Wearable Computers for the Quantification of Lower Back Disorder

Team 07

Progress Report

24 October 2016

ECEN 404 – Senior Design Project

Texas A&M University, College Station, TX

Kevin Burns

Benny Chan

Alex Dubois

Benjamin Johnston

# Abstract

Using a combination of wearable inertial sensors and a Microsoft Kinect, our group will create a cost effective system to model the motion of a patient’s lower back in order to quantify the severity of lumbar spinal disorders. This will provide an objective pre-surgical screening, where current methods are largely subjective.

Two small wearable inertial sensors will be placed in line with the patient’s T12 and L5 vertebrae to model integral points on the patient’s back. The patient will be asked to perform simple back exercises to thoroughly evaluate the range of lumbar motion in all planes. The Kinect, a relatively inexpensive motion capturing camera, will be introduced to monitor the patient’s posture and assist the wearable sensors with an all-encompassing look at the subject.

This system would be used in a physician’s clinic room and relatively simple to set up. There are other, more accurate, motion capture systems available that use infrared emission and detection to isolate specific markers on a subject. However, these options consume a large amount of space and cost hundreds of thousands of dollars or more.

This document will further detail the system design, current and future progress of the subsystems, and the code driving the subsystems.

# Table of Contents

[**Abstract**](#_gjdgxs)

[**Table of Contents**](#_30j0zll)

[**1 Project Overview**](#_3znysh7)

[1.1 Proposed System](#_2et92p0)

[1.2 Project Deliverables](#_4d34og8)

[1.2.1 Project Specifications](#_2s8eyo1)

[1.2.2 Significant Direction Changes](#_35nkun2)

[1.3 Project Timeline and Task Ownership](#_1ksv4uv)

[**2 Subsystem Wearable Sensors**](#_3j2qqm3)

[2.1 Significant Changes in Direction](#_1y810tw)

[2.2 Subsystem Specifications](#_4i7ojhp)

[2.3 Subsystem Status](#_3whwml4)

[2.4 Subsystem Technical Details](#_2bn6wsx)

[2.5 Subsystem Testing](#_49x2ik5)

[2.5.1 Stationary Test](#_2p2csry)

[2.5.2 Record Test](#_147n2zr)  
[2.5.2 Drift Test](#_147n2zr)  
[2.5.2 Madgwick Test](#_147n2zr)  
[2.5.2 Record Basic Functionality](#_147n2zr)

[**3 Subsystem Vicon Verification**](#_3j2qqm3)

[2.1 Significant Changes in Direction](#_1y810tw)

[2.2 Subsystem Specifications](#_4i7ojhp)

[2.3 Subsystem Status](#_3whwml4)

[2.4 Subsystem Technical Details](#_2bn6wsx)

[2.5 Subsystem Testing](#_49x2ik5)

[2.5.1 Basic Functionality Test](#_2p2csry)

[2.5.2 Calibration Test](#_147n2zr)

[**4 Kinect Subsystem**](#_3j2qqm3)

[2.1 Significant Changes in Direction](#_1y810tw)

[2.2 Subsystem Specifications](#_4i7ojhp)

[2.3 Subsystem Status](#_3whwml4)

[2.4 Subsystem Technical Details](#_2bn6wsx)

[2.5 Subsystem Testing](#_49x2ik5)

[2.5.1 Kinect Basic Functionality Test Overview](#_2p2csry)

[2.5.2 Kinect Basic Flex and Sagittal Calculation Test Overview](#_147n2zr)

[2.5.3 Kinect vs. Vicon Accuracy Test overview](#_147n2zr)

[2.5.5 Kinect Consistency Test Overview](#_147n2zr)

[**5 Subsystem Signal Correction and Control**](#_3o7alnk)

[3.1 Significant Changes in Direction](#_23ckvvd)

[3.2 Subsystem Specifications](#_ihv636)

[3.3 Subsystem Status](#_1hmsyys)

[3.4 Subsystem Technical Details`](#_41mghml)

[3.5 Subsystem Testing](#_vx1227)

[3.5.1 Signal Synchronization Test](#_3fwokq0)

[3.5.2 Drift Correction Test](#_1v1yuxt)

[**6 Subsystem Feedback Display**](#_3o7alnk)

[3.1 Significant Changes in Direction](#_23ckvvd)

[3.2 Subsystem Specifications](#_ihv636)

[3.3 Subsystem Status](#_1hmsyys)

[3.4 Subsystem Technical Details](#_41mghml)

[3.5 Subsystem Testing](#_vx1227)

[3.5.1 Feedback Display Visibility Test Overview](#_3fwokq0)

[**7 Subsystem Database and Analysis**](#_3o7alnk)

[7.1 Significant Changes in Direction](#_23ckvvd)

[7.2 Subsystem Specifications](#_ihv636)

[7.3 Subsystem Status](#_1hmsyys)

[7.4 Subsystem Technical Details](#_41mghml)

[7.5 Subsystem Testing](#_vx1227)

[7.5.1 Accuracy and Timing Test](#_3fwokq0)

[7.5.2 Precision Test](#_1v1yuxt)

[**8 Subsystem Data Display Interface**](#_3o7alnk)

[8.1 Significant Changes in Direction](#_23ckvvd)

[8.2 Subsystem Specifications](#_ihv636)

[8.3 Subsystem Status](#_1hmsyys)

[8.4 Subsystem Technical Details](#_41mghml)

[8.5 Subsystem Testing](#_vx1227)

[8.5.1 Visibility Testing](#_3fwokq0)

[8.5.2 Input and Updating Test](#_1v1yuxt)

[**4 Conclusions**](#_2u6wntf)

[**References**](#_3tbugp1)

[**Appendix A Project Budget**](#_nmf14n)

[**Appendix B System Testing Code**](#_46r0co2)

[**Appendix C Subsystem Kinect Code  
Appendix D Subsystem Signal Sync Code  
Appendix E Subsystem Database, Analysis, and GUI Code**](#_206ipza)

**List of Figures**

[**Figure 1.1: System block diagram.**](#_1t3h5sf)

**List of Tables**

[**Table 1.1 Specification Compliance Matrix**](#_26in1rg)

[**Table 1.2 Responsibility Matrix**](#_44sinio)

[**Table 2.1 <Wearable Sensors> specification Compliance Matrix**](#_2xcytpi)

[**Table 3.1 <Kinect Motion Capture> specification Compliance Matrix**](#_2xcytpi)

[**Table 4.1 <Vicon Verification> specification Compliance Matrix**](#_2xcytpi)

[**Table 5.1 <Signal Correction and Control> specification Compliance Matrix**](#_2xcytpi)

[**Table 6.1 <Feedback Display> specification Compliance Matrix**](#_2xcytpi)

[**Table 7.1 <Database and Analysis> specification Compliance Matrix**](#_2xcytpi)

[**Table 8.1 <Data Display> specification Compliance Matrix**](#_32hioqz)

[**Table A-1 Project Budget**](#_37m2jsg)

# Project Overview

## Proposed System

The difference between a lower back disorder requiring surgical intervention versus pain management is largely subjective. Our team seeks to develop a system of wearable electronics supplemented by motion capture that will test the severity of lower back issues and quantify the results. This will provide an objective pre-surgical screening where currently there is none.

Lower back disorders (LBDs) are the most common work-related musculoskeletal disorders. This translates to a significant cost to society. However, in 80 to 90 percent of cases, precise diagnosis is not known for disabling LBDs. An objective, quantified measure of the severity of lower back disorders is needed. A proven metric can provide a benchmark for returning to the job without risk of exacerbating the disorder, as well as a delineation between normal variability of pain and a disorder.

Our team will combine the use of wearable inertial sensors and a Microsoft Kinect working in tandem to track this motion. The inertial sensors will deliver the pertinent angular characterization of the lower back, and the Kinect will assist in error correction of the inertial sensors as well as ensure proper patient posture throughout the exercise. This data will be synchronized and corrected for standard sensor error. The corrected data will be analyzed to provide the severity quantification as well as being stored on a database. The motion and quantified information will be shown on a graphical user interface on the operator’s computer and posture correction feedback will be shown on a separate display monitor.

Two wearable inertial sensors will be used in the current system design, worn on the patient under test. The Kinect will interface with these sensors under a single data collection program in order to feed the digital signals into the computer system for data processing. The bridge between the sensor network and the analysis programming consists of the signal correction unit, which will deal with any latency issues from the data capturing network. Additionally, sensor drift is an issue that will be resolved in this subsystem. The data analysis section will utilize information such as acceleration, jerk, and velocity in order to provide a numerical value that applies to said patient’s lower back disorder level. The quantification level of the current system is still under development, and will continue to be dynamic with input provided from the Spine Research Institute of Ohio State University. The data display interface is designed to provide information to a trained technician, to ensure correct capture is being accomplished. This subsystem should not be visible by the patient. The feedback display will interface directly with the patient, providing near real time instruction when an error occurs, in order to attain accurate and useful data. Finally, the system control block will provide the entire system with instruction and error handling, to ensure individual subsystem errors do not crash the overall system.

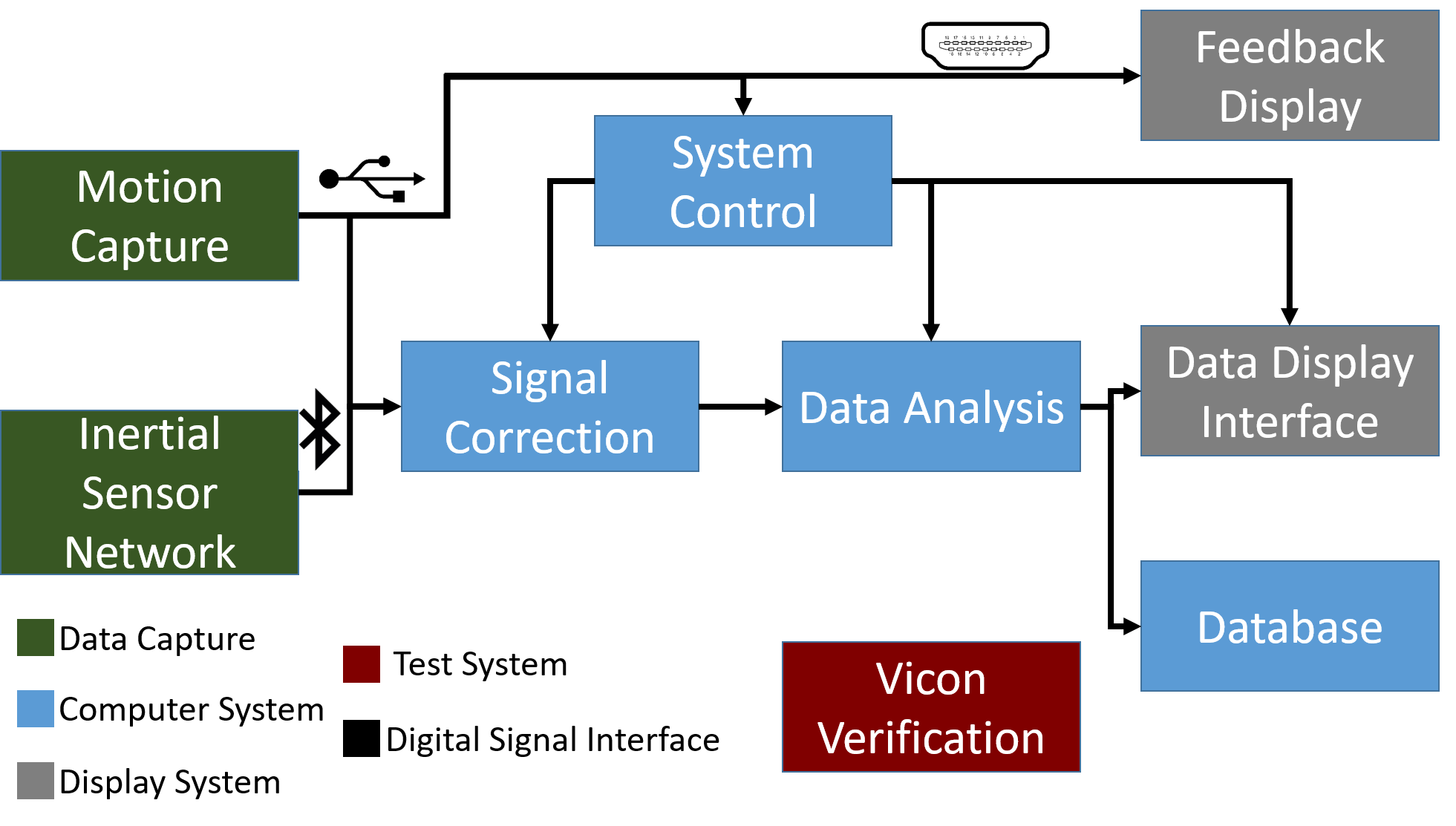


Figure 1.1: System block diagram.

