ord, s)) # making it to bytes
137.         #print("\r\n4. PipeStatus: %x %x   %x %x" %(t[0], t[1], t[2], t[3]));
138.         pipes_len=t[0]+(t[1]*256); # 2 bytes are the length of the notification pipe
139.         pipem_len=t[2]+(t[3]*256); # 2 bytes are the length of measurement
140.     except:
141.         if (write == 1):
142.                     print("error")
143.     if (pipes_len>0):
144.         if (write == 1):
145.                 print "\r\n5. Notification: ",
146.         dev.write([OPCODE_NOTIFICATIONPIPE])                # we work with NotificationPipe(5) (link:5, Page
     :11)
147.         s = dev.read(pipes_len);    # read NotificationPipe. S is string, byt we need bytes
148.         t = bytearray() # create new emtpy array of bytes
149.         t.extend(map(ord, s)) # making it to bytes
150.         for i in range(pipes_len):  # print Notification
151.             if (write == 1):
152.                         print hex(t[i]),
153.     if (pipem_len>0):
154.         dev.write([OPCODE_MEASUREMENTPIPE])             # we work with MeasurementPipe(6) (link:5, Page:11)

155.         s =  dev.read(pipem_len)    # read MeasurementPipe. S is string, byt we need bytes
156.         t = bytearray() # create new emtpy array of bytes
157.         t.extend(map(ord, s)) # making it to bytes
158.         sum=0
159.         for i in range(pipem_len):  # make checkSum of message
160.             sum=sum+t[i];
161.         if ((sum%256)==1):          # and check it
162.             if (write == 1):
163.                         print("CheckSum is correct")
164.         if (t[0]==0x36):            # MtData2 is MID 0x36, we need it
165.             if (write == 1):
166.                         #print(chr(27) + "[2J") # clear output
167.                     print("\r\nWe have MtData2 message")
168.                         #zf=open("/home/pi/data("+ts+").txt", 'a')  # open file to save values
```

```
                        #zf.write(num+str(datetime.now())+"\r\n")# write data and time to file
            #-------------------
            k=2; # k is our position of bytes in message
            while (k<t[1]): # read byte after byte if it more than lenght in message
                xdi=t[k]*256+t[k+1];    # make XDI from 2 bytes
                k=k+1;
                if (xdi == 0x1020):     #  0x1020 = PacketCounter
                    cnt=t[k+2]*256+t[k+3]
                    if (write == 1):
                                    print("PacketCounter = "+str(cnt))
                    k=k+4
                elif (xdi == 0x1060):   #  0x1060 = Sample time fine
                    if (write == 1):
                                    print("Sample time fine")
                    k=k+6
                elif (xdi == 0x2010):   #  0x2010 = Quaternion
                    k=k+1
                            dd1 = bytearray([t[k+1], t[k+2], t[k+3], t[k+4]])
                                d1= struct.unpack('>f', dd1) # make float from bytes array
                    dd2 = bytearray([t[k+5], t[k+6], t[k+7], t[k+8]])
                                d2= struct.unpack('>f', dd2) # make float from bytes array
                    dd3 = bytearray([t[k+9], t[k+10], t[k+11], t[k+12]])
                                d3= struct.unpack('>f', dd3) # make float from bytes array
                    dd4 = bytearray([t[k+13], t[k+14], t[k+15], t[k+16]])
                    d4 = struct.unpack('>f', dd4)
                    if (write == 1):
                                    print(num+"Quaternion: "+str(d1)+" "+str(d2)+" "+str(d3)+" "+str(d4)) #
    print to output
                                    #zf.write(num+"Quaternion: "+str(d1)+"   "+str(d2)+"   "+str(d3)+" "+str
    (d4)+"\r\n")       # print to file
                    csvQ=str(d1)+str(d2)+str(d3)+str(d4);
                    csvQ=csvQ.replace("(","")
                    csvQ=csvQ.replace(")","")
                    k=k+17
                elif (xdi == 0x8030):   #  0x8030 = DeltaQ
                    if (write == 1):
                                    print("DeltaQ: not need")
                    k=k+18
                elif (xdi == 0xe020):   #  0xe020 = Status word (4 bytes)
                    w=t[k+2]*256*256*256+t[k+3]*256*256+t[k+4]*256+t[k+5];
                    if (write == 1):
                                    print("Status word: "+hex(w))
                    k=k+6
                    #if (write == 1):
                                    #zf.close()             # close file because it is last field
                elif (xdi == 0xc020):   #  0xc020 = Magnetic
                    k=k+1
                    dd1 = bytearray([t[k+1], t[k+2], t[k+3], t[k+4]])
                    d1= struct.unpack('>f', dd1)     # make float from bytes array
                    dd2 = bytearray([t[k+5], t[k+6], t[k+7], t[k+8]])
                    d2= struct.unpack('>f', dd2) # make float from bytes array
                    dd3 = bytearray([t[k+9], t[k+10], t[k+11], t[k+12]])
                    d3= struct.unpack('>f', dd3)     # make float from bytes array
                    if (write == 1):
                                    print(num+"Magnetic: "+str(d1)+"   "+str(d2)+"   "+str(d3))
    # print to output
                                    #zf.write(num+"Magnetic: "+str(d1)+"   "+str(d2)+"   "+str(d3)+"\r\n")
    # print to file
                                csvM=str(d1)+str(d2)+str(d3);
                                        csvM=csvM.replace("(","")
                                        csvM=csvM.replace(")","")
                    k=k+13
                elif (xdi == 0x4020):   #  0x4020 = Acceleration
                    k=k+1
                    dd1 = bytearray([t[k+1], t[k+2], t[k+3], t[k+4]])
                    d1= struct.unpack('>f', dd1)     # make float from bytes array
                    dd2 = bytearray([t[k+5], t[k+6], t[k+7], t[k+8]])
                    d2= struct.unpack('>f', dd2)     # make float from bytes array
```

