Inertial Measurement Units for Tracking Human Pose and Motions: Testing and Calibration

By

Casey Quinn LaCanna

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Biomedical Engineering

Written under the direction of

William Craelius

And approved by

New Brunswick, New Jersey

October, 2018

ABSTRACT OF THE THESIS

Inertial Measurement Units for Tracking Human Pose and Motions: Testing and Calibration

by CASEY QUINN LACANNA

Thesis Director:

William Craelius

This thesis presents the development of CAM, a Cranial Angle Monitor, using a Metawear Cpro Inertial Measurement Unit (IMU). The goal of CAM is to measure static cranial position, so the major task was to optimize the accuracy and stability of components of the IMU, by developing algorithms to compensate for errors. It was found that the gyroscope drift was linear, but varied among IMUs and among axes. Gyroscope performance was measured during multiple test runs under static conditions, to test the influence of several parameters, including, device, axis, run time, orientation, and time between runs. Drifts between devices and the 3 angular axes were tested and found to be inconsistent and could not be used for calibration. Other factors, including run duration, orientation, and time between runs, did not significantly influence repeatability, and results were consistent within 1°/min. Static tests were done with the IMU sitting on a table or attached to a vertical surface. Dynamic tests were done with the IMU fixed to a robotic arm and rotated 1 radian in each gyroscope axis individually to determine the sensor's accuracy. While the IMU proved highly accurate for dynamic motions, with < 0.33 degrees error, the inherent baseline drift of the gyroscope proved too large for static applications of hour long testing, such as HoBE tracking in hospitals. These results provide the first systematic test of factors influencing IMU accuracy in static and dynamic conditions for the CAM.

Table of Contents

Introduction

For patients who are confined to bed for long periods In the hospital, positioning the patient's head with respect to gravity is a simple but critical maneuver that attempts to optimally adjust cerebral blood flow (CBF). Maintaining proper levels of CBF, and the related parameter, intracranial pressure (ICP), is necessary to reduce the risk of ventilator-associated pneumonia and aspiration [1-3], as well as the risk of bedsores [1]. Its efficacy depends on several factors. Firstly, bed angle does not necessarily correlate with head posture: the patient's head seldom aligns with bed angle, due to bedding and other materials. Patients will also migrate on the bed, usually toward the foot, which reduces the torso and head angle [4]. Secondly, although recommendations of 30° or 45° head of bead elevation (HoBE) are generally followed in different situations, no clear standards exist for monitoring the cranial angle of Patients with Traumatic Brain Injury (TBI) during treatment in the Surgical Intensive Care Unit (SICU). Furthermore, compliance with recommendations is variable in ICUs [5-7]. Thirdly, reporting accuracy and perception of HoBE by clinicians can be poor, with only 53% of Physicians, and 50% of Nurses estimating it within ±5° in a study of compliance [8]. Finally, there is evidence that raising the head may not always be efficacious for patients with brain injury [9]. These factors must be understood and/or corrected in order to optimize treatments for TBI in the SICU, and are addressed here.

There is a need for an autonomous monitor of HoBE. Various HoBE sensing systems have been developed, but none are ideal. Beds can be instrumented with positional sensors [10,11], but these do not directly measure head angle as stated above, and are impractical to install in several beds of the SICU. Some advanced beds (i.e. Hill-Rom Advanta 2) include a digital goniometer that displays, but does not transmit bed angle. Motion capture systems have been applied to quantifying HoBE for research [4,12], however these are impractical in the SICU for reasons of bulkiness and privacy. Inclinometers for measuring torso angle in bed, via a sensor placed on the sternum have been designed, but these are not yet commercially available, and may not be appropriate [11,13]. My design for monitoring HoBE is the Cranial Angle Monitor (CAM), which is an adaptation of Inertial Measurement Units (IMUs) that are in common use within our lab for the past 10 years.

HoBE is a common and simple intervention, that when properly used in the ICU, can make the difference between (1) preserving or losing brain function, (2) aspirating fluid and contracting pneumonia or breathing normally, and possibly (3) acquiring bed sores or not. As noted, current practice in HoBE is sometimes arbitrary, not quantitative, and often inaccurately estimated. Electronic HoBE monitoring is a fairly small but necessary step to achieving these clinical goals.

CAM data can be used to estimate patient’s motions in the bed, in particular, sliding and subsequent repositioning by a nurse. Sliding in hospital beds can be detrimental to the patient, since they involve friction and shear to the skin, which are major sources of skin breakdown. It is also detrimental to the Nurses, who frequently injure their spines lifting the patient back up. The motion data will help to quantify these events, and possibly suggest improvements in bed adjustments.

Besides HoBE, there are other uses for a small device that can measure human movements, including balance tests and athletic training. In the lab the CAM was being used to measure head angle while a person was brushing their teeth. The CAM was programed to start collecting head position at the same time as a Adafruit sensor in the Dynabrush recorded force, acceleration, and gyroscope data. Other methods used to measure head angles are depth cameras and motion capture suits. While each of these methods is well studied to measure position of the body, it would be more expensive, and very difficult to implement in a clinical setting.

IMU's are electronic devices that contain a accelerometer, gyroscope, and sometimes a magnetometer. The primary purpose of an IMU is to record its position in space. IMU's can be used to help monitor and control unmanned aerial units, as well as help monitor movement of vehicles when a GPS signal is not available [14]. The difficulty in using an IMU is that accelerometer and gyroscope data cannot be simply integrated to determine the polar and cartesian position of the device. Some types of errors in position can result from numerous factors such as baseline offset, noise, and unstable mounting, but these are relatively easy to fix. The major problem with IMUs is signal drift with time, due to continuous accumulation of errors during the integration. Small errors in acceleration or angular velocity will continuously accumulate during integration, creating increasing signal error with time, which is referred to as drift [15]. To attempt corrections for the drift, filters, have been developed which incorporate both acceleration and gyroscopic data. For example, the Kalman filter minimizes error involved using mean-squared estimation, and is routinely used for autonomous manufacturing, and monitoring the altitude, position, and speed of ships, aircraft, and even satellites [16]. While very accurate, the Kalman filter would not be necessary for use in a CAM device. The other, more simplistic filter is the complementary filter, which used the equation

$$angle(i) \ = \ a \ * (angle(i-1) \ + gyro * dt) + (1-a) * acc$$

As a loop calculation, i refers to the current time point in the raw data, and angle is the calculated angle at that given time point. Gyro is the current angular velocity in the raw data of the gyroscope, acc is the angular velocity measured using acceleration, and dt is the time difference between each data point. a is a constant that ranges from .9 to 1 to increase the effect of gyroscope data over acceleration. The purpose of the complementary filter is that angular velocity based on acceleration acts as a low pass filter minimizing the effect of drift on the final output [17].

Methods

Design and Specification of CAM

The CAM is a fixation device that attaches a gyroscope to the postauricular region. This is a location where skin movement is negligible during cervical joint articulation. The device has a hook, similar to modern bluetooth headsets, that attaches to the ear. The hook then connects to a 3D printed container for the sensor. The weight of the total package, when printed, should be less than 100 grams. The current design for the CAM used in the lab can be seen in figure 1. The container is used to stabilize the position of the sensor in the device as well as create a non-conducting barrier between the sensor and patients skin. The container is designed in an upper and lower component that has a lip for attachment. The top portion of the container has a small hole for access to the push-button switch as well as creating a visual marker for the current orientation of the sensor. Additionally the hook and container will be covered by silicone for further safety and comfort. The sensor used for our CAM device is the Metawear Cpro from Mbientlab. Currently mbientlab sells containers similar to the container that our lab designed for their metawear sensors.