## Project Deliverables

By the end of the project, a fully functional and integrated system that can accurately quantify the severity will be delivered. This system will produce a specific value for this quantification, as well as store collected data in developed database. System hardware will include the inertial wearable sensors, the Kinect, and the computer to run our programs. Additionally, a Vicon motion capture system, located in the Cardiac Device and Mechanobiology Lab, will be used to verify project results. Software implementation will be performed in C# using standard and special libraries, the Kinect API, and Windows Forms framework. The resulting system will contain data algorithms for correction of drift, correction of signal delay, and quantifying analysis. These algorithms will be developed by team members and implemented through the software functions mentioned above.

### Project Specifications

Below is the compliance matrix for the overall system. As can be seen, the system should be run at ambient room temperature to ensure proper comfort for the patient, as well as to keep the system at reasonable temperatures. Additionally the system runs with integrated electronics systems, such as the Kinect and a computer for processing. Therefore standard 120 VAC should be provided for system use. The target for error between the data collected and the “golden standard” of the Vicon should be below 25%, but it is desired to be significantly lower at around 5%.

Table 1.1 *System Compliance Matrix*

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Ideal | Max |
| Temperature requirement | 21º C | 24º C | 27º C |
| Electrical power requirement |  | 120 VAC  60 Hz |  |
| Measure angular data compared to Vicon (% error) | N/A | 5% | 25% |
| Delay between posture error detected and feedback given | > 80 ms | < 100 ms | < 3sec |
| Precision between multiple tests on same subject (LBD value) | 80% | 90% | 100% |

## 

### Significant Direction Changes

The most significant direction change for the overall system is the removal of the calibration exercise previously performed prior to the exercise sets were performed. This exercise previously was to produce a sharp movement in order to determine latency lag between sensors and Kinect, used for correction. Additionally, the sensors were also to be placed on a flat surface without movement to determine inherent drift for the specified sensor, which is also no longer performed.

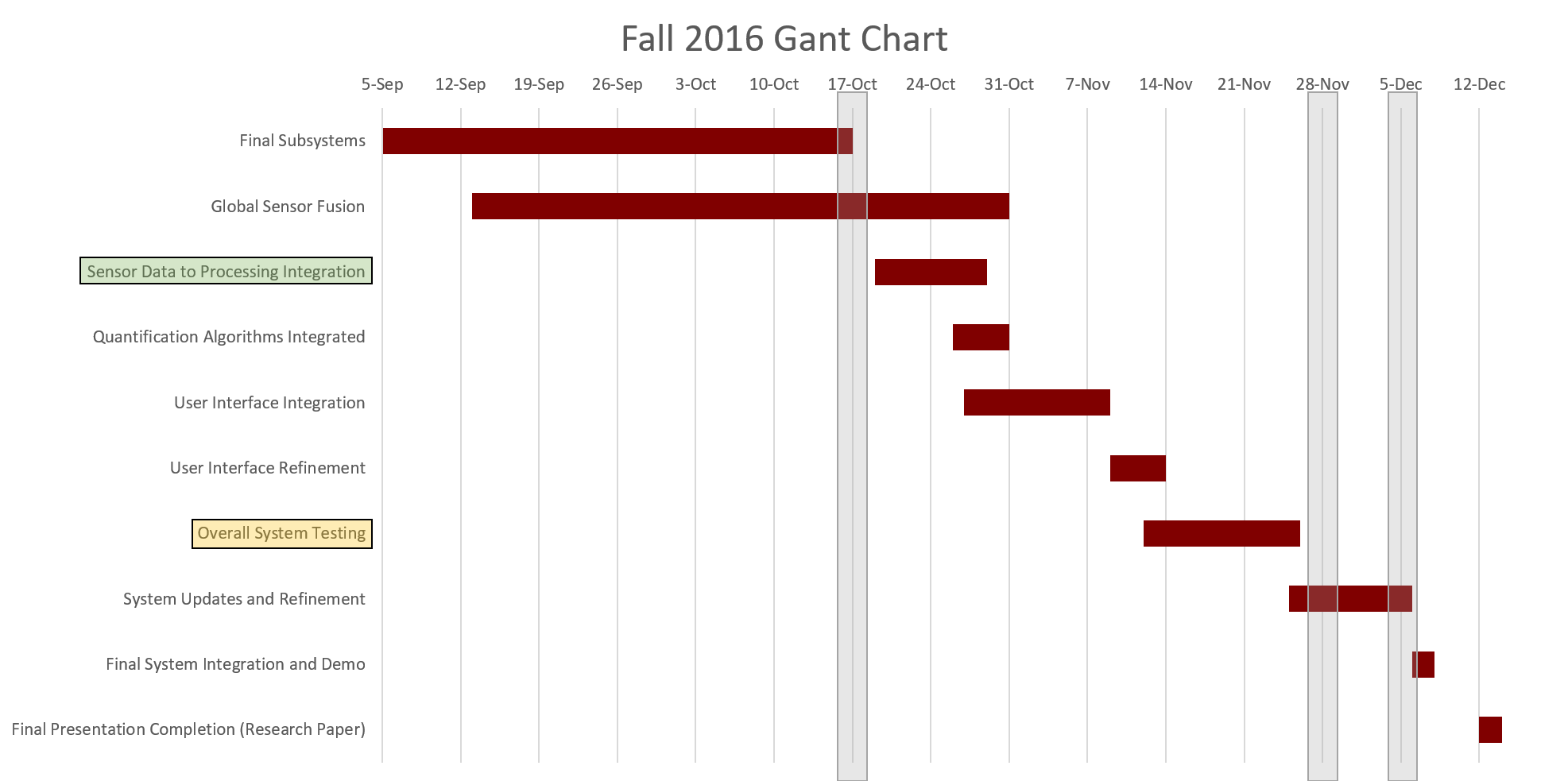
Instead of these exercises, the inherent nature of the exercises to produce sinusoidal data will be leveraged with the Kinect to correct the sensor signals. The latency will be detected between maximum peaks from the Kinect and the sensors. The drift will be similarly adjusted using a comparison between best fit lines between the peaks of the Kinect compared to those of the sensors.

These changes were determined to be necessary for the direction of the project, as unnecessarily quick motions to create peaks could result in injury of a patient suffering from lower back disorder. Another consideration is the reduced amount of time needed to complete a full test battery, as more of the work is shunted to post processing work. This is expected to increase overall project efficiency and flexibility, with little to no cost in accuracy. These changes are not expected to result in a revision of subsystem specifications that they affect.

## Project Timeline and Task Ownership

Table 1.2  
Responsibility Matrix

|  |  |  |
| --- | --- | --- |
| Subsystem | Responsible Engineer | Responsibilities |
| Wearable Inertial Sensors | Ben Johnston | Primary source of data for lower back motion during exercise routine |
| Kinect Motion Capture | Benny Chan | Provide feedback to patient under test and utilize long term trends for drift correction |
| Vicon Verification | Ben Johnston | Accepted standard to compare system results to |
| Signal Correction and Control | Kevin Burns | Correct sensor data for time synchronization and drift correction |
| Database and Analysis | Alex Dubois | Database to contain information gathered by sensors and analysis of data to provide desired parameters |
| Feedback Display | Benny Chan | Display data from Kinect to provide feedback to patient under test |
| Data Display Interface | Alex Dubois | Display data to operating technician |
| Vicon Sagittal Angle Testing | Kevin Burns | Develop MATLAB analysis code to gather sagittal angle from Vicon test data |
| Vicon Flex Angle Testing | Ben Johnston | Develop MATLAB analysis code to gather flex angle from Vicon test data |
|  |  |  |
|  |  |  |
|  |  |  |



Currently the step of getting sensor data to processing is completed, while the remaining steps of integration are stil in the process of being completed. Additionally, subsystem completion has not yet been achieved, with some changes being made after data was collected and analyzed.

## ABET Constraints

### Safety

The problem of diagnoses is also reflected in the societal impact, as well as health and safety. At some point, approximately 12 million people will experience error in diagnoses when seeking medical treatment [Pinnacle]. This greatly reduces the faith of society in its medical infrastructure. Without the trust of their patients, medical personnel cannot perform their function effectively, leaving a key gap in society. This system objectively diagnoses LBD, and therefore can help to relieve the burden placed upon practitioners. This objective system also benefits the health and wellbeing of the patients themselves. In the elderly, for example, there is a 10% risk of a decline in cognitive function after a major surgery [Fines]. This systems objective diagnoses will help to determine with more precision if surgery is necessary or not, which is a major concern for LBD.

### Economic

First and foremost is the economic improvement this system can bring to the medical industry. With a precise diagnoses being impossible 80% to 90% of the time [Marras et al], monetary waste is guaranteed in the treatment process of LBD. These misdiagnoses can result in lawsuits, repeat clinical visits, and waste of medical material, all of which cost significant amounts of money with such a low level of precision being available. It is estimated that $750 billion is wasted, due to unnecessary services and other inefficiencies [Pinnacle]. This system will be able to improve upon the number of wasted services and improve the diagnostic process, saving money for both the patient, and medical industry.

### Manufacturability

This system provides a solution that is relatively inexpensive to manufacture and sustain. The overall cost of this system falls at approximately $250 as noted in the budget section. This is significantly less manufacture cost when compared to other systems usable for accurate analysis of patient motion. The system is also easy to keep operational and is relatively mobile, only requiring standard 120VAC NEMA wall socket input power, and moveable by one person if required.

## Project Standards and Constraints

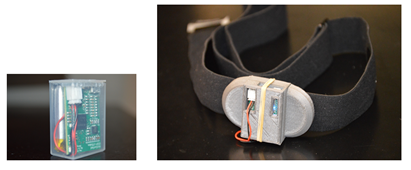
Describe the final system design standards (IEEE standards or protools, such as Bluetooth, Wi-Fi, Zigbee, power, temperature, etc.) and constraints (ambient temperature, lighting, cost, etc.).

## IEEE 802.15.1: WPAN / Bluetooth The IEEE has derived a Wireless Personal Area Network standard based on the Bluetooth™ v1.1 Foundation Specifications.

IEEE 802.3af PoE: Output power 11 W; back up time up to 1 hour;

# Wearable Inertial Sensors

The goal of the project is to deliver a quantitative measurement of a patient’s lower back disorder, based on a characterization of the motion of their back. The wearable inertial sensors provide the bulk of the information for this characterization, and are supported by the Kinect motion capture system. The sensors pass along their data via Bluetooth to the signal correction system for synchronization and error rectification.  
  
The inertial wearable sensor used is the MotionNet sensor, developed by the Embedded Signal Processing Laboratory at University of Texas at Dallas and Texas A&M University. Figure 2.1 shows the sensor in a box, as delivered to the team, and the sensor in the prototype housing and enclosure.

  
Figure 2.1: MotionNet sensor, in Flexatop F-4 box (left), and 3-D printed enclosure (right)

## Significant Changes in Direction

The design for the housing and PCB enclosure have undergone several iterations. Now, the housing has a finalized design and the enclosure has a finalized general design, that fits within 0.1 mm of the PCB. The enclosures are now situated on comfortable heart rate monitor straps, but started as something much bulkier as seen below.



Figure 2.2: Spine Research Institute housing

Initially, the housing was provided by the Spine Research Institute. The housing was the same model used in their research. However this proved too bulky and heavy for the much smaller sensors used in this project. This is seen in Figure 2.2.

## Subsystem Specifications

Table 2.1  
Wearable Inertial Sensors Specification Compliance Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Nominal | Max |
| Capture rate of sensors (in Hz) | 190 | 200 | 210 |
| Maximum mass of 2 IMUs with enclosures (in oz) | 5 | 16 | 40 |
| Maximum PCB movement within enclosure (in mm) | 0 | 0.1 | 5 |
| Captures 3 axis gyroscope, accelerometer, and magnetometer readings per point in addition to local time, machine time, and packet number in (amount of bits) | N/A | 16 | N/A |
| Drift error discrepancy of gyroscopes over a 15 second window  (in degrees per second) | 0 | 0.01 | 0.1 |
| Battery voltage (V) | 3.5 | 3.7 | 4.0 |
| Battery life (Watt-hours) | 1.2 | 1.48 | 2.0 |

The capture rate of the sensors can be erratic. All are set to 200 Hz, but some may capture around 170 Hz. Instead sensors that have an accurate capture rate will be used. If even the captures are not done correctly, one cannot trust other measures to be correct. The enclosures were designed to be smaller than the original sensor system, pictured in Figure 2.2. 40 ounces is the maximum weight that can be comfortably worn on a person. Any movement by the PCB within the enclosure is superfluous and decreases the accuracy of the gyroscopes, so this must be limited. Basic functionality must be met as well in terms of communications and battery life.

## Subsystem Status

The MotionNet sensors are fully functional. They deliver three axis measurements from a gyroscope, accelerometer, and magnetometer with 16 bits per capture and an approximate rate of 200 Hz. The term “approximate” is used for the capture rate because it is not perfectly consistent with each trial, nor is it uniform across different sensors. To ensure that the sensor is performing as it should, tests with average capture rates with 10 Hz difference from 200 Hz should be disregarded. These readings are transmitted via Bluetooth to operating computer. For further specifications, refer to the Design Proposal document. For testing purposes, the data is received and converted to a .txt file by a simple program provided by the ESP lab. Additionally, the program captures video while recording sensor data to show the experiment happening in addition to its numerical results. The gyroscope, accelerometer, and time stamp data is imported into Matlab for testing and developing an error correction model.