```
                    dd3 = bytearray([t[k+9], t[k+10], t[k+11], t[k+12]])
                    d3= struct.unpack('>f', dd3)     # make float from bytes array
                    if (write == 1):
                                    print(num+"Acceleration: "+str(d1)+" "+str(d2)+" "+str(d3))
     # print to output
                                    #zf.write(num+"Acceleration: "+str(d1)+"   "+str(d2)+"   "+str(d3)+"\r\n
    ")# print to file
                            csvA=str(d1)+str(d2)+str(d3);
                                    csvA=csvA.replace("(","")
                                    csvA=csvA.replace(")","")
                    k=k+13
                elif (xdi == 0x8020):   #  0x8020 = RateOfTurn
                    k=k+1
                    dd1 = bytearray([t[k+1], t[k+2], t[k+3], t[k+4]])
                    d1= struct.unpack('>f', dd1)     # make float from bytes array
                    dd2 = bytearray([t[k+5], t[k+6], t[k+7], t[k+8]])
                    d2= struct.unpack('>f', dd2)     # make float from bytes array
                    dd3 = bytearray([t[k+9], t[k+10], t[k+11], t[k+12]])
                    d3= struct.unpack('>f', dd3)     # make float from bytes array
                    if (write == 1):
                                    print(num+"RateOfTurn: "+str(d1)+" "+str(d2)+" "+str(d3))
    # print to output
                                    #zf.write(num+"RateOfTurn: "+str(d1)+"   "+str(d2)+"   "+str(d3)+"\r\n")
     # print to file
                    csvG=str(d1)+str(d2)+str(d3);
                                    csvG=csvG.replace("(","")
                                    csvG=csvG.replace(")","")
                    k=k+13
                elif (xdi == 0x4010):   #  0x4010 = DeltaV
                    if (write == 1):
                                    print("DeltaV: not need")
                    k=k+14
                else:
                    if (write == 1):
                                    print("unknown xdi: "+hex(xdi))
            return (""+csvA+csvG+csvM+csvQ)
    return "";
#-------------------------------------------------------------------------------------

#--------init LED and BUTTON-----------------------------------------------------------
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(29, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(38, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(40, GPIO.OUT)
GPIO.setup(31, GPIO.OUT)
need_wr = 0;
time_for_power_off=15;
GPIO.output(31, GPIO.LOW)
##----------open i2c bus-----------------------------------------------------------------
dev1 = i2c(ADR1,1)
dev2 = i2c(ADR2,1)
sensor_init(dev1)
sensor_init(dev2)
#-------------------------------------------------------------------------------------------
##----------Fifth: Put sensor to Measure mode ---------------------------------------------
dev1.write([OPCODE_CONTROLPIPE,MID_GOTOMEASUREMENT,0,241])  # goto measure mode for sensor1
dev2.write([OPCODE_CONTROLPIPE,MID_GOTOMEASUREMENT,0,241])  # goto measure mode for sensor2
#--------ready to write-----------------------------------------------------------------
GPIO.output(40, GPIO.HIGH)
print("-------------------STOP MODE-------------------")
#infinity loop
while True:
    if (GPIO.input(29) == GPIO.HIGH):
            if (need_wr == 0):
                    print("-------------------START MODE-------------------")
                    need_wr = 1;
                    GPIO.output(31, GPIO.HIGH)
```

```
            ts_temp=str(datetime.now());
            ts4file=ts_temp.replace(" ", "_");
                        zf=open("/home/pi/data_"+ts4file+".csv", 'a')
                        zf.write("Date Time,,Sensor1,,,,,,,,,,,,,Sensor2\r\n")
            zf.write(",,Acc,,,Gy,,,Mag,,,Qua,,,,,Acc,,,Gy,,,Mag,,,Qua,,,,\r\n");
            zf.write(",,X,Y,Z,X,Y,Z,X,Y,Z,A,B,C,D,,X,Y,Z,X,Y,Z,X,Y,Z,A,B,C,D\r\n");
            zf.close()
            else:
                    print("--------------------STOP MODE--------------------")
                    need_wr = 0;
                    GPIO.output(31, GPIO.LOW)
            ts4file="";
            time.sleep(0.5)
    time.sleep(0.1) # we read PipeStatus 5 time in second
        s1=sensor_get_measure(dev1, "Sensor1 ", need_wr)
        time.sleep(0.1)
        s2=sensor_get_measure(dev2, "Sensor2 ", need_wr)
    if (len(s1)>0  and len(s2)>0):
        if (need_wr>0):
            zf=open("/home/pi/data_"+ts4file+".csv", 'a')
            zf.write(str(datetime.now())+",,"+s1+","+s2+"\r\n")
            zf.close();
    if (GPIO.input(38) == GPIO.HIGH):
        time_for_power_off=time_for_power_off-1;
        print("time_for_power_off = "+str(time_for_power_off))
        if (time_for_power_off <= 0):
            print("--------------------POWER OFF--------------------")
            GPIO.output(40, GPIO.LOW)
            GPIO.output(31, GPIO.LOW)
            time.sleep(1)
            call("sudo nohup shutdown +0", shell=True)
            break;
    else:
        time_for_power_off=15;
dev1.close()
dev2.close()
#-------------------------------------------------------------------------------------------
```

Checksum calculation:

```
import smbus
import time

bus = smbus.SMBus(1)
add = 0x6b

#bus.write_i2c_block_data(add,3,[250, 255, 40, 0, 193])
bus.write_i2c_block_data(add,3,[250, 1, 40, 0, 191])
time.sleep(1)
var = bus.read_i2c_block_data(add,5)
print(var)
```