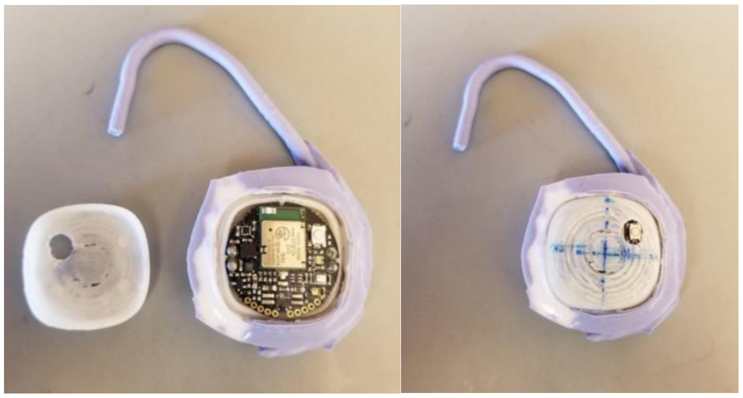


Figure 1- CAM Design

Metawear Cpro is a 9 axis IMU developed by mbientlab INC. For the implementation of the CAM we primarily used the gyroscope, the specifications of which can be found in table 1. Here I tested two Cpro sensors with serial numbers of 00C08D and 00C500, which will be referred to as metawear 1 and 2 respectively.

Table 1 - Metawear Cpro gyroscope specifications [18].

Specs	Description	Min.	Typ.	Max.	Units
	measurement range	±125		±2000	°/s
	Resolution	16		262	counts/°
f data	Data sample frequency	25		3200	Hz
I gyro	Gyro active current		850	900	uA

Baseline Drift Correction: Static tests

While angular position theoretically is the integral of gyroscope angular velocity, this calculation is prone to errors, mainly due to baseline drift. To counter this implementation of complementary filter was attempted but difficult because the application used with the metawear sensor could not simultaneously measure both acceleration and gyroscope. During experimentation it was found that the drift for the Cpro gyroscopes is linear. This is a result of the gyroscope baseline being slightly off of zero, and stable. Assuming that the integration of this small offset is perfectly linear and consistent, it would be easy to correct for drift by measuring the baseline and removing it at every integration step instead of using the complementary filter. In fact, drift is linear during any given run, however, it is not consistent across all factors. For example, figure 2 shows that the drift varies greatly between each axis as well as between each device. Each axis and each device can be corrected for separately, but the drift might not be consistent between runs, or might be affected by the current orientation of the sensor. These two confounding factors would defeat attempts at baseline correction.

To determine what factors influence the drift, stationary tests were performed. Under different conditions each of the IMUs were held in a stationary position for at least 30 seconds. 9 to 10 runs were measured back to back for each position.

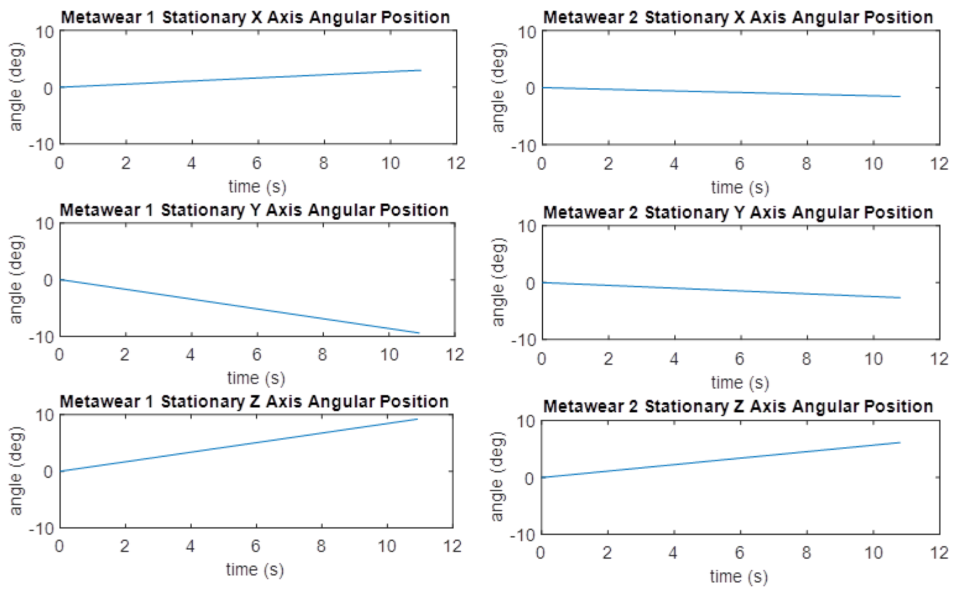


Figure 2- Line plot for angular position of metawear C sensor 1. The recorded angular position was integrated from angular velocity of the three angular axes, which was measured using the sensor's gyroscope. Sensor was stationary throughout measurement.

The average and standard deviation of these runs were calculated for comparison. To determine if orientation is a factor, each sensor underwent this test on a flat table as well as attached onto a vertical surface. The tests were conducted again after two weeks to determine if the drift remained consistent over time. In addition, differences among axes, x, y, and z, and between sensors were quantified for comparison.

Three methods for correcting baseline drift were tested which can be seen as a matlab code in Appendix A. Method 1: During integration of the gyroscope's angular velocity to polar position, a fixed value was removed after each integration step. The fixed value was the slope of the first 30 seconds of stationary recording. The effects of this method can be seen in figure 3, were each axis was rotated approximately 90° then brought back to its original position in succession. Method 2: Doing a linear fit of the integral of the gyroscope data, and then re-integrating by removing the measured slope from the linear fit. Method 3 is similar to the first, but instead of measuring drift of each run, a calibration measurement is taken beforehand and used for all calculations using that metawear. While the third method will be insufficient if the baseline changes over time, it also can create a more reliable and streamlined method of removing drift.

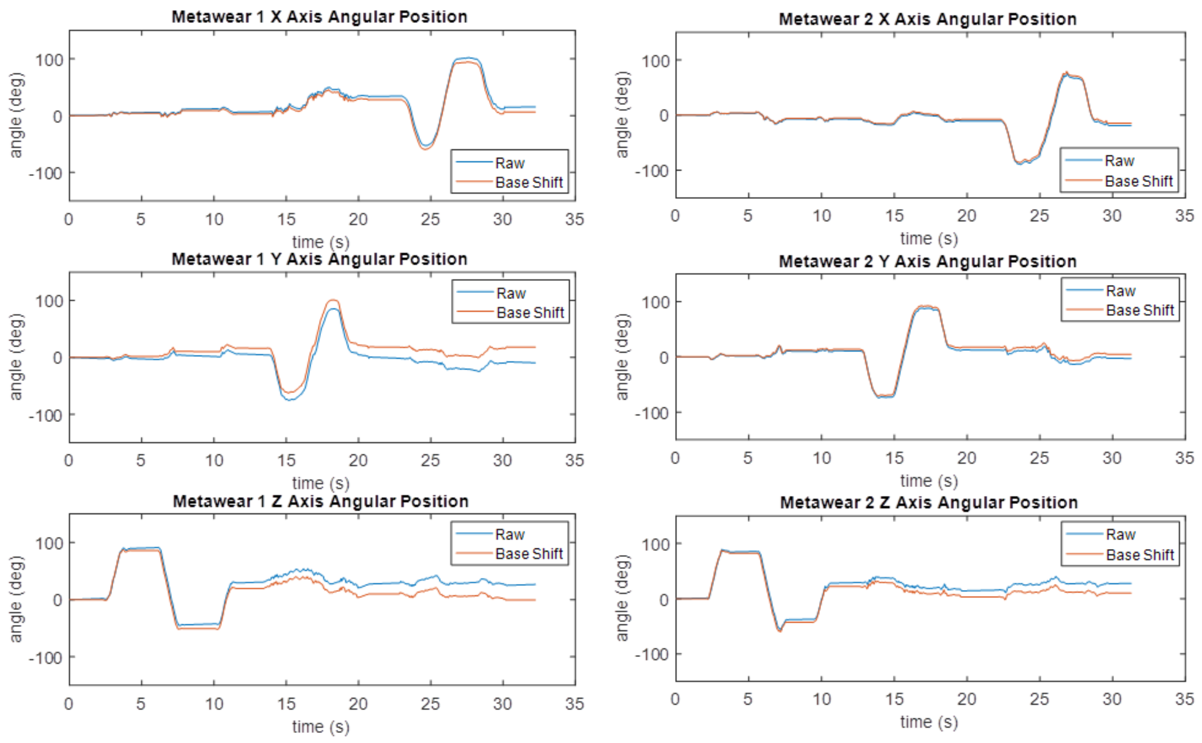


Figure 3- Comparison between raw (blue) and baseline corrected (red) angular positions of Metawear C sensor 1. Raw data were measured by integrating the gyroscope angular velocity, while the base shift data had continuous removal of gyroscope data using a previous stationary test. Each axis of the sensor was rotated approximately 90 degrees followed by a 180 degree shift in the opposite direction.