The subsystem itself has been operational since last semester. The large change this semester is integrating everything in system testing. Being able to compared the IMU’s fused with the Kinect against the Vicon is invaluable. The drift correction model now incorporates the peaks of the Kinect to correct the offset. The only change to be made is incorporating a scaling error check and fix.

The sensor housing is complete. Six Polar Soft Straps have been purchased for securing the sensors to the patient. The straps are durable, stay secured to the patient (better on skin, viable over clothing) flexible, adjustable from a waist size of 32 inches to 46 inches, and will use buttons to snap to the enclosures.

The enclosure prototypes were 3D printed at the Engineering Innovation Center at Texas A&M. The 3D printers available at the EIC are sufficient for prototyping, but the final product comes with too many minor deformities for the small, precise details required (walls two millimeters thick and some smaller features still). However the current prototypes are sufficient for testing so they are a success for now. The first enclosure model has too strict of tolerances and was not designed to button to the Polar Soft Straps, and is much smaller in size as a result (eight grams of material used). The next iteration of the enclosure design had sufficient tolerances and was designed to snap to the housing, but additional space was not allocated for the battery. This is significant as the battery is nearly the size of the PCB itself. This model was nearly twice the size of the first (14 grams of material used). For the final and current prototype, a space between the PCB and housing was allocated for the battery, which increases its size (24 grams of material used), but is now fully functional and ready for use with testing. Figure 2.4 shows the 3D models of each design and Figure 2.5 shows the resultant printed enclosure.

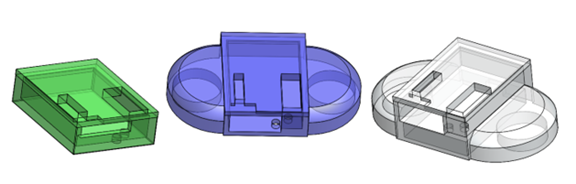


Figure 2.4 3D Models

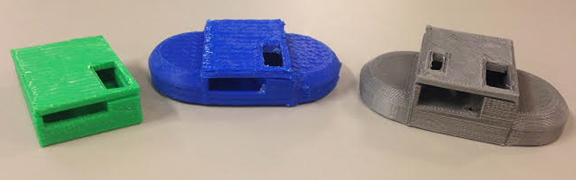
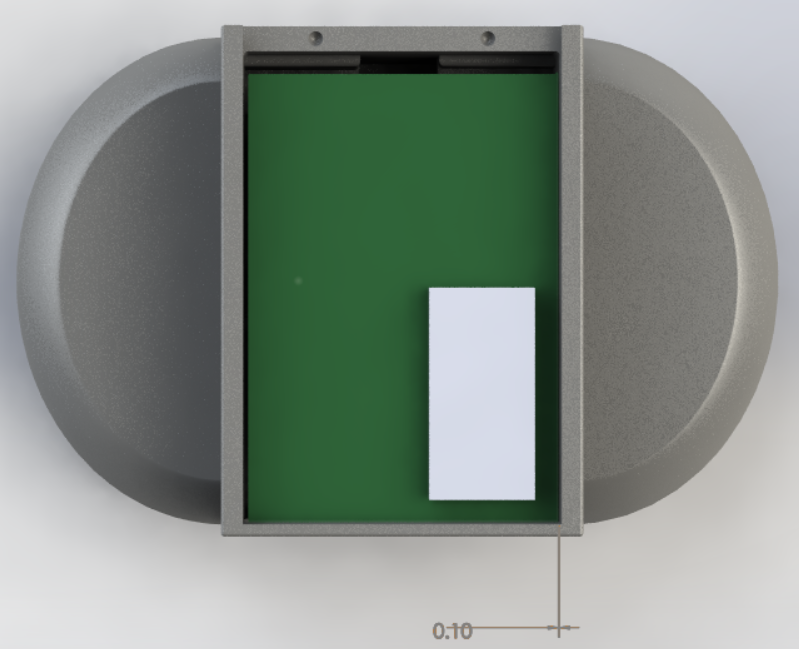
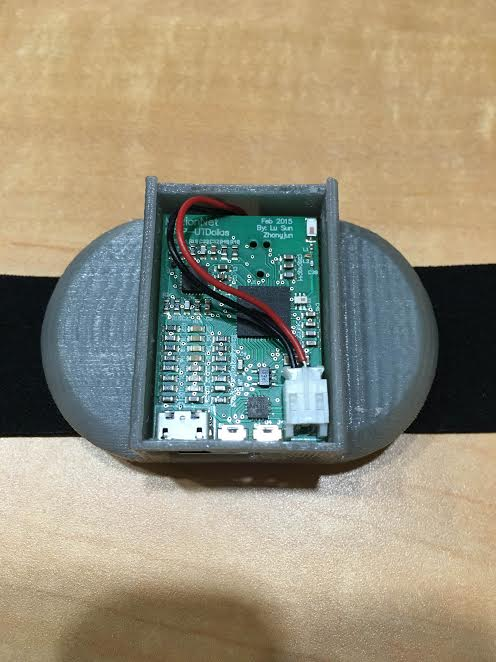


Figure 2.5: Printed Enclosures

Once the prototypes were created, the next phase of enclosures were created using the architecture department’s 3-D printer. The new 3-D printer uses much better quality material and has higher resolution. The final model is fitted to the PCB and is show below rendered in SolidWorks and pictured.



SolidWorks Rendering of Enclosure



Printed Enclosure, open for clearance reference

## Subsystem Technical Details

“Drift” is a function of change in battery voltage, temperature changes on the PCB, and rapid motion experienced by the IMU’s. It can be seen most clearly as an offset of the angular velocity reported by the system’s gyroscopes. This offset is relatively constant, as proven in section 2.5, though it shifts slightly over time. To find angular distance from angular velocity, the velocity is integrated.

(1)

(2)

(3)

When drift is viewed as a constant, this results in an unwanted sloping of the angular distance measurement, as seen in Figure 2.7. In reality, due to the small shift in drift as time progresses, the impact of drift on angular distance is exponential. This leads to the desire for a correction model that changes as drift changes over time. The preliminary attempts at such a model are shown in section 2.5.5, windowed testing.

In addition to draft characterization, the gyroscopes will be supported by IMU sensor fusion and the Kinect through the use of Kalman filters. Sensor data must first be synchronized and converted to quaternion form for simple interpolation. Quaternions are a 4-D complex set of numbers used to represent rotation in 3-D.

(4)

(5)

(6)

(7)

Where the end result is the rotation as a quaternion in the form:

(8)

(9)

This representation now easily updates position points (also in quaternion form) by using the equation:

*(10)*

This can be used, for instance, to rotate the original axes to be in line with a standing subject using gravity as read by the accelerometers. Now that quaternion form has been attained, a Kalman filter may be applied.

The Kalman filter algorithm produces estimates of the true values of sensor measurements and their associated calculated values by predicting a value, estimating the uncertainty of the predicted value, and computing a weighted average of the predicted value and the measured value. The most weight is given to the value with the least uncertainty. The estimates produced by the algorithm tend to be closer to the true values than the original measurements because the weighted average has a better estimated uncertainty than either of the values that went into the weighted average [5]. Code has not yet been written to put a Kalman filter into effect, but that will be accomplished in the coming months.

All of the work shown above with quaternions is necessary to understand the advanced filtering shown in the “Madgwick” filter test below. This represents a large time commitment and something that was worked towards since last semester. Unfortunately, for the reasons listed below this will not be able to be used, but does represent a fair amount of the work done on the project.

The PCB enclosures for the MotionNet sensors are required to secure them to the housing straps, as well as guard against the PCB jostling within the enclosure, adding superfluous rotations to the data. There were property rights issues with directly obtaining the PCB files used to make the MotionNet sensor, and instead the gerber files, used for board fabrication, and bill of materials were given. The top and bottom silk overlay gerber files were imported into Altium to recreate new PCB. The largest, or protruding parts were determined and their corresponding 3D STL files procured from their manufacturer's websites. All other parts have a maximum clearance of 1.5 millimeters, which was taken into consideration when designing the enclosure. These 3D models were lined up with their approximate actual position, using the silk overlay lines as a guide, this is seen in Figure 2.8.

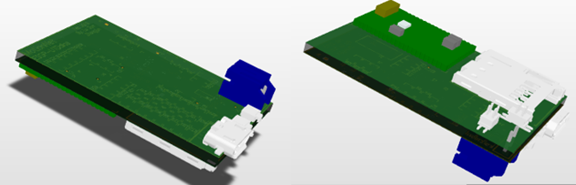


Figure 2.8: 3-D model of the MotionNet circuit board, excluding discrete components, top view (left), bottom view (right)

This Altium 3-D model was then exported to Solidworks to design an enclosure around the devices of the circuit board. This resulted in the enclosure previously shown.

## Subsystem Testing

Give a summary of the important testing here, followed by the test plans (in order). An example of the structure of a test plan is shown below. These should be filled out, with data and test conclusions.

### Stationary Test Overview

The gyroscopes of the IMU are quite accurate except for their inherent “drift.” Drift is caused by a number of hardware issues such as the change in battery voltage and temperature of the device and is seen as recorded motion that does not actually exist.

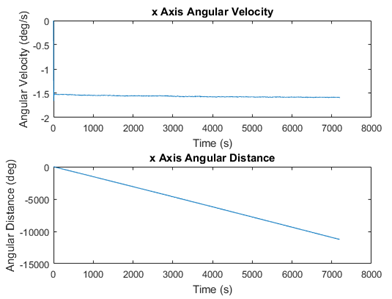
A test with no movement at all is a necessary control experiment to understand the inherent offset of the gyroscopes and thus monitoring drift error over time. Ideally, the sensors would read zero for all angular velocities and angular distance traveled measurements. As these sensors are not ideal, this will not be the case, and will aid in the understanding of drift error.

#### Test Setup

The sensor is placed on an undisturbed surface and run for 7199.2 seconds (~two hours). The sensor is fully charged and then unplugged for the duration of the test. The sensor used is 209.

#### Data

Figure 2.11 shows the x axis angular velocity capture by the gyroscopes, as well as the integrated angular distance. The recorded mean angular velocity is -1.5563, with an expected velocity of 0. The final x axis angular distance is 11293 degrees, with an expected distance of 0.



Gyroscope Drift Over Time

#### Test Conclusion

There is a near constant velocity of -1.5 degrees per second recorded. When this near constant offset is integrated over time, the result is a very clear depiction of drift error. It is safe to say that in this instance the offset is constant as the velocity only changes 0.0551 degrees over a span of two hours. This should mean that merely offsetting the gyroscopes’ readings by a known constant offset should nearly eliminate error due to drift.

### Record Player Test

The stationary test does not provide a meaningful demonstration of the proposed drift correction, so now a motion test is run to provide context, and test if the current theory holds while in motion. A sensor is placed on a rotating record player for this test, as seen in the figure below.

#### Test Setup

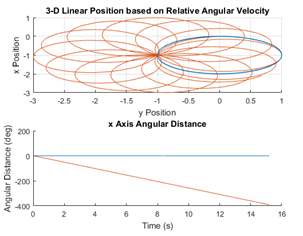


Record Player Test Setup

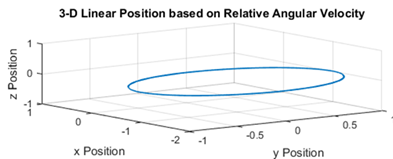
The sensor is placed on a rotating record player (rotates on z-axis). The duration of the test is 15.18 seconds. The sensor used is 209.

#### Data

The plotted data in red shows the filtered, but uncorrected for drift, data from the gyroscopes. Drift causes the uncorrected data to appear as though the record player is rotating in the x and y axes, in addition to the z axis. This is shown in the first figure below. When corrected, the spin in the x and y axes is unappreciable, as expected, and 3-D position plot takes on a disc shape, as expected. This is seen more clearly in the second figure below.



Uncorrected Record Player Data



3-D Position Plot of Corrected Record Player Data

#### Test Conclusion

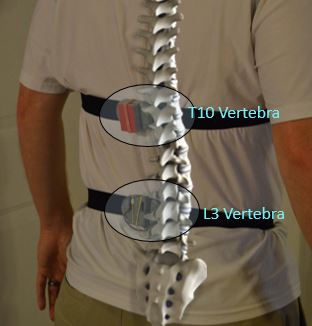
Over a fifteen second window (length of individual patient test), the drift correction model appears to still be viable as the results are exactly as one would predict.

### Drift Correction Test

The previous corrections for drift required calibration before each instance of use, and were less accurate as time went on until recalibrated again. The new method of drift correction uses sensor fusion with the Kinect to determine a proper average angular displacement. The Kinect can be inaccurate in the short term due to obstructions and other inherent errors in the system, but notably does not drift. This means that a long term average can provide an appropriate standard to correct the drift to.

#### Test Setup

* + - 1. All testing is now performed in the BEL lab of the Emerging Technologies Building by the permission of Dr. Madigan. This is done so that the accuracy of all test can be measured against the high cost, high accuracy motion capturing system, the Vicon. The comparisons to the Vicon will be discussed later in the Vicon Verification section. Two IMU’s are strapped around the subject’s midsection in line with the T10 and L3 vertebrae as seen below:

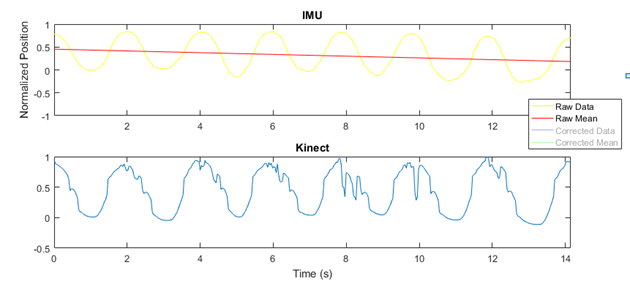


Proper sensor placement

The patient then performs the trunk flex exercise, while being monitored by the IMU’s, Kinect, and Vicon. IMU drift will occur, but will be corrected in accordance with the Kinect.

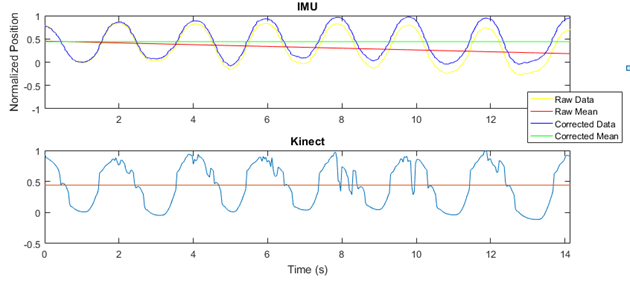
#### Data

As previously described, the short term data of the IMU’s is accurate, smooth motion is tracked without interruption. But, when shown with a best fit line in red, the average is clearly falling. The Kinect’s data is untrustworthy in the short term, the motion tracked is not smooth, but the long term average remains consistent and does not experience drift.



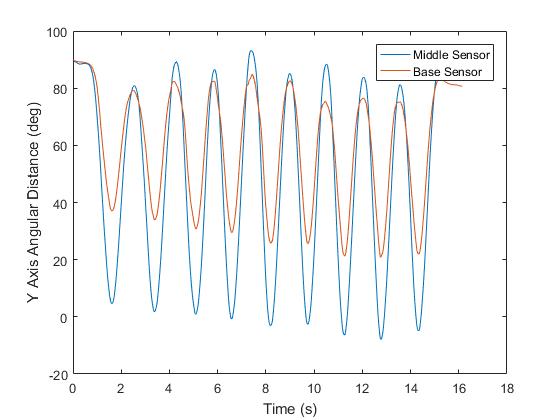
Raw Data Without Correction

Armed with this information, the mean of the Kinect data can be determined, and each individual point offset by the discrepancy between the Kinect mean and the IMU best fit line. This yields the blue corrected data with the green corrected mean seen below. In effect, the IMU no longer experiences drift.



Corrections Based on Kinect Mean Displacement

More can be done to compare the IMU worn on the middle of the back to the IMU worn on the base of the back. Alternatively, the base IMU may prove unnecessary, which would be an advantage as the patient would be less encumbered and the tests would take less time to set up. The results of the same data as before but compared from the mid IMU to the base IMU can be seen below:



Mid and Base Sensors Simultaneously Tracking Angle

Clearly, drift is experience with both sensors. The base sensor sees less rotation than that of the mid sensors, which is understandable given its position on the back. The question is how to use this information to aid the mid sensor, which would bring the accuracy of it even higher, and surpass specifications even more.

#### Test Conclusion

This new method of correcting the IMU drift by comparing to the Kinect mean saves time and effort for all involved, as well as being more accurate over longer periods of time. The one potential disadvantage at this point is that the Kinect mean does not line up with the Vicon mean, because the Kinect is obstructed as the patient moves down a full 90 degrees with respect to the z vector (or straight up in the air). This is being looked into and is only one small knock against an otherwise very successful fix for a sophisticated problem.

### Madgwick Filter Test

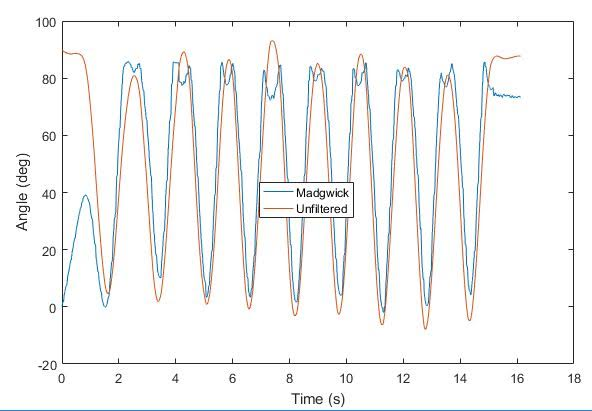
This test is conducted in the same way as the Kinect drift correction. This is done with the IMU’s, Kinect, and Vicon tracking a subject performing the Trunk Flex exercise.

#### Test Setup

This test is conducted in the same way as the Kinect drift correction. This is done with the IMU’s, Kinect, and Vicon tracking a subject performing the Trunk Flex exercise.

#### Data

The unfiltered results can be seen in orange and the filtered data seen in blue. The initial discrepancy is expected, as the filter takes a few seconds to align axes with the data, but the differences beyond that are not expected and must be corrected.



Madgwick vs Unfiltered Angular Displacement Over Time

#### Test Conclusion

The results are not yet as expected. The Madgwick filtered data should be even smoother than the unfiltered data, but instead has unexplained concavity in the peaks. The PhD students in Dr. Jafari’s Embedded Signal Processing lab will be consulted on the meaning of these results.

### Basic Functionality Test

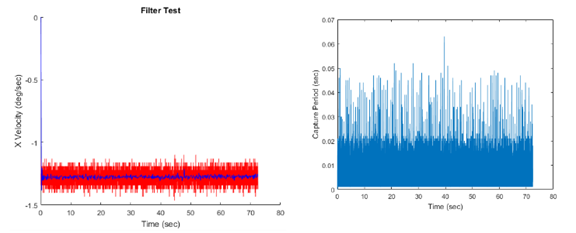
Testing the basic functionality of the sensors is actually a combination of several minor tests: ensuring the capture rate is consistently around 200 Hz, ensuring that the digital noise filter is sufficient, as well as ensuring that the sensor is capturing data.

#### Test Setup

* + - 1. All of the requirements may be met with any of the test formats. The digital noise filter is a third order low pass IIR filter. For this example, stationary testing was used. The data is from a stationary test of an IMU for approximately 72 seconds. Sensor 23D’s battery was charged until full, unplugged, and the test started immediately.

#### Data

The x-axis reading from the gyroscope was arbitrarily chosen and filtered to demonstrate the noise filter. The red line is the raw unfiltered data, the blue line is the filtered data. On the right, the period of time between captures as reported by the local machine time of the operating computer is graphed in the figure below.



Basic Functionality IMU Test

#### Test Conclusion

The sensors are obviously collecting desired information, so they are successful in that regard and operation. The digital noise filter successfully filters noise from the raw signal. The capture period as reported by the computer is erratic, but the average frequency is 201.0516 Hz. Additionally, some timestamps are exactly the same as ones adjacent to them, causing an infinite frequency. It is for this reason that in all subsequent tests time is modeled based on average frequency rather than the individual timestamps.

# Vicon Subsystem

Simple testing can be done with known parameters such as a stationary test, motion recorded on a record player, or motion recorded on a swing. But more complicated tests do not have a known baseline for comparison. The Vicon motion capture subsystem uses an array of eight infrared cameras pointed to a central platform to accurately capture the full range of motion of the subject, as seen in Figure 4.1. This will be the basis on which the successes of the sensors will be quantified.

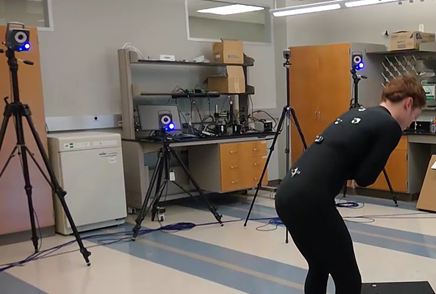


Figure 3.1: Vicon system in Cardiac Device & Mechanobiology Laboratory at Texas A&M University

## Significant Changes in Direction

Some Vicon systems are capable of inaccuracy in position measurement of only 15 microns [10]. However, the system available to the team has several issues that result in that same inaccuracy being up to one millimeter, per Achu Byju the lead technician responsible for the system. This does not change the direction of this subsystem, rather the expectations of it. The Vicon is still viable as a standard for the sensors, but the confidence in the system is hindered.

## Subsystem Specifications

Table 3.1  
Vicon specification Compliance Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Nominal | Max |
| Inaccuracy in position measurement | 15μm | 100μm | 1mm |
| Calibration time (in minutes) | 4 | 5 | 6 |
| Resultant image error (coefficient) | N/A | 0.1 | 0.3 |
| Operational infrared filtering and capturing cameras | N/A | 8 | N/A |
| Reflective dots on subject (it is necessary that these are the only points tracked) | 4 | 5 | 6 |

The Vicon system is fully functional, and is used for monitoring full system tests. The system does require a certain amount of calibration is order to capture successfully. The calibration wand should be rotated in front of all cameras for four to six minutes. This should result in an image error of less than 0.3, but ideally less than 0.1. If the required image error is not met, the calibration must be repeated. This process takes approximately 5 minutes.

## Subsystem Status

The Vicon has always been operational, though there was a learning curve in operating it. It is used for system testing, but no additional work is required. It is useful to mention the Vicon, and explain its utility though, given that it is central in reporting the success of the project.

## Subsystem Technical Details

Vicon motion capture is an infra-red marker tracking system. The system used for this project consists of 8 cameras with IR optical filters, an array of IR LEDs, and 5 reflective dots. All other light is filtered so that the cameras only detect the reflective markers

The reasons for the inaccuracy of this specific Vicon system are a combination of positioning of the cameras, the reflectivity of the floors and windows, and the older technology being used. The reflections seen from other cameras or reflective surfaces in the room cause blind spots in the field of vision of the cameras where tracking is not possible, as seen in Figure 3.2. The captured motion can be interpolated to fix this lost data manually by an operator, which is a time consuming process. Desired data must be carefully considered prior to capture in order to make an efficient use of time.

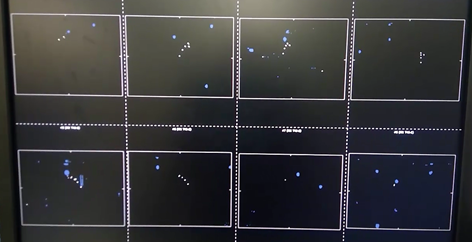


Figure 3.2: Camera view of the Vicon system. Blue spots are blind spots, white spots are markers

## Subsystem Testing

As mentioned, the Vicon is fully functional, but is used in conjunction with system testing. There are a few basic functionality tests for the Vicon, though.

### Basic Functionality Test

The purpose of this test is to ensure that the Vicon system is functioning properly and will be ready for the all important gold standard in the future.

#### Test Setup

The Vicon must first be calibrated, as mentioned before, and the origin is dictated by placing the calibration wand in the center of the platform. Test, subject, and joint information are entered into the system to organize the database of results. The capture is begun when the operator is confident these steps have been followed carefully.

#### Data

The results from an individual trunk flex test may be found in the IMU section. This contains the position data for each recorded point and edge, an example of which can be seen below.

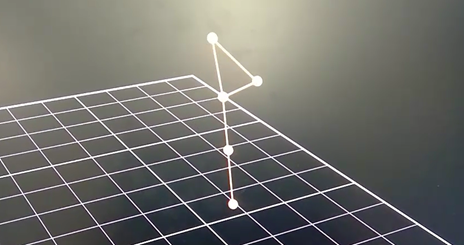


Figure 3.2 GUI View of Vicon Data

#### Test Conclusion

The subsystem is working as intended. Clearly, the calibration is sufficient and the system is capable of collecting data which can be exported to Excel, Matlab, or the C# program designed for this project.

### Calibration Test

Calibration of the Vicon motion capture system is required each time that it is turned on. This calibration remains valid between tests in the same set, but not after being turned off. The calibration takes around 5 minutes, and reports its results in terms of image error coefficients.

#### Test Setup

The calibration wand is waved around all 8 cameras until they have all collected 2000 points of data. The motion is described in the manual as “painting the walls horizontally.” It is possible for a camera to be given too much data during calibration, so care must be taken to evenly distribute the motion of the wand to ensure success. As previously mentioned, after around five minutes of performing this action, all of the cameras will have ceased blinking and the calibration is complete.