Accuracy Testing with Robotic Arm.

A robotic arm was used to determine how accurate the angular position calculated with the metawear. The robotic arm is a product developed by Barrett Technology called the WAM. The sensor was attached with velcro to the robot. The metawear was attached to what is effectively the wrist of the robot and oriented such that the x axis is aligned with the direction of the arm, as seen in figure 4. To quantify accuracy assuming significant angular velocity in one axis, one joint on the arm corresponding to a specific axis was rotated back and forth by one radian. This allows verification of sensor data with actual position of the robot.

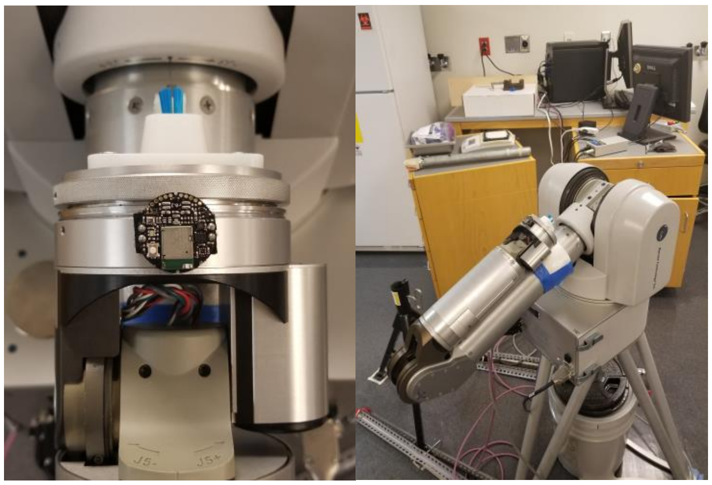


Figure 4- CAM Setup on the robotic arm

Coordination with Other Sensors

One other aspect for the development of the Metawear Cpro into a medical device is having it work in tandem with multiple sensors. Being able to collect multiple types of data in multiple locations allows for the device to be used for more diverse applications. Currently mbientlab has already designed a mobile application that allows for multiple metawear sensors to be recorded at the same time. On the other hand there was no way of having different types of sensors synchronized with the Cpro. Creating a bluetooth connection and a simultaneous starting signal are the two important aspects

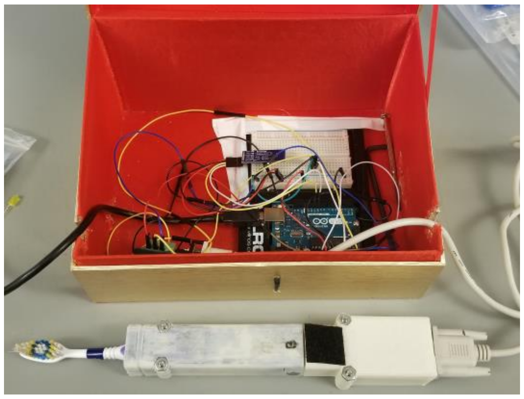


Figure 5- Dynabrush and Arduino board used to test synchronization with Metawear Cpro

of recording data from both sensors. To implement and test a Bluetooth connection we decided to use the Dynabrush system in figure 5. A bluetooth connection was formed using a bluetooth module that attached to the arduino board. In addition the mobile application needed to be programed to allow for non metawear bluetooth connections. The new application code was programed using android studio and the original Metawear application code from mbientlab. The code that was designed by us can be found in Appendix B. The arduino program was used to start data collection of the arduino sensor. We added code that can be seen in Appendix C, which would send a 5 digit signal from the bluetooth module when the arduino started recording. If the metawear application was connected to the bluetooth signal, the metawear would read the signal and start collecting data.

The metawear would always start recording after the Adafruit sensor in the Dynabrush, so this needed correction. Figure 6 shows recordings of both sensors when the metawear was attached to the Dynabrush. While this could be corrected by shifting one set over, when the sensors are measuring separately it would be very difficult to match them together. To alleviate this problem calibration tests were performed to determine if the offset between the two sensors is consistent. The Cpro was attached, with velcro, to the Dynabrush with a similar orientation and position to the sensor in the Dynabrush. Five runs were performed, and points that were easily distinguishable in both sensors were picked for each of the three axes. The difference in time between the two sensors was measured and compared.

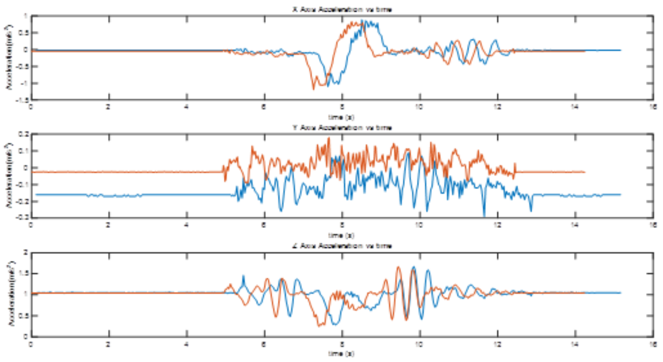


Figure 6– Time lag between metawear sensor (blue) and arduino sensor (red) using edited mbientlab application.

Results

Baseline Correction

To determine the best baseline correction of IMU signals, stationary tests were conducted to measure the drift slope. These tests were performed to determine the linearity and consistency of the slope, and the results are displayed in tables 2 and 3. The factors included (1) the measured axis, (2) time between runs (3) device, and (4) the current angular position of the sensor. The recording of table 2 and 3 were taken 2 weeks apart to measure the consistency over long period of time. When comparing between the 4 major factors, instead of using standard deviation, the average difference between the angular drift averages is recorded.

Table 2- 30 second recordings of two stationary metawear devices on February 22nd, 2018. Recordings were taken between 9 to 10 times on a flat table (labeled ad 0 degree position) and attached to a vertical object with velcro(labeled as 90 degree position). Averages and standard deviations of angular drift were calculated using Excel.

Metawear	Axis	Position (°)	Duration (s)	Iterations	Angular Drift Average ± S.D. (°/s)
1	x	0	30	9	0.29266 ± 0.00258
1	y	0	30	9	-0.94509 ± 0.00577
1	z	0	30	9	0.82520 ± 0.00225
1	x	90	30	10	0.32767 ± 0.00658
1	y	90	30	10	-0.92189 ± 0.00831
1	z	90	30	10	0.82041 ± 0.00335
2	x	0	30	10	-0.17152 ± 0.00761
2	y	0	30	10	-0.24403 ± 0.00798
2	z	0	30	10	0.56894 ± 0.00460
2	x	90	30	10	-0.17172 ± 0.00708
2	y	90	30	10	-0.26093 ± 0.00356
2	z	90	30	10	0.57133 ± 0.00262

Table 3- 30 second recordings of two stationary metawear devices on March 8th, 2018. Recordings were taken 10 times on a flat table (labeled ad 0 degree position) or attached to a vertical object with velcro (labeled as 90 degree position). Averages and standard deviations of angular drift were calculated using Excel.