#### Data

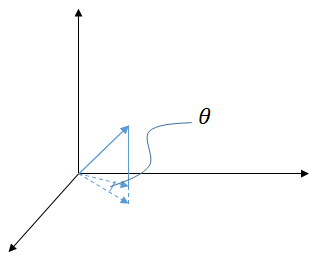
Even if the calibration is complete, there is no guarantee of it being successful. Each camera must have an image error coefficient of less than 0.3, but less than 0.1 is preferred. If these conditions are met, then the accuracy of the Vicon is confirmed and tests are ready to begin. The lead technician, Achu Byju, has confirmed that the maximum position error of each individual point of this particular Vicon system is one millimeter. High end Vicons can boast a maximum error of 15 microns, but one millimeter is sufficient for the scope of this project.

#### Test Conclusion

Basic operation and calibration of the Vicon has been confirmed which is all one can say for this subsystem. Our group has begun overall system testing which relies heavily on the Vicon. This will be discussed further in some of the individual subsystem test plans, and in more detail in the upcoming system test plan.

### Sagittal Plane Testing

This test shows the sagittal plane calculations derived from the Vicon, for comparison against the IMU data. This data was derived using three dimensional vector mathematics, and therefore requires some explanation on how it was collected. Below is an image showing exactly what is desired.



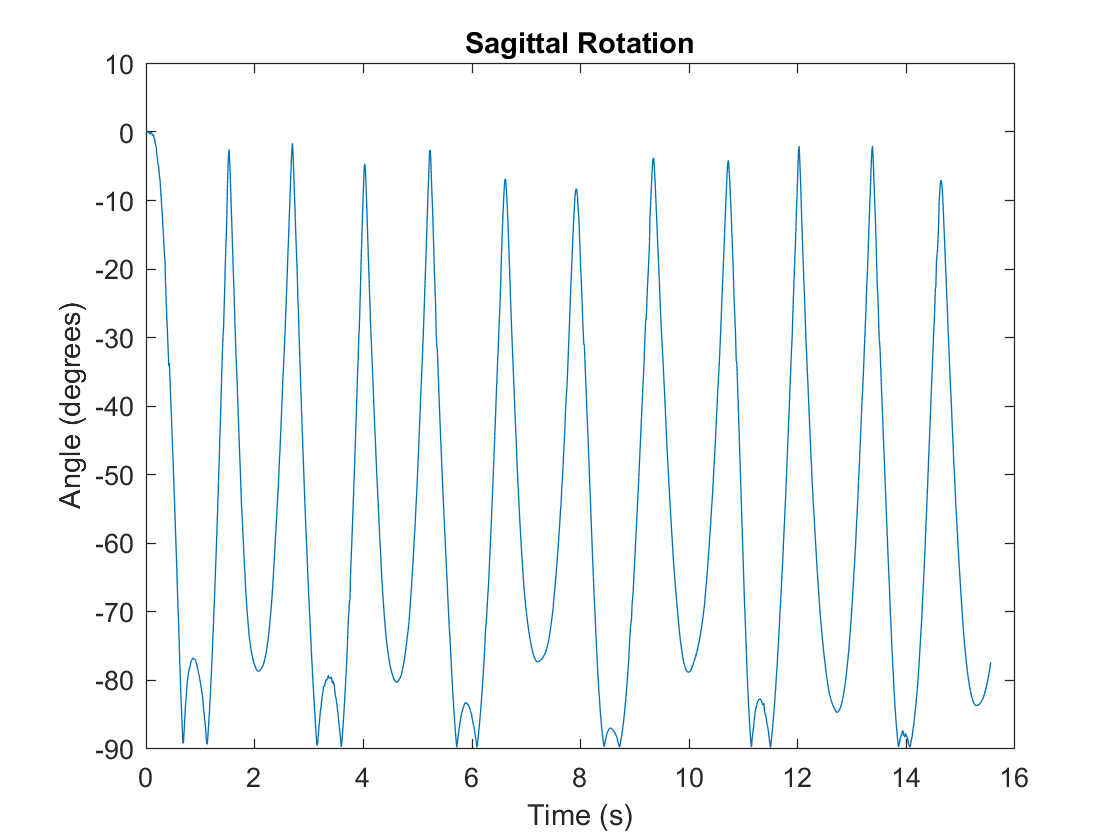
In order to find this “theta” value given only the solid free vector, it was necessary to treat the first x-component of the vector as the zero reference point. This allowed each subsequent data reading to be compared to this reference to determine what the angle was. The angle was found by first normalizing the vectors and then using the arccosine function on the vector dot product. This results in the angle between the two x-component vectors.

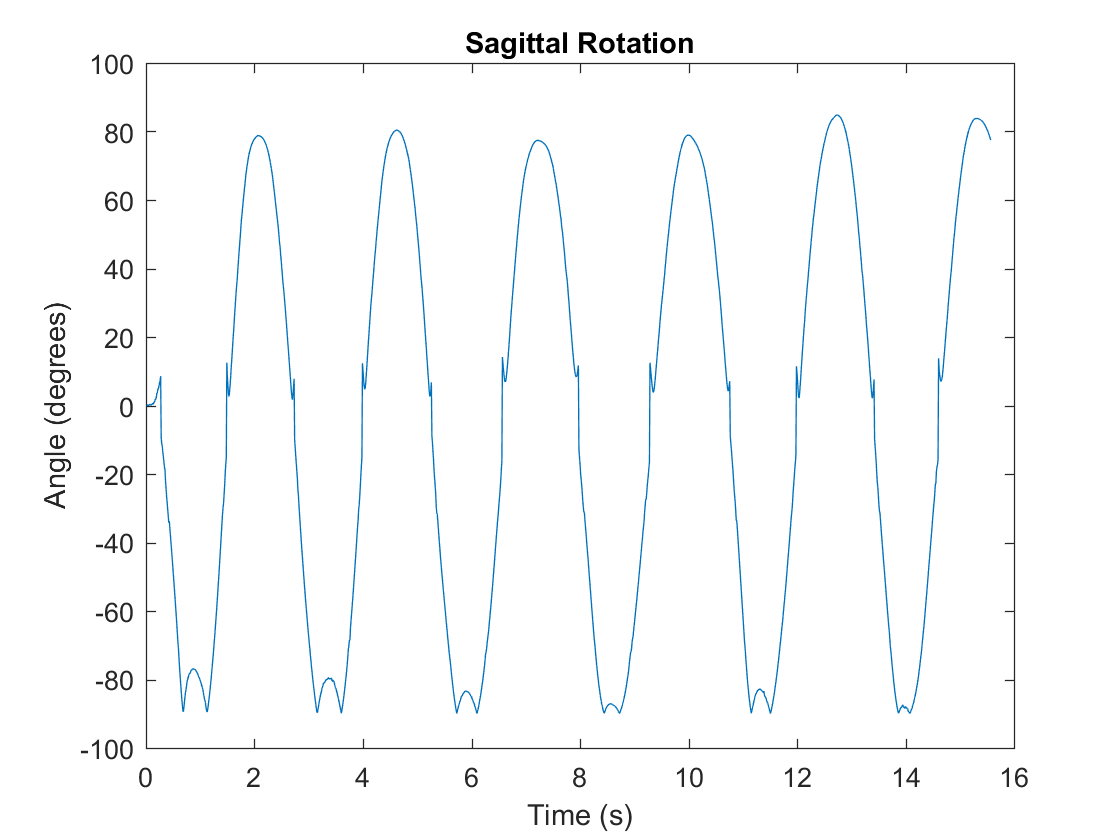
#### Test Setup

The test setup was the same as used in overall system testing. The Vicon markers were placed on the specific points of the patient’s body, and then data was recorded performing rotational motion with the z-axis as the rotational axis. A 15 second trial was performed.

#### Data

* + - 1. The first data plot shows the sagittal rotation, but it shows it being rectified along the 0 point, which should not happen.



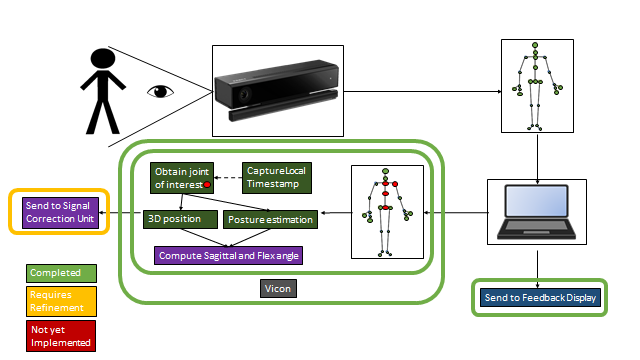
* + - 1. This is due to the angle always being measured in a single direction due to the nature of the arccosine function. This issue was resolved with a simple conditional statement that multiplied the signal by -1 when it was greater than the reference value, detailed above. After this correction was in place, the data plot below could be observed.
      2. 

#### Test Conclusion

Some additional adjustment to the scripts used to gather these plots is still needed in order to ensure fully accurate data is accessible. However the core functionality of the scripts is shown to be working properly by this test. As more system integration testing is performed, further refinement will be performed on this script.

# Kinect Subsystem

The purpose of the motion capture is to provide consistency in the overall system compared to the wearable sensors. This is going to measure a set of joint data to acquire the 3D positional data, and give a posture estimation to verify the correctness of the patient’s posture. This will receive an input from the patient’s movement during test and it will capture the joint of interest to have a posture estimation and the 3D position with a local timestamp. In addition, it will compute the sagittal and flex angle, which it currently returns a custom data structure as a text file to analyze it in Matlab, in order to compare it with the Vicon.



*Figure 3.1: Kinect Block Diagram*

## Significant Changes in Direction

Overall, the main purpose of this subsystem has remained unchanged after its definition in the proposal. However, we have decided to change the direction in which the overall positive peaks will be used to help correction the drift of the IMU instead of the average of the Kinect. In addition, we will be utilizing multiple trials throughout a patient test in order to average out and potentially throw away any erroneous data collected throughout a single trial.

## Subsystem Specifications

Table 3.1  
Kinect specification Compliance Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Nominal | Max |
| Capture rate of Kinect (in fps) | 28 | 30 | 32 |
| Angle, angular velocity error with respect to Vicon | <5% | <10% | 15% |
| Angle offset compared with Vicon (in degrees) | 3 | 5 | 15 |
| Motion capture will ensure proper posture control, within expected repeatable limit specified | 95% | 98% | 100% |
| Kinect motion capture should be positioned horizontally and vertically to optimize capture accuracy |  | 2-3 feet above ground  2.5-3.5 feet from patient |  |
| Device must provide repeatable results for posture control and joint measurements | +/- 5% | +/- 7% | +/- 10% |
| The feedback display should be able to be clearly viewed by the physician from the monitor. This means they can easily view any text, buttons, or filters on the GUI display | 1 ft. | 1.5 - 2 ft. | 2.5 ft |
| The feedback display should have a refresh rate which is that of a standard monitor to allow for clear viewing of the GUI | 30 Hz | 60 Hz |  |
| The display should provide feedback to patient within specified time period | > 80 ms | < 100 ms | < 3sec |

As mentioned in the overview, this subsystem is tasked to provide consistency in the overall system compared to the wearable sensors to aid quantify the severity of Lower Back Disorder within the patient. To test the overall performance of this subsystem, the flex and the sagittal angles of the Kinect are compared against our golden standard of testing, which is our Vicon subsystem. By using the incredibly accurate vicon, we can check the accuracy of the data we are collecting and computing in order to ensure that the quantification metrics are as accurate as possible with the model of Lower Back Disorder we have created.

Other metrics include the precision of this quantification which should remain as constant as possible for the same patient in order to ensure that the same patient is not provided wildly different results and potentially give incorrect severity quantifications. Lastly, we wish to measure how long this overall process takes in order to ensure that patient testing can be performed with minimal time overhead for this subsystem.

## Subsystem Status

Currently the Motion capture subsystem is functionally complete. It gathers the 3D positional data of a set of joint of interest with respect the Kinect and outputs a .txt file, which includes the X, Y and Z axis. In addition, it includes a local timestamp, which will be used for the integration of the whole system. While having the 3D positional data, we also computed the sagittal and flex angle. Also, the Kinect subsystem is already integrated with the IMU subsystem, and we can obtain data simultaneously.

In addition, test have been made with the Vicon, which is our gold standard, to check the accuracy of the data we are collecting and computing in order to ensure that it falls within the specifications of the subsystem requirement. Also, while we were testing this subsystem, a timer and a directory structure were implemented to facilitate the gather of data with little overhead.

As of right now, the Kinect subsystem is being integrated with the database and analysis subsystem to have one piece of software to encompass all functionalities of the final product.

## Subsystem Technical Details

The Microsoft Kinect is a device for motion sensing for the Xbox One video game consoles. It is based on a webcam style peripheral, which it enables the user to control and interact through a natural user interface using gestures. However, the Kinect was not developed with the intention of clinical use. As such, the accuracy of the Kinect measurements must be evaluated thoroughly for movements of interest before clinical application.