Metawear	Axis	Position (°)	Duration (s)	Iterations	Angular Drift Average ± S.D. (°/s)
1	x	0	30	10	0.29473 ± 0.00268
1	y	0	30	10	-0.93504 ± 0.00562
1	z	0	30	10	0.77847 ± 0.00451
1	x	90	30	10	0.29408 ± 0.00220
1	y	90	30	10	-0.95385 ± 0.00846
1	z	90	30	10	0.77228 ± 0.00145
2	x	0	30	10	-0.14406 ± 0.00819
2	y	0	30	10	-0.26186 ± 0.003021
2	z	0	30	10	0.56432 ± 0.00592
2	x	90	30	10	-0.14223 ± 0.00270
2	y	90	30	10	-0.28351 ± 0.00357
2	z	90	30	10	0.56147 ± 0.00382

Recorded average slopes from Tables 2 and 3 ranged from -0.9539°/s to 0.8252°/s. If the raw data were integrated without any correction, the largest magnitude of drift would result in a 3433.86° error over 1 hour. To correct this drift, previous data were used as calibration to remove error at every integration step. But if the expected linear drift is not consistent then there would still exist drift errors in the final results. For comparison the difference between average drift readings were recorded, which was used to approximate expected error if one was used to calibrate the other. The average difference between two axes of the same metawear is 1.2316°/s, and between the same axis of different metawear is 0.4562°/s. If we were to use measurements from one to remove the drift of the other the resulting errors would respectively create an offset of 4433.76° and 1642.32° over 1 hour. The average standard deviation is 0.004718°/s which relates to an offset of 16.986°/hr. The average difference in drift between 0 and 90 degree recordings was 0.0112°/s which would result in a shift of 40.32°/hr. Lastly the average difference in drift over a two week period is 0.0237 °/s which results in a shift of 85.32°/hr.

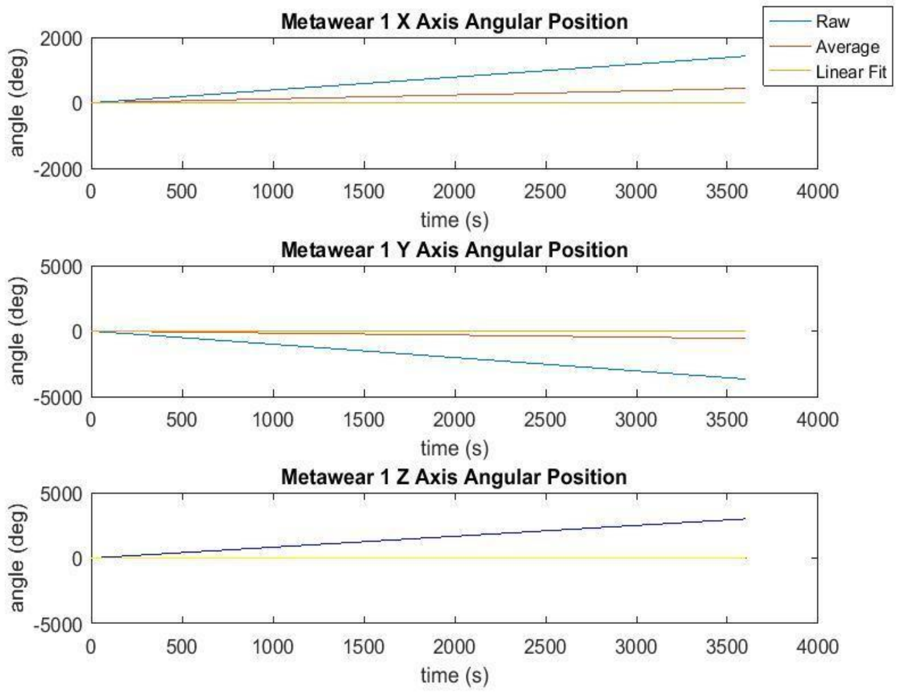


Figure 7 - Comparison between raw angular position (blue) with angular position with a baseline shift using measured average (red), and with a baseline shift using linear regression (yellow) over 1 hour.

A long stationary test was performed to determine if drift remained linear over a 1 hour period of time. Figure 7 shows the total shift in the raw data over the 1 hour period. Figure 8 also shows the result of using the two methods to correct the base line. The average method used the average slope of stationary tests performed a few weeks prior. Linear fit used the matlab function to correct for baseline. Table 4 shows the measured angle at 60 seconds for the raw data as well as the two baseline corrective methods.

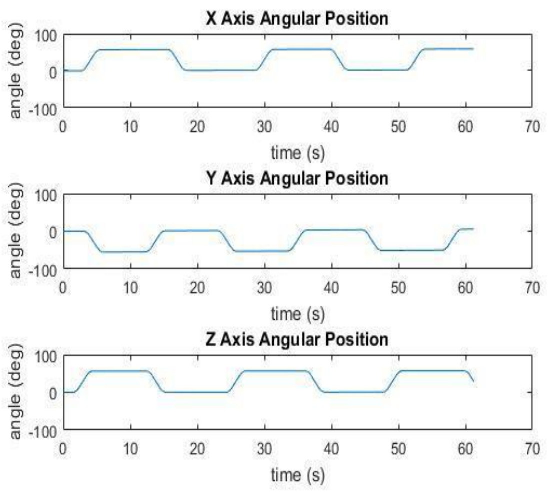
Table 4- Final angular positions measured after one hour without moving sensor, starting at 0°. Comparison between data without any baseline shift, using a measured average to shift base, and a linear fit attached to shift baseline. Results visualized in figure 7.

Axis	Final Time (s)	Raw Final Position(°)	Average Final Position(°)	Linear Fit Final Position(°)
x	3604	1424	438.5	-0.2048
y	3604	-3672	-574	2.463
z	3604	2997	-35.23	-1.768

Accuracy Analysis

After analysis of drift it is important to determine if the sensor and integration of the angular velocity can accurately measure the angular position of the device. Accuracy is being accessed through the three methods of movement of one axis, movement of multiple axes, and consistency of cyclic movement. Table 5 shows the results of repetitive 1 radian shift of each axis individually. The amplitude of after rotation was recorded and each axis differed by less than 0.3° over the three rotations.

Table 5- Recorded angular position after 1 radian rotations of metawear Cpro using WAM robotic arm.

X axis height(°)	Y axis height(°)	Z axis height(°)	
56.26	-55.69	56.17	
56.52	-55.44	56.25	
56.57	-55.43	56.35	
56.49	-55.36	56.35	
56.5	-55.4	56.27	

Calibration with Arduino Sensor

To calibrate the metawear to an arduino sensor, the time difference between the two sensors accelerometer data was recorded. The time difference exists because when the arduino starts recording the metawear is waiting for a Bluetooth signal, and therefore will begin recording later that the arduino. For calibration purposes it would be best if time differences between the two are consistent. Table 6 shows 15 different measurements over 5 different runs. The average difference was 0.2996 ± 0.0374 s. The points were picked by finding distinguishable peaks in both the arduino and metawear data. In conclusion the two signals were in concordance with each other, but had a lag of approximately 0.3 s between when the Adafruit and metawear started recording.

Table 6- Time differences between peak acceleration values of metawear sensor and arduino sensor contained inside the Dynabrush.

Run	Axis	Metawear Time (s)	Toothbrush Time (s)	Difference (s)
1	X	10.4	10.75	.35
	Y	7.68	8	.32
	Z	6.76	7.062	.302
2	X	5.82	6.098	.278
	Y	4.8	5.060	.26
	Z	2.94	3.187	.247
3	X	5.12	5.4	.28
	Y	3.1	3.355	.255
	Z	1.68	1.917	.237
4	X	5.7	6.033	.333
	Y	4.28	4.627	.347
	Z	1.66	1.984	.324
5	X	3.18	3.489	.309
	Y	5.2	5.531	.331
	Z	2.5	2.821	.321

Discussion

For practical use in testing and clinical settings as a HoBE, the CAM would need to be accurate within a few degrees. While recording short term tests such as balance or doing activities such as tooth brushing, the accuracy should hold true for approximately 10 minutes. On the other hand measuring HoBE would require hours of continuous recording. As such, it is necessary to judge the applicability of the Metawear Cpro for both long and short term testing separately. Based off of data in tables 2 and 3, differences in drift between each axis and sensor would lead to hundreds of degrees of error over the course of 10 minutes. Errors of this magnitude are unacceptable, but could be corrected on specific metawear sensors. The difference in drift correlated to a two week gap between measurements would result in an 85.32°/hr error. Compared to axis and device data the difference is significantly smaller, but would still be unacceptable in either short or long term testing. While calibration data from previous weeks are capable of being used depending on the length of the test and necessary accuracy, it would be recommended that calibration is redone for each testing session or everyday.

If the differences in standard deviation between sets of back to back runs are too large, than that would mean that calibration would have to be performed before each run or though post processing. For use in a clinical setting, constant calibrations would create more work for nurses, which is counterproductive to the purpose of the CAM. Similarly post processing the data defeats the entire purpose of continuous real time monitoring of HoBE. The standard deviations for back to back runs ranged from 0.00145°/s to 0.00846°/s which would result in 5.22°/hr and 30.456°/hr shifts respectively. The large difference makes it unreliable to us an average standard deviation. Even the largest drift is suitable for short term testing but cannot be used for long periods.

If the drift has significant differences at different orientations, it that would suggest that orientational factors such as gravity affects the baseline. If true, the possibility of removing drift through a linear manner would be impossible because the linearity would change as the device is moving. Orientation testing resulted in an error of 40.32°/hr which is usable for short term but should not be used for long-term testing. While the average deviation of orientation is larger than any specific set of runs, the difference is not large and might be attributed to the less secure method of holding the metawear device vertically with Velcro. The orientation of the device does not seem to have significant enough impact on the drift to eliminate the possibility of baseline correction though linear drift removal.

For short term testing the use of a calibration method to remove drift is sufficient, but unfortunately it would not work for long term tests such as HoBE monitoring. The hour long stationary test in figure 7 shows that using previously measured drift could still result in hundreds of degrees of error. While the calibration method is not sufficient it is possible to use the linear fit method of correcting the baseline. In short term testing using a linear fit would often create significantly more error than using the calibration method. Large fluctuations of the data in a short period of time made linear fitting unreliable, but if used over hour long periods linear fitting in combination with the calibration method could correct the leftover drift. In particular a linear fit would work well with HoBE monitoring because the purpose of the device is to keep the position of the CAM stationary at its initial position.

When using the robotic arm to determine accuracy of the Cpro the device was rotated in one axis for a radian. The measured change using post processing drift removal ranged from 55.36° to 56.57°. A radian is approximately 57.3° which creates a maximum difference of 2° between expected and measured change. While 2° is significant it falls within the necessary range. Previous studies using IMUs also commonly have errors between 0.5° and 2° of error [19]. The largest difference between readings of a single axis was 0.33° which shows that the metawear device is very precise and that the error in accuracy might be a result of the robotic arm rather than the sensor. Other studies find much larger values for precision, as high as 4.3°, but these test are done on a human where movements are more dynamic, sporadic, and less controllable [20]. It is important to note that the only method of recording real angle is the program controlling the WAM which could have had a small error with calibration. While other studies tracking human motion with IMUs have used robotic arms for testing, the gold standard for tracking movement is using a optical motion sensor (OMS) [21]. To further verify the accuracy of the Metawear Cpro it might be useful to incorporate an OMS.

The average difference when measuring time points for the metawear and arduino sensor was 0.2996 ± 0.0374 s. For practical purposes adding a shift of 0.3 seconds could potentially create a time shift of 0.05s between the two measurements. It is important to note that the Cpro has a frequency of 25 Hz which is significantly larger than the 3 Hz of the Adafruit sensor. The difference in time was recorded by determining distinguishable peaks in both data sets and measuring the time difference between them. Due to the difference in sampling rates there is a chance that despite being similar the peaks were not actually the same point, which could exaggerate the standard deviation. Even if this was not true a difference of 0.05 seconds is not relevant for most practical purposes.

Compared to other similar studies using IMUs [19-21], we share similar problems. They also seemed to find that the drift was linear but changed between runs and patients, with differences up to 116° over 6 minutes [21]. Even though all of the other studies used Kalman or some alteration of Kalman filtering, none performed long term testing. Normally tests were a few minutes long with a one lasting 15 minutes [21]. Common difficulties of other studies such as soft tissue artifacts and calibration based on location and expected movement were not yet explored using our method and the Metawear Cpro. Soft tissue artifacts are a displacement of the device compared to the bone, and can cause significant errors in IMU measurement and differ depending on where the IMU is placed [20]. For the CAM soft tissue artifacts would not be as large of an issue due to its position behind the ear with small amounts of tissue, but would have to be accounted for other applications. It was found that the placement of the IMU and the expected movement have to be taken into account, for example some algorithms were specifically designed for gait or IMU placement on the thigh [20].

To make the CAM a product for clinical use, several steps must be taken. First, long-term non-stationary measurements should be taken to determine if a linear fit can correct drift for hour long uses. Second, if linear fitting seems usable, data from the metawear needs to be accessible through a computer and integrated in real time. While linear fitting can increase the accuracy of results over long periods of time, the initial results of the test might be too unreliable for use. A combined method of calibration and linear fitting could decrease the effect of drift is for initial readings as well as long-term results. Other research involving use of IMUs for study of human movement sometimes use sections of stationary movement to help in reduction of drift [21]. While not ideal, stationary readings during each run would eliminate the need of preliminary calibration. If the sensor is still incapable of measuring accurate head position over hour long periods, it would be best to implement the complementary or Kalman filter.

References

1. Metheny, N.A. and R.A. Frantz, Head-of-Bed Elevation in Critically Ill Patients: A Review. Critical Care Nurse, 2013. 33(3): p. 53-65.

2. Hunter, A.J., et al., HOBOE (Head-of-Bed Optimization of Elevation) Study: Association of Higher Angle With Reduced Cerebral Blood Flow Velocity in Acute Ischemic Stroke. Physical Therapy, 2011. 91(10): p. 1503-1512.

3. De Blasio, D., et al., The effect of head of bed position on aspiration of gastric contents in patients receiving mechanical ventilation: A comparison of 30 vs 45 degrees. Chest, 2006. 130(4): p. 212S-212S.

4. Wiggermann, N., et al., The Effect of Patient Migration in Bed on Torso Elevation. Nursing Research, 2015. 64(3): p. 221-225.

5. Helman, D.L., et al., Effect of standardized orders and provider education on head-of-bed positioning in mechanically ventilated patients. Critical Care Medicine, 2003. 31(9): p. 2285-2290.

6. Fitch, Z.W., et al., Hospital Bed Type, the Electronic Medical Record, and Safe Bed Elevation in the Intensive Care Setting. American Journal of Medical Quality, 2016. 31(1): p. 69-72.

7. Llaurado-Serra, M., et al., Evaluation of head-of-bed elevation compliance in critically ill patients under mechanical ventilation in a polyvalent intensive care unit. Medicina Intensiva, 2015. 39(6): p. 329-336.

8. Hiner, C., et al., CLINICIANS' PERCEPTION OF HEAD-OF-BED ELEVATION. American Journal of Critical Care, 2010. 19(2): p. 164-167.

9. Kim, M.N., et al., Continuous Optical Monitoring of Cerebral Hemodynamics During Head-of-Bed Manipulation in Brain-Injured Adults. Neurocritical Care, 2014. 20(3): p. 443-453.

10. Balonov, K., et al., A novel method of continuous measurement of head of bed elevation in ventilated patients. Intensive Care Medicine, 2007. 33(6): p. 1050-1054.

11. Johnson, M.D. and R.L. Clark, Angular elevation notification system for alerting caregiver or hospital staff about improper absolute angular orientation of upper body of patient lying on hospital bed, has notification device alerting caregiver if angle is outside range, Egresson Llc; Egression Llc.

12. Kotowski, S.E., et al., Quantification of Patient Migration in Bed: Catalyst to Improve Hospital Bed Design to Reduce Shear and Friction Forces and Nurses' Injuries. Human Factors, 2013. 55(1): p. 36-47.

13. Glaser, F., V. Gamarnik, and D. Karlin, Method for managing health conditions associated with posture of bedridden patient in hospital, involves calculating angle formed by cross sectional plane of device and plane of reference based on detected change in posture, Glaser F; Gamarnik V; Karlin D; Angulus Corp.

14. Song, Y., Nuske, S., & Scherer, S. (n.d). A Multi-Sensor Fusion MAV State Estimation from Long-Range Stereo, IMU, GPS and Barometric Sensors. *Sensors*, *17*(1).