For the first programming block, it was necessary to create a class called KinectBodyView, which allows the Kinect to create a set of joints and track the skeletal raw data of the 3D position of the body. This is initialized main window of the GUI to be able to visualize the programing block.

After obtaining the skeletal raw data, it was necessary to obtain the joints of interest, which then we would use to compute the sagittal and flex angle. For example in the following code:

Code 3.1 Joint storage into lists.

List<float> spineMidData = new List<float>();

spineMidData.Add(body.Joints[JointType.SpineMid].Position.X);

spineMidData.Add(body.Joints[JointType.SpineMid].Position.Y);

spineMidData.Add(body.Joints[JointType.SpineMid].Position.Z);

To compute the sagittal and flex angle, we have created a class called KinectFeedback. For this, we wrote a function which calculates the angle of two vectors through the dot product as seen in the next sample code:

Code 3.2 Computation of dot product with respect two vectors.

private float DotProduct(List<float> vector1, List<float> vector2)

{

float dotProduct = 0;

for (int i = 0; i < vector1.Count; i++)

{

dotProduct += (vector1[i] \* vector2[i]);

}

return dotProduct;

}

Finally, it would display the angle by calculating the cosine of the dot product as seen in the next sample code:

Code 3.3 Computation of flex angle

public float CalcFlexAngleWithRespectToInitialPos(List<float> jointPos)

{

List<float> vector0 = new List<float>();

List<float> vector1 = new List<float>();

for (int i = 0; i < jointPos.Count; i++)

{

float posDiff = jointPos[i] - initialPosSB[i];

vector0.Add(posDiff);

}

for (int i = 0; i < jointPos.Count; i++)

{

//0 is the reference point of the Kinect

float posDiff = 0 - initialPosSS[i];

vector1.Add(posDiff);

}

float dotProduct = DotProduct(vector0, vector1);

float magnitudeVector0 = CalcMagnitude(vector0);

float magnitudeVector1 = CalcMagnitude(vector1);

float cos = dotProduct / (magnitudeVector0 \* magnitudeVector1);

flexAngle.Add((float)(Math.Acos(cos) \* (180 / Math.PI)));

double angle = Math.Acos(cos) \* (180 / Math.PI);

flexAngleTxt = ((float)(angle)).ToString();

currentFlexAngle = (float)(angle);

return (float)(angle);

}

## Subsystem Testing

# Kinect Basic Functionality Test Overview

The purpose of this test is to verify the general operation of the Kinect and test its main functionality, which it will then provide consistency in the overall system versus the Vicon system.

#### Test Setup

First, we have to set up the Kinect system. The Kinect is placed in front of the subject far enough away in order to capture their whole body. Now, you bring up our C# program through Visual Studio and run it. The user hits start recording in order to collect data into a set of text files and stop record to close them.

With all of the data we require collected, we begin processing through a custom Matlab program. The Matlab program parses all of the Kinect positional data and then it is output by the program.

#### Data

## https://lh6.googleusercontent.com/L8l2r4lhL08iJ0ySl8yBJvj_7okNqUYBhZl0lSEqK01Vimj0Q1Y9FKc7zWP5hM9uXJqZI0xoZEeeJWSQrohMvmN8y5H-GuRLiF78Gb1cLuJ9ou6lvC1Qp3AI2i4lWHnb4jnF3aIAVWo

Fig. Spine Shoulder Positional Data from Kinect

As you can observe in the figure above, it can be seen that as for a specific joint, in this case the Spine Shoulder, we were able to gather the positional data with respect of the Kinect as a point of reference. The unit of each position is given in meters and the x-axis is the difference between the beginning and the end of the recording.

#### Test Conclusion

The Kinect is obviously collecting desired information, so it is successful in that regard and operation. However, some timestamps are exactly the same as ones adjacent to them, causing an infinite frequency. It is for this reason that in all subsequent tests time is modeled based on average frequency rather than the individual timestamps.

# Kinect Basic Flex and Sagittal Calculation Test Overview

The purpose of this test is to verify the general operation of the Kinect and test its main functionality, which it will then compute the flex and sagittal angle with respect of the subject positional data.

#### Test Setup

First, we have to set up the Kinect system. The Kinect is placed in front of the subject far enough away in order to capture their whole body. Now, you bring up our C# program through Visual Studio and run it. The user hits start recording in order to collect data into a set of text files and stop record to close them and start performing our angle calculation.

With all of the data we require collected, we begin processing through a custom Matlab program. The Matlab program parses all of the Kinect flex and sagittal data and then it is output by the program.

#### Data

## 

Fig. Flex and Sagittal Angle from Kinect

This test consist on the subject flexing as much as he/she can, while maintaining a sagittal angle of 0 degrees. As you can observe in the figure above, it can be seen that we were able to gather the flex and sagittal angle with respect of the Kinect as a point of reference.

#### Test Conclusion

The Kinect is obviously collecting and computing the desired information, so it is successful in that regard and operation. However, some timestamps are exactly the same as ones adjacent to them, causing an infinite frequency. It is for this reason that in all subsequent tests time is modeled based on average frequency rather than the individual timestamps.

# Kinect vs. Vicon Accuracy Test Overview

The purpose of this test is to verify the accuracy of the Kinect, which it will then provide consistency in the overall system versus the Vicon system.

We will be using the Vicon motion capture system as our gold standard for comparing the accuracy of our computations for supporting data. We will collect all of the data from the Kinect after they have been recorded and the Vicon motion capture data and compute the same sets of supporting data and compare the measurements and calculate the root mean squared error and see if it meets the 15% RMS error requirement. This will be done by comparing the data with Matlab to easily visualize the information recorded and determine its relative accuracy.

#### Test Setup

First, we have to set up both the Vicon and the Kinect system. To set up the Vicon, a skeletal frame is created out of little reflective markers placed on the body. Then attached to the spine middle and spine base and the last 2 markers are placed on points of interest to create a model of the back (in our case, these points include the spine shoulder and right shoulder). Once the Vicon is properly calibrated, our system needs to be set up. The Kinect is placed in front of the subject far enough away in order to capture their whole body. Now, you bring up our C# program through Visual Studio and run it. The user hits start recording in order to collect data into a set of text files and stop record to close them and start performing our angle calculation.

With all of the data we require collected, we begin processing through a custom Matlab program. The matlab program parses all of the Vicon positional data, converts it to angle, angular velocity and then it is compared to the set of data collected from the Kinect. The RMS percent error is then calculated between the two and output by the program.

#### Data

## 

|  |  |
| --- | --- |
| **Trial** | **% RMS error** |
| 1 | 16.1693 |
| 2 | 15.5316 |
| 3 | 40.377 |
| 4 | 39.9142 |
| 5 | 15.5456 |
| 6 | 16.4486 |
| 7 | 21.2303 |
| 8 | 17.6615 |
| 9 | 38.3734 |
| 10 | 8.8399 |
| Avg. | 23.00914 |

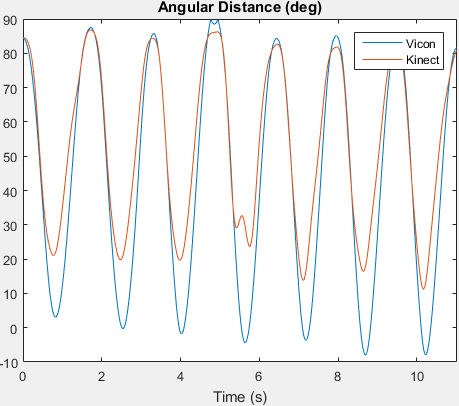


Fig. Kinect vs. Vicon Angular Distance

During this test, we conducted 10 trials, which we calculated the RMS percent error between the Kinect and the Vicon and took the average of the RMS. As you can observe in the image above, it shows the angular distance in degrees between the Kinect and the Vicon of one trial.

#### Test Conclusion

From the results of this test, we can conclude that the Kinect performs consistently on each trial. Although some of the trials have high RMS percent error, the reason behind that, it is because the Kinect does not perform very well while the subject is flexing in since the joints between the Spine Mid and the Spine Base are being obstructed by the head of the subject. Hence, increasing the percent error on lower angles, which it is expected by the Kinect since its major issue is the obstruction of the skeletal frame. As a result, we have decided as a team, to use the positive peaks of the Kinect to aid the IMU for the drift correction, instead of the average mean of the kinect.

# Kinect Consistency Test Overview

The purpose of this test is to verify the consistency of the Kinect, which it will provide in the overall system versus the Vicon system.

Again, we will be using the Vicon motion capture system as our gold standard for comparing the accuracy of our computations for supporting data. We will collect all of the data from the Kinect after they have been recorded and the Vicon motion capture data and compute the same sets of supporting data and compare the measurements and calculate the mean and see if it meets the 15 degrees offset requirement. This will be done by comparing the data with Matlab to easily visualize the information recorded and determine its relative accuracy.

#### Test Setup

Like previously tests, we have to set up both the Vicon and the Kinect system. To set up the Vicon, a skeletal frame is created out of little reflective markers placed on the body. Then attached to the spine middle and spine base and the last 2 markers are placed on points of interest to create a model of the back (in our case, these points include the spine shoulder and right shoulder). Once the Vicon is properly calibrated, our system needs to be set up. The Kinect is placed in front of the subject far enough away in order to capture their whole body. Now, you bring up our C# program through Visual Studio and run it. The user hits start recording in order to collect data into a set of text files and stop record to close them and start performing our angle calculation.

With all of the data we require collected, we begin processing through a custom Matlab program. The matlab program parses all of the Vicon positional data, converts it to angle, angular velocity and then it is compared to the set of data collected from the Kinect. The mean of each signal (KInect and Vicon) is then calculated between the two and output by the program.

#### Data

|  |  |  |  |
| --- | --- | --- | --- |
| **Trial** | **Kinect Mean** | **Vicon Mean** | **Mean Diff** |
| 1 | 53.7786 | 44.1744 | 9.6042 |
| 2 | 56.6584 | 46.7118 | 9.9466 |
| 3 | 52.5263 | 44.8297 | 7.6966 |
| 4 | 56.7593 | 44.7862 | 11.9731 |
| 5 | 58.6343 | 46.1638 | 12.4705 |
| 6 | 58.7251 | 47.2282 | 11.4969 |
| 7 | 61.0196 | 48.7837 | 12.2359 |
| 8 | 61.1412 | 49.8239 | 11.3173 |
| 9 | 59.6146 | 48.0592 | 11.5554 |
| 10 | 79.7464 | 86.8241 | 7.0777 |
| Avg. | 59.86038 | 50.7385 | 9.12188 |

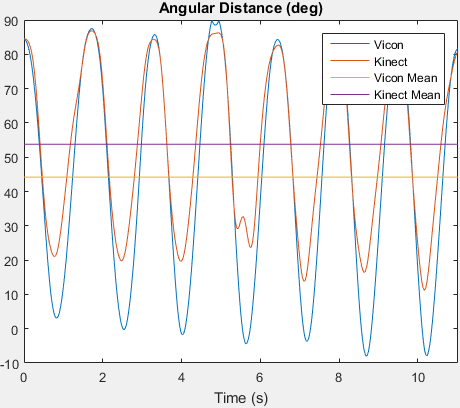


Fig. Kinect vs. Vicon Angular Distance with Mean

During this test, we conducted 10 trials, which we calculated the mean between the Kinect and the Vicon and took the average of the mean of all trials. As you can observe in the image above, it shows the angular distance in degrees between the Kinect and the Vicon of trial one.

#### Test Conclusion

From the results of this test, we can conclude that the Kinect performs consistently on each trial. Although some of the trials have high offset, all the trials stayed below our requirement that is 15 degrees offset. The reason behind that, it is because the Kinect does not perform very well while the subject is flexing in since the joints between the Spine Mid and the Spine Base are being obstructed by the head of the subject. Hence, increasing the average of the Kinect on lower angles, which it is expected by the Kinect since its major issue is the obstruction of the skeletal frame. Therefore, for our subsystem requirement, we are going to look only at the positive peaks, since the negative peaks cannot be reliable to provide consistency of the overall system compared to the IMU subsystem.

# Signal Correction and Control Subsystem

The signal correction subsystem is primarily made up of two code blocks made using the C# programming platform. The first block corrects drift that is inherent to the wearable inertial sensors, with a characterized drift correction algorithm which compares the linear best-fit line of the peaks of the IMUs to the same of the Kinect. This is done in a segmented fashion between each set of peaks.

The second code block is the signal synchronization block, which reads through the input signal data to determine any latency present between the signals, and correcting this latency. This is done using the peaks of the Kinect and IMU sensors as well, however only the maximum peak is selected and used to shift all necessary signals to the most delayed signal. Additionally, a minor coding block filters each input signal in order to eliminate as much noise as possible.

This subsystem is required for system functionality, as corrected signals must be provided to the data analysis subsystem to accurately determine the patient’s degree of lower back disorder. This subsystem serves as a bridge between the digital inputs of the entire sensor network, both Kinect and wearable sensors, and the rest of the system, feeding directly into the Database and Analysis subsystem. The input signals from the sensor network arrive to the computer containing this subsystem via USB 3.0 from the Kinect, and Bluetooth 4.0 BLE from the wearable sensors. Once correction has been accomplished, these signals are forwarded on to the data analysis unit for further processing.

Signal control is another subsystem contained in in the Control and Signal correction subsystem. It is responsible for system wide error handling and instruction. The design of this subsystem is still in progress, as full system integration will be accomplished in the near future. Once integration is underway, appropriate error handling for overall system can be integrated as well. Once an error is detected, this subsystem should be able to provide instruction for each individual subsystem on which processes to run and what to do with current data.

## Significant Changes in Direction

This subsystem, though exclusively software, has been affected by a procedural change in how the system collects data. As stated in the overall system changes, the calibration exercise has been eliminated in an effort to reduce stress on the patient under test, as well as streamline the system. This requires the signal synchronization unit to rely solely on the sinusoidal nature of the experiments for the peaks to be detected. Testing has shown this is possible, however proper filtering must be performed on the Kinect signal, as ripple at the peaks can cause significant error in the correction of latency.

Additionally, the calibration period for the drift correction has been removed as well. This was done primarily as a streamlining effort to push more workload into post processing. This also allows for variation in expected drift for any reason to be corrected for by the second motion sensing unit, the Kinect. This is performed by implementing an algorithm that defines the linear best fit lines between peaks of the IMU signal and the Kinect signal. The sensor drift is inherent in the IMUs, and therefore the long term trend cannot be trusted. However the long term trend of the Kinect is accurate, so these best fit lines are compared to the best fit lines between the Kinect peaks, and corrected to match them.

## Subsystem Specifications

Below is the subsystem’s specification matrix. These specifications are what was deemed appropriate in order to process as accurate data as possible.

Table 3.1  
Signal Correction Specification Compliance Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Nominal | Max |
| Signal synchronization sync error | N/A | 5 ms | 10 ms |
| Drift offset correction error | N/A | 0% | 5% |
| Posture error delivery time | > 80 ms | <100 ms | < 3 s |
| System error delivery time | N/A | <50 ms | < 100 ms |

The first compliance metric is the error between synchronized signals of the Kinect and IMUs. The IMUs capture at an average rate of 200 Hz, which corresponds to approximately once every 5 ms. The goal of the synchronization is to sync accurately within one of these captures, in order to provide the best data for use in drift correction. A buffer of one additional capture rate is the maximum error that is acceptable.

The new drift correction algorithm is being kept to the same metrics as the previous system, of being within 5% of the true values. However, as the updated algorithm is expected to provide a much more dynamic correction to the IMU drift, it is also expected for the error to be well within the 5% maximum.

The delivery times of errors were selected to provide the best possible reaction time by the patient under test. Excessive error delivery could result in frustration, as the patient would potentially need to repeat exercises due to the lack of prompt notification.

## Subsystem Status

The signal synchronization portion of the code is currently operational. Additional tuning is being experimented with on the filtering done before the signals are synchronized, as ripple in the Kinect signals continues to appear at some peak instances. Additionally, code was added to “skip” the initial positions of the patient by ignoring a certain timeframe from the beginning of the test trial. This is done to avoid the data when the patient is reacting to the prompt to start, as motion during this timeframe is irrelevant to peak correction. Should this prove to cause error in results, a contingency is to utilize the derived velocity of the signals in order to correct for synchronization. This however comes with it’s own set of problems, as the derivation of the Kinect signal significantly propagates the noise components, so additional filtering would need to be performed.

The drift correction subsystem has its logic completed in MATLAB code, but still needs to be ported into C# code. This is expected to be completed within the next week, for use in fully integrated system testing. Should this translation into C# prove to be too problematic, one contingency would be the readdition of the drift calibration, which involves placing the IMU sensor on a flat surface to characterize the velocity drift.

## Subsystem Technical Details

A flow chart of the signal synchronization block can be seen below in figure 5.4.1. It takes in data from the combined data bank of both IMU sensors and the Kinect. The values each are recording are read into this processing block, as well as the timestamps associated with each. These values are imported into the “list” data structures created in C# for the post processing portion of the system.

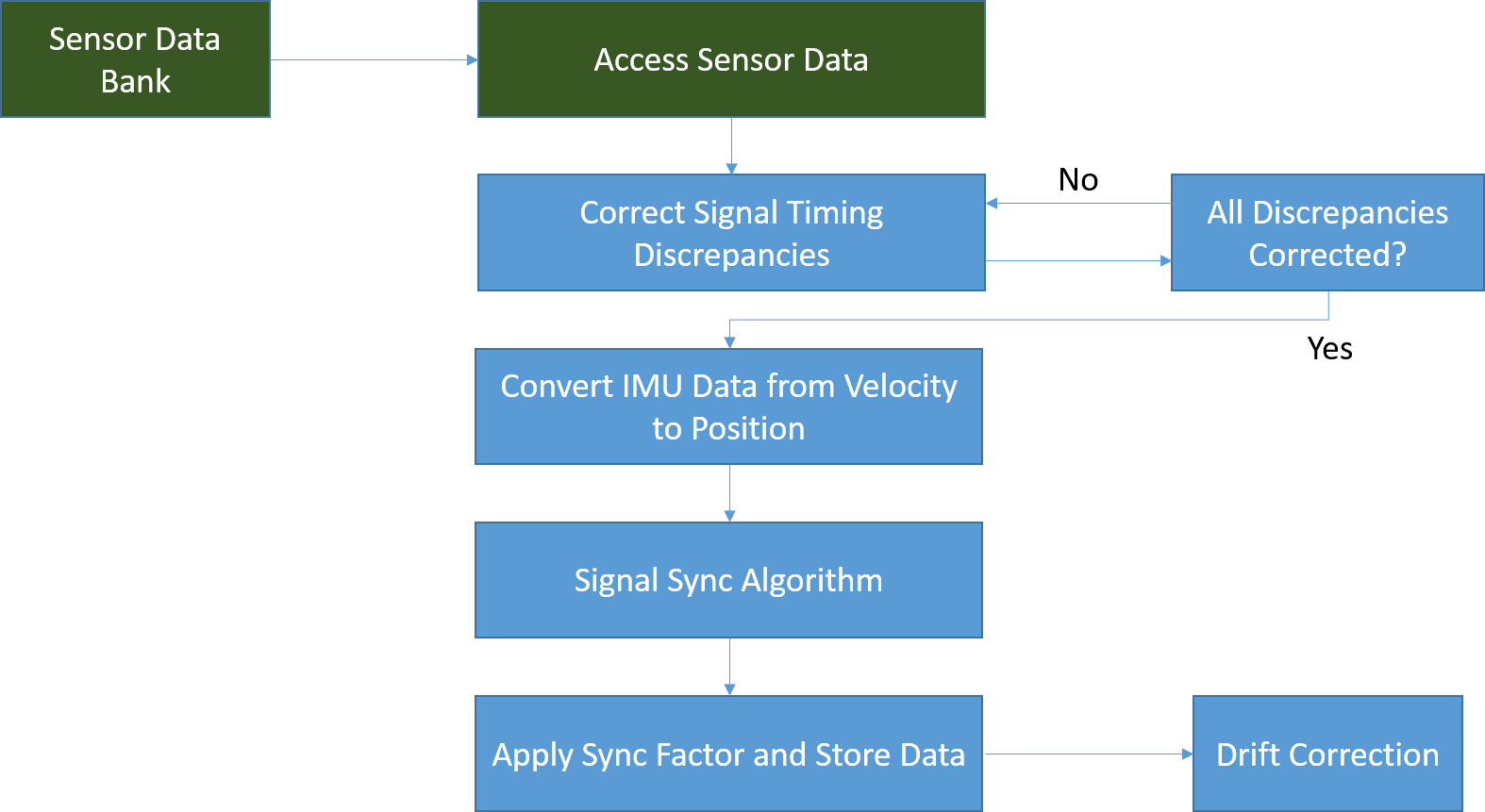


Figure 5.4.1: Signal Sync Flow Chart

Once data has been stored in these list, the mathematical functions simply manipulate data within these list by extracting each element in a “for” loop. The magnitude of each extracted element from the data point list is compared to a storage variable that initially contains a zero value. If the extracted value is strictly greater than the value contained in the storage variable, that element is now considered the new max of the list and is stored in the variable. It is important to note that the element value must be strictly greater than the existing storage value, as this eliminates multiple maximums overwriting each other. It was determined before the code was written that the first maximum value would be taken for each signal input, regardless of multiple maximums further along the signal. Each time a maximum value overwrites an existing maximum, the value’s index is stored in a variable as well. Once the entire signal has been read and the loop exits, the saved index variable for a maximum is used to extract the appropriate timestamp from the timestamp list. The “for” loop can be seen below in Code 3.1.

Code 3.1

Maximum comparison loop

for (int i2iter = 0; i2iter < Input2.Count; i2iter++)

{

I2cont = Input2[i2iter];

if (Math.Abs(I2cont) > I2max)

{

I2max = I2cont;

I2Eletracker = I2Ele;

}

I2Ele = I2Ele + 1;

}

Once the comparison has been completed and the maximums have been found, they are simply compared to one another to detect the latency between the two. This latency value is then added to the respective timestamps of the least delayed signals in order to bring them in line with the most delayed one.

The drift correction block of this subsystem is still being coded in C#, so no code is available beyond simple import data functions. However, below in figure 5.4.2 a flow chart of general processing for the correction can be seen.

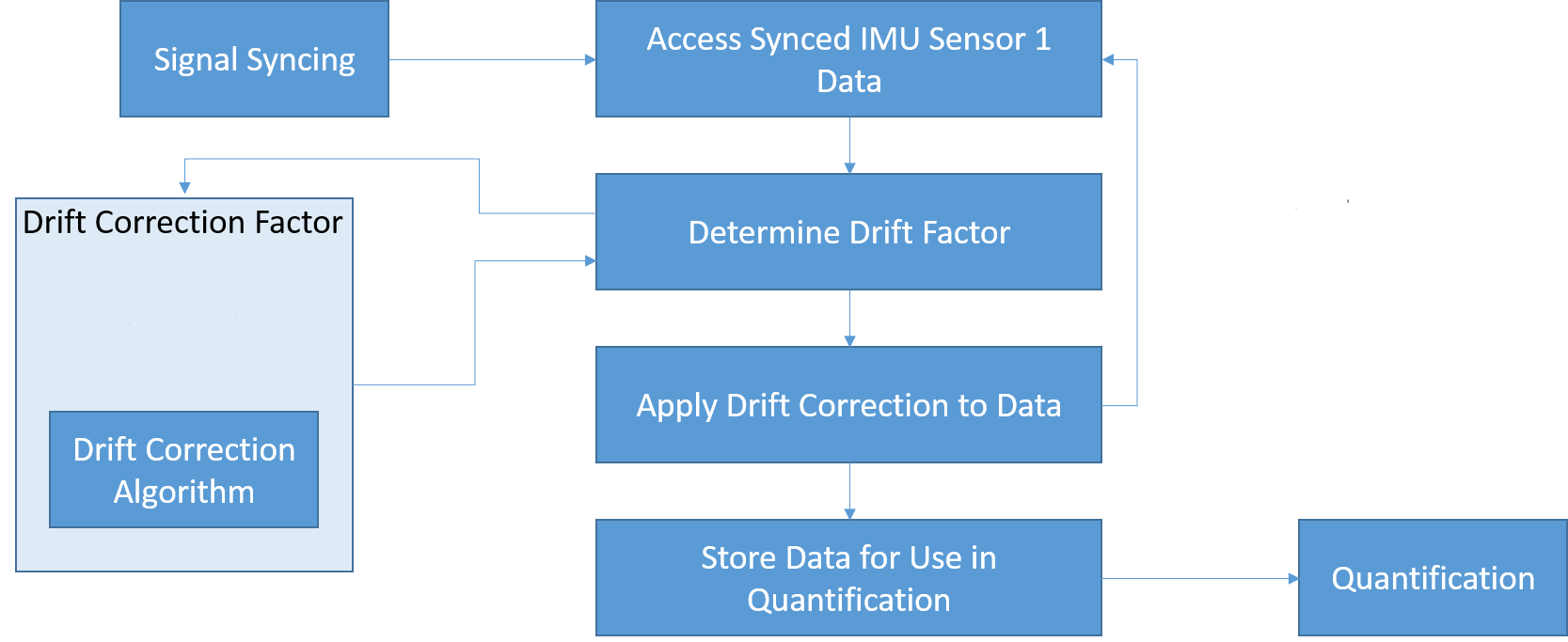


Figure 5.4.2: Drift Correction Flow Chart

A code snippet can be seen below showing the MATLAB representation of calculating a best fit line.

## Subsystem Testing

The testing done on this subsystem involves passing data collected through the code blocks and observing the outputs graphically in MATLAB to ensure the peaks are aligned properly. This is done for both code blocks, as the peaks should be aligned in time as well as in magnitude (y-axis) due to the signal synchronization code and the drift correction code.