15. Trajkovski, G. (2009). Springer handbook of robotics. *CHOICE:* Current Reviews for Academic Libraries, (6). 1132. p. 480-490

16. Grewal M, Andrews A. Kalman Filtering : Theory And Practice Using *MATLAB*. Hoboken, New Jersey: John Wiley & Sons, 2015: p. 1-7

17. Higgins W. A Comparison of Complementary and Kalman Filtering. *IEEE Transactions On Aerospace & Electronic Systems* [serial online]. January 3, 1975;AES-11(3):321.

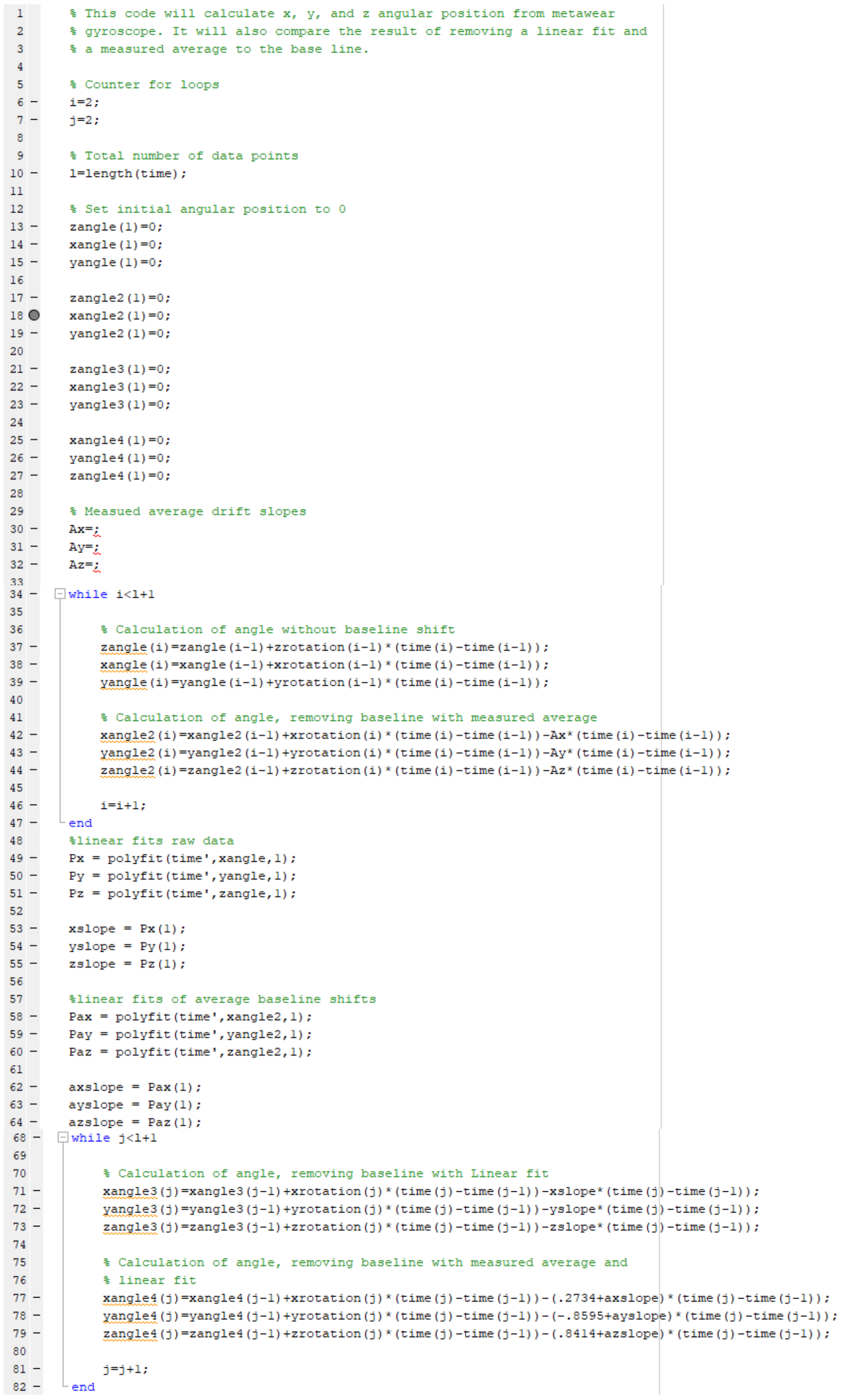
18. MetawearC Product Specification v1.0. MbientLab Inc. https://mbientlab.com/documents/MetaWearC-CPRO-PS.pdf

19. Yoshioka, S., Fujita, Z., Hay, D.C. et al. Pose tracking with rate gyroscopes in alpine skiing. Sports Eng (2018) 21: 177.

20. A New Training Assessment Method for Alpine Ski Racing: Estimating Center of Mass Trajectory by Fusing Inertial Sensors With Periodically Available Position Anchor Points.

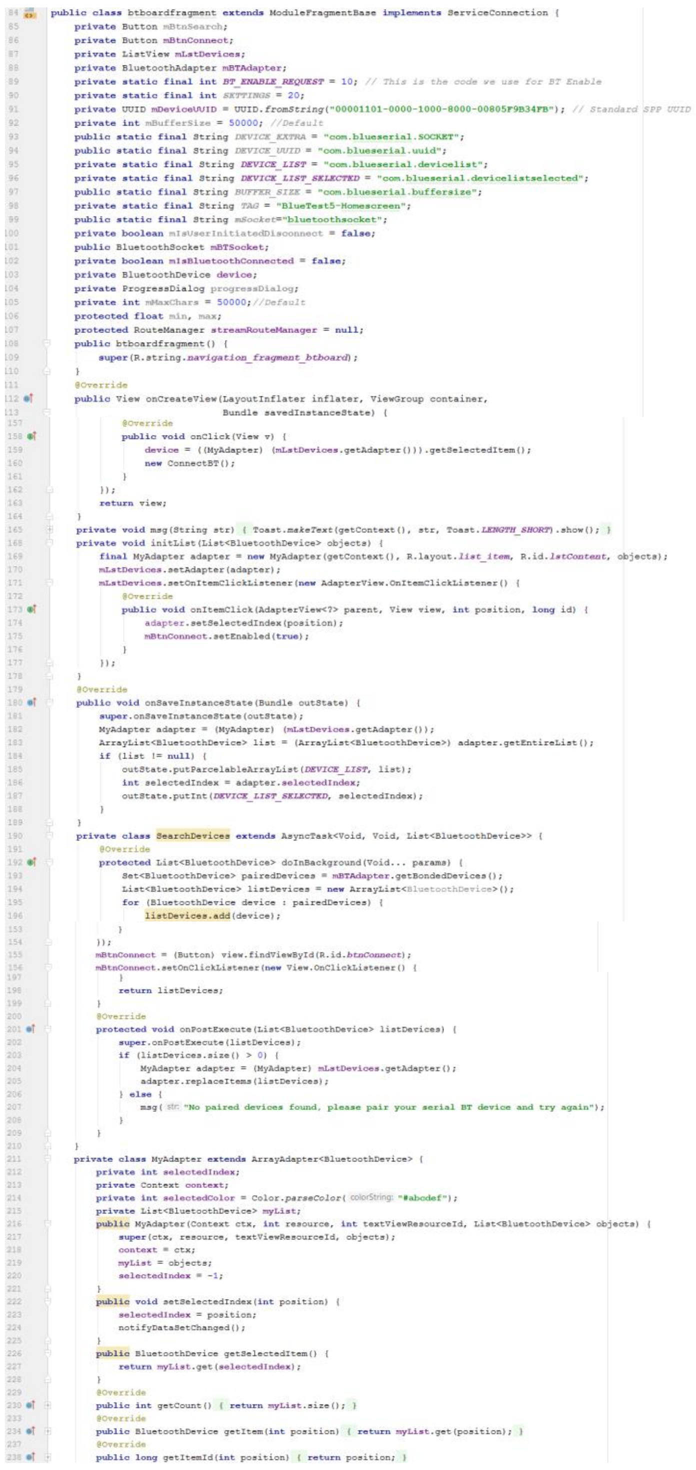
21. Teufl, Wolfgang & Miezal, Markus & Taetz, Bertram & Fröhlich, Michael & Bleser, Gabriele. (2018). Validity, Test-Retest Reliability and Long-Term Stability of Magnetometer Free Inertial Sensor Based 3D Joint Kinematics. Sensors. 18. 10.3390/s18071980.

Appendix A

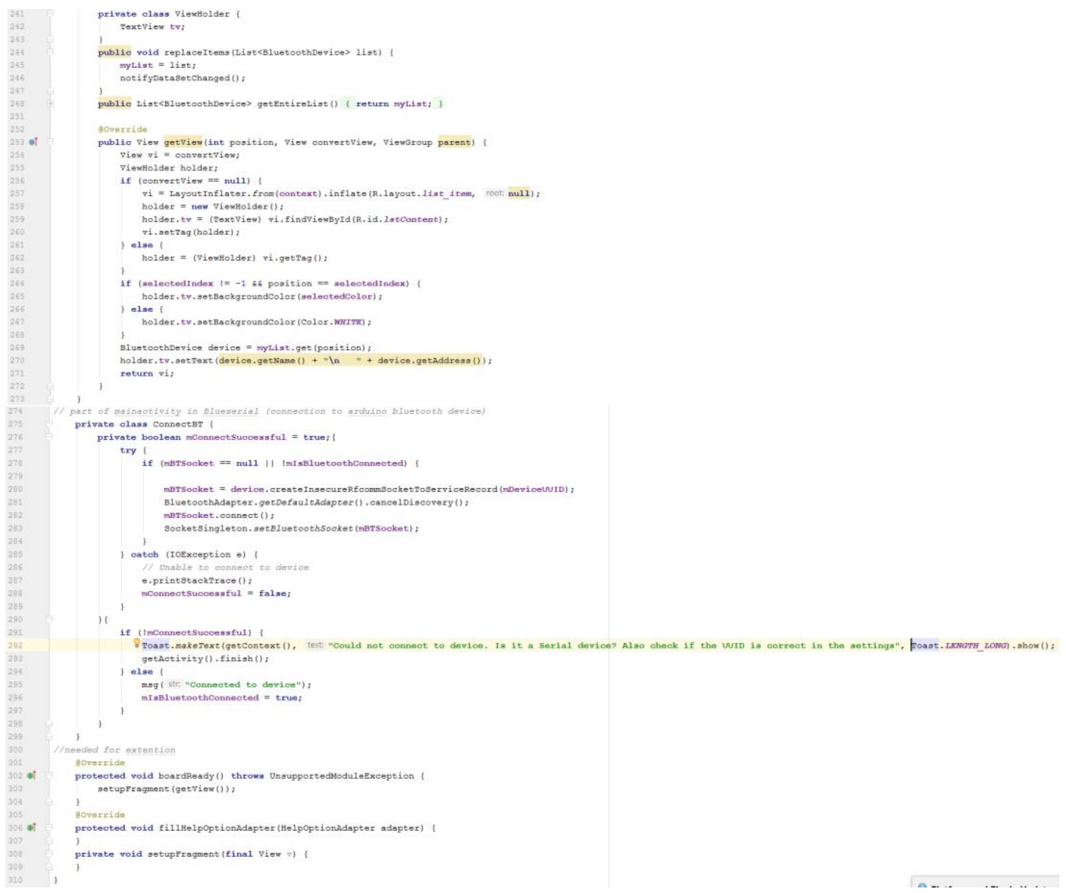


```matlab
% This code will calculate x, y, and z angular position from metawear
% gyroscope. It will also compare the result of removing a linear fit and
% a measured average to the base line.

% Counter for loops
i=2;
j=2;

% Total number of data points
l=length(time);

% Set initial angular position to 0
zangle(1)=0;
xangle(1)=0;
yangle(1)=0;

zangle2(1)=0;
xangle2(1)=0;
yangle2(1)=0;

zangle3(1)=0;
xangle3(1)=0;
yangle3(1)=0;

xangle4(1)=0;
yangle4(1)=0;
zangle4(1)=0;

% Measued average drift slopes
Ax=;
Ay=;
Az=;

while i<l+1

    % Calculation of angle without baseline shift
    zangle(i)=zangle(i-1)+zrotation(i-1)*(time(i)-time(i-1));
    xangle(i)=xangle(i-1)+xrotation(i-1)*(time(i)-time(i-1));
    yangle(i)=yangle(i-1)+yrotation(i-1)*(time(i)-time(i-1));

    % Calculation of angle, removing baseline with measured average
    xangle2(i)=xangle2(i-1)+xrotation(i)*(time(i)-time(i-1))-Ax*(time(i)-time(i-1));
    yangle2(i)=yangle2(i-1)+yrotation(i)*(time(i)-time(i-1))-Ay*(time(i)-time(i-1));
    zangle2(i)=zangle2(i-1)+zrotation(i)*(time(i)-time(i-1))-Az*(time(i)-time(i-1));

    i=i+1;
end
%linear fits raw data
Px = polyfit(time',xangle,1);
Py = polyfit(time',yangle,1);
Pz = polyfit(time',zangle,1);

xslope = Px(1);
yslope = Py(1);
zslope = Pz(1);

%linear fits of average baseline shifts
Pax = polyfit(time',xangle2,1);
Pay = polyfit(time',yangle2,1);
Paz = polyfit(time',zangle2,1);

axslope = Pax(1);
ayslope = Pay(1);
azslope = Paz(1);
while j<l+1

    % Calculation of angle, removing baseline with Linear fit
    xangle3(j)=xangle3(j-1)+xrotation(j)*(time(j)-time(j-1))-xslope*(time(j)-time(j-1));
    yangle3(j)=yangle3(j-1)+yrotation(j)*(time(j)-time(j-1))-yslope*(time(j)-time(j-1));
    zangle3(j)=zangle3(j-1)+zrotation(j)*(time(j)-time(j-1))-zslope*(time(j)-time(j-1));

    % Calculation of angle, removing baseline with measured average and
    % linear fit
    xangle4(j)=xangle4(j-1)+xrotation(j)*(time(j)-time(j-1))-(.2734+axslope)*(time(j)-time(j-1));
    yangle4(j)=yangle4(j-1)+yrotation(j)*(time(j)-time(j-1))-(-.8595+ayslope)*(time(j)-time(j-1));
    zangle4(j)=zangle4(j-1)+zrotation(j)*(time(j)-time(j-1))-(.8414+azslope)*(time(j)-time(j-1));

    j=j+1;
end
```