### Signal Synchronization Test

This test determines the validity of the signal synchronization block to align the peaks properly. It is important to note that the peaks are the focus of this test, as the rest of the data is subject to problems such as obstruction inherent to the Kinect, as well as potential scaling issues on the IMUs. The signal synchronization feeds directly to the drift correction portion, which requires the peaks to be aligned to properly function.

#### Test Setup

The recorded sinusoidal motions are passed through the code block and the outputs are saved via a text file. This text file is then imported into MATLAB to observe the plotted waveforms and analyze the values of peaks to determine if proper synchronization was performed.

#### Data

Below are two plots illustrating the translation of one signal to the other, showing that the synchronization block is working as expected. Additional tweaking to filtering before this process will result in smoother Kinect curves, and therefore improve accuracy.

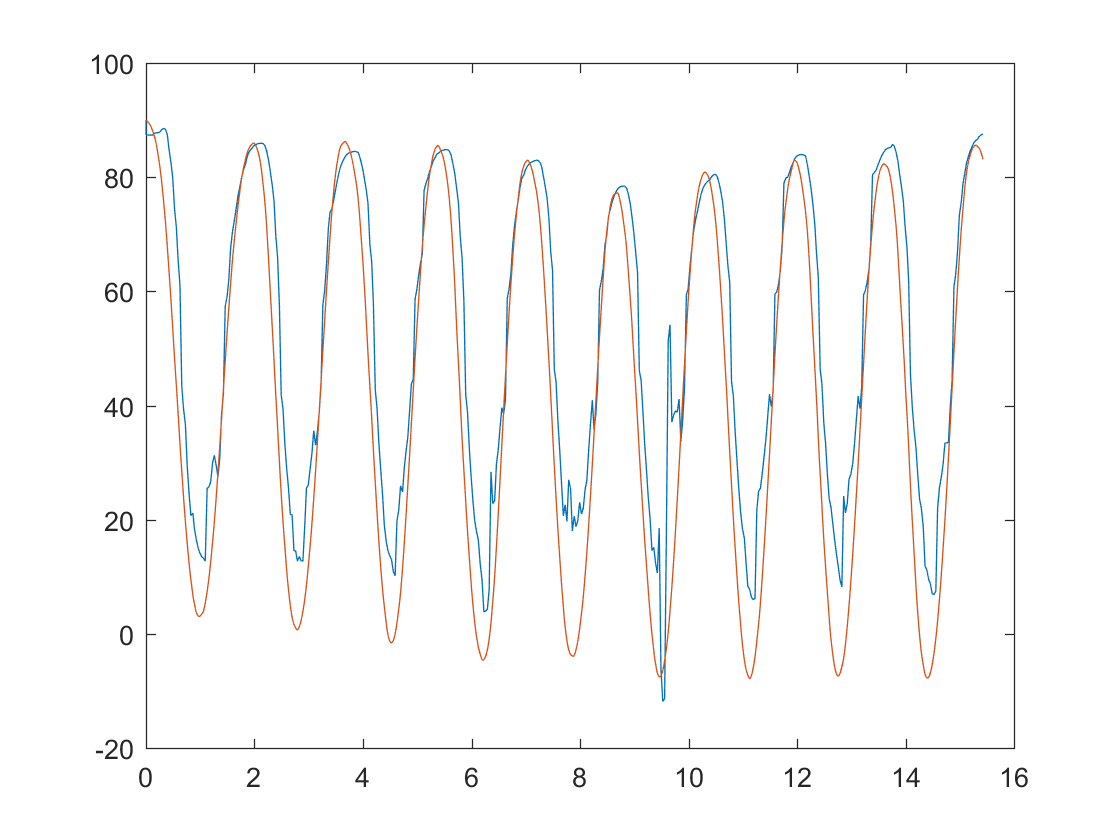


Figure 5.5.1.1: Uncorrected signals

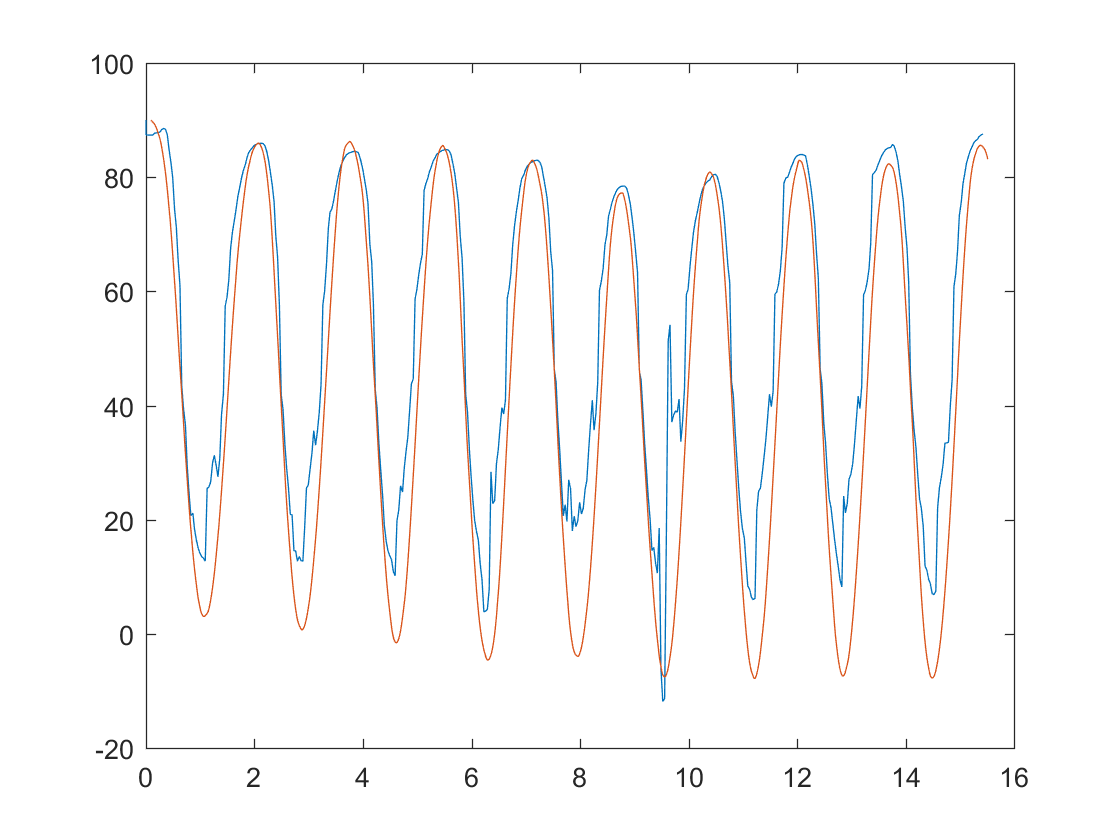


Figure 5.5.1.2: Corrected Signals

#### Test Conclusion

It can be inferred from the data that the subsystem is operational, as it is accurately shifting signals in the time axis back and forth to correctly line up peak values. From this data, the drift correction code should be able to correctly adjust the IMU signals for drift, providing accurate data for analysis in the Database subsystem.

### Drift Correction Test

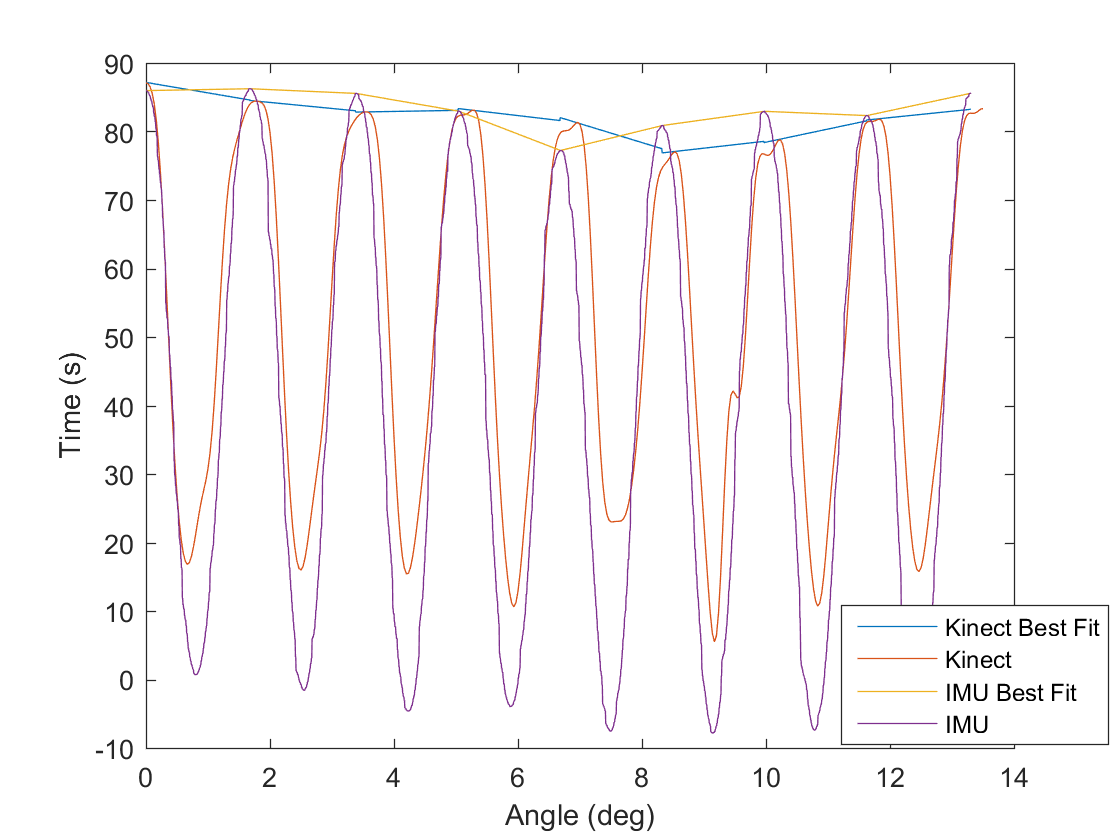
The current drift correction test is implemented in MATLAB code only. The algorithm used now is relatively new to the project, as the design change away from a calibration exercise was recently implemented.

#### Test Setup

The test performed was simply taking recorded data and using MATLAB functions to determine best fit lines between adjacent peaks in a signal. The recorded data was taken using the integrated system test currently in place for this project. A patient in a wet suit has the motion sensors and Vicon markers placed on their body and then begin the exercise regime dictated in the detailed test plan. The Kinect is also used to observe their movements. All the data collected from these systems is loaded into MATLAB for the analysis.

#### Data

* + - 1. Below is a data plot showing the Kinect and IMU data that has been captured. Drift can clearly be seen in the IMU data. The individual best fit lines are drawn showing how each one varies from the other. The data will be manipulated to follow the best fit lines of the Kinect, in order to correct for the long term drift exhibited by the IMUs

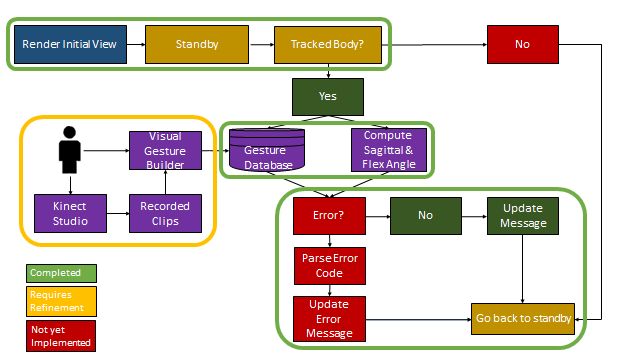


#### Test Conclusion

Once this algorithm is implemented into C#, this subsystem will be operating at a high level of accuracy. It will also be extremely flexible to variations in drift from the IMU sensors. However, it is also evident that this part of the subsystem will be difficult to code, as some of the native functions in MATLAB may need to be coded as their own functions in C#, depending on what the MATH libraries have available.

# Feedback Display Subsystem

The main purpose is to provide a display feedback to the patient, it will display a clear view of an error signal whenever the patient is not doing the experiment in the correct manner. In addition, it alerts the patient on how to fix their posture. First, it renders the initial view, which then it will wait till it tracks a body. If it tracks the body, it will then compute the sagittal and flex angle and it will initialize the gesture detector, which runs out of a gesture database created in Visual gesture builder. It will then check if it detects any error, which then it will parse it and update the error message.



*Figure 3.1: Feedback Display Flowchart*

## Significant Changes in Direction

Overall, the main purpose of this subsystem has remained unchanged after its definition in the proposal. However, we have decided to change the 15 degree sagittal angle requirement for the patient, as it would be very difficult for the patient, as well as the Kinect to stay within the 15 degrees of sagittal angle while performing the trials. Instead, we are just going to look at the 0 degree and 30 or above for the sagittal angle with the patient is doing the test, since this adds a lot of flexibility, and also meets the specification of our research.

## Subsystem