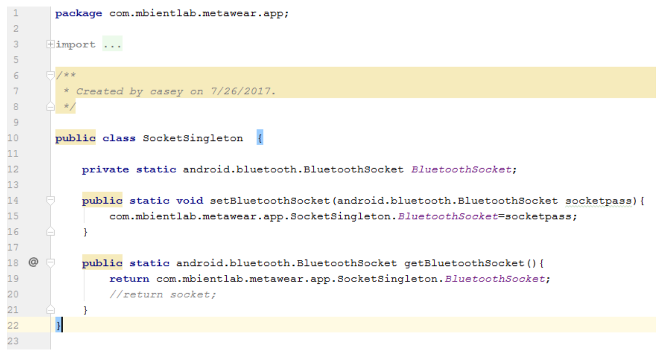
Appendix B



```java
public class btboardfragment extends ModuleFragmentBase implements ServiceConnection {
    private Button mBtnSearch;
    private Button mBtnConnect;
    private ListView mLstDevices;
    private BluetoothAdapter mBTAdapter;
    private static final int BT_ENABLE_REQUEST = 10; // This is the code we use for BT Enable
    private static final int SETTINGS = 20;
    private UUID mDeviceUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"); // Standard SPP UUID
    private int mBufferSize = 50000; //Default
    public static final String DEVICE_EXTRA = "com.blueserial.SOCKET";
    public static final String DEVICE_UUID = "com.blueserial.uuid";
    private static final String DEVICE_LIST = "com.blueserial.devicelist";
    private static final String DEVICE_LIST_SELECTED = "com.blueserial.devicelistselected";
    public static final String BUFFER_SIZE = "com.blueserial.buffersize";
    private static final String TAG = "BlueTest5-Homescreen";
    public static final String mSocket="bluetoothsocket";
    private boolean mIsUserInitiatedDisconnect = false;
    public BluetoothSocket mBTSocket;
    private boolean mIsBluetoothConnected = false;
    private BluetoothDevice device;
    private ProgressDialog progressDialog;
    private int mMaxChars = 50000;//Default
    protected float min, max;
    protected RouteManager streamRouteManager = null;
    public btboardfragment() {
        super(R.string.navigation_fragment_btboard);
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
            @Override
            public void onClick(View v) {
                device = ((MyAdapter) (mLstDevices.getAdapter())).getSelectedItem();
                new ConnectBT();
            }
        });
        return view;
    }
    private void msg(String str) { Toast.makeText(getContext(), str, Toast.LENGTH_SHORT).show(); }
    private void initList(List<BluetoothDevice> objects) {
        final MyAdapter adapter = new MyAdapter(getContext(), R.layout.list_item, R.id.lstContent, objects);
        mLstDevices.setAdapter(adapter);
        mLstDevices.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                adapter.setSelectedIndex(position);
                mBtnConnect.setEnabled(true);
            }
        });
    }
    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        MyAdapter adapter = (MyAdapter) (mLstDevices.getAdapter());
        ArrayList<BluetoothDevice> list = (ArrayList<BluetoothDevice>) adapter.getEntireList();
        if (list != null) {
            outState.putParcelableArrayList(DEVICE_LIST, list);
            int selectedIndex = adapter.selectedIndex;
            outState.putInt(DEVICE_LIST_SELECTED, selectedIndex);
        }
    }
    private class SearchDevices extends AsyncTask<Void, Void, List<BluetoothDevice>> {
        @Override
        protected List<BluetoothDevice> doInBackground(Void... params) {
            Set<BluetoothDevice> pairedDevices = mBTAdapter.getBondedDevices();
            List<BluetoothDevice> listDevices = new ArrayList<BluetoothDevice>();
            for (BluetoothDevice device : pairedDevices) {
                listDevices.add(device);
            }
        });
        mBtnConnect = (Button) view.findViewById(R.id.btnConnect);
        mBtnConnect.setOnClickListener(new View.OnClickListener() {
            }
            return listDevices;
        }
        @Override
        protected void onPostExecute(List<BluetoothDevice> listDevices) {
            super.onPostExecute(listDevices);
            if (listDevices.size() > 0) {
                MyAdapter adapter = (MyAdapter) mLstDevices.getAdapter();
                adapter.replaceItems(listDevices);
            } else {
                msg( str: "No paired devices found, please pair your serial BT device and try again");
            }
        }
    }
    private class MyAdapter extends ArrayAdapter<BluetoothDevice> {
        private int selectedIndex;
        private Context context;
        private int selectedColor = Color.parseColor( colorString: "#abcdef");
        private List<BluetoothDevice> myList;
        public MyAdapter(Context ctx, int resource, int textViewResourceId, List<BluetoothDevice> objects) {
            super(ctx, resource, textViewResourceId, objects);
            context = ctx;
            myList = objects;
            selectedIndex = -1;
        }
        public void setSelectedIndex(int position) {
            selectedIndex = position;
            notifyDataSetChanged();
        }
        public BluetoothDevice getSelectedItem() {
            return myList.get(selectedIndex);
        }
        @Override
        public int getCount() { return myList.size(); }
        @Override
        public BluetoothDevice getItem(int position) { return myList.get(position); }
        @Override
        public long getItemId(int position) { return position; }
```

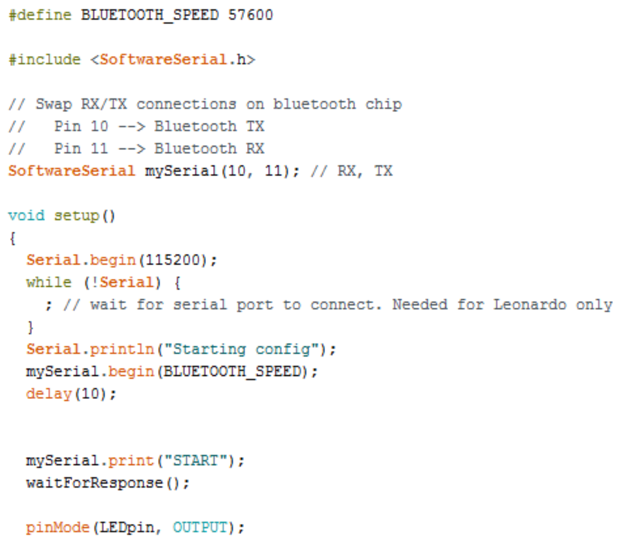


```
        private class ViewHolder {
            TextView tv;
        }
        public void replaceItems(List<BluetoothDevice> list) {
            myList = list;
            notifyDataSetChanged();
        }
        public List<BluetoothDevice> getEntireList() { return myList; }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            View vi = convertView;
            ViewHolder holder;
            if (convertView == null) {
                vi = LayoutInflater.from(context).inflate(R.layout.list_item,  root: null);
                holder = new ViewHolder();
                holder.tv = (TextView) vi.findViewById(R.id.lstContent);
                vi.setTag(holder);
            } else {
                holder = (ViewHolder) vi.getTag();
            }
            if (selectedIndex != -1 && position == selectedIndex) {
                holder.tv.setBackgroundColor(selectedColor);
            } else {
                holder.tv.setBackgroundColor(Color.WHITE);
            }
            BluetoothDevice device = myList.get(position);
            holder.tv.setText(device.getName() + "\n    " + device.getAddress());
            return vi;
        }
    }
// part of mainactivity in Blueserial (connection to arduino bluetooth device)
    private class ConnectBT {
        private boolean mConnectSuccessful = true;{
            try {
                if (mBTSocket == null || !mIsBluetoothConnected) {

                    mBTSocket = device.createInsecureRfcommSocketToServiceRecord(mDeviceUUID);
                    BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
                    mBTSocket.connect();
                    SocketSingleton.setBluetoothSocket(mBTSocket);
                }
            } catch (IOException e) {
                // Unable to connect to device
                e.printStackTrace();
                mConnectSuccessful = false;
            }
        }{
            if (!mConnectSuccessful) {
                Toast.makeText(getContext(),  text: "Could not connect to device. Is it a Serial device? Also check if the UUID is correct in the settings", Toast.LENGTH_LONG).show();
                getActivity().finish();
            } else {
                msg( str: "Connected to device");
                mIsBluetoothConnected = true;
            }
        }
    }
//needed for extention
    @Override
    protected void boardReady() throws UnsupportedModuleException {
        setupFragment(getView());
    }
    @Override
    protected void fillHelpOptionAdapter(HelpOptionAdapter adapter) {
    }
    private void setupFragment(final View v) {
    }
}
```



```
package com.mbientlab.metawear.app;

import ...

/**
 * Created by casey on 7/26/2017.
 */

public class SocketSingleton  {

    private static android.bluetooth.BluetoothSocket BluetoothSocket;

    public static void setBluetoothSocket(android.bluetooth.BluetoothSocket socketpass){
        com.mbientlab.metawear.app.SocketSingleton.BluetoothSocket=socketpass;
    }

    public static android.bluetooth.BluetoothSocket getBluetoothSocket(){
        return com.mbientlab.metawear.app.SocketSingleton.BluetoothSocket;
        //return socket;
    }
}
```

Appendix C



```
#define BLUETOOTH_SPEED 57600

#include <SoftwareSerial.h>

// Swap RX/TX connections on bluetooth chip
//   Pin 10 --> Bluetooth TX
//   Pin 11 --> Bluetooth RX
SoftwareSerial mySerial(10, 11); // RX, TX

void setup()
{
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }
  Serial.println("Starting config");
  mySerial.begin(BLUETOOTH_SPEED);
  delay(10);


  mySerial.print("START");
  waitForResponse();

  pinMode(LEDpin, OUTPUT);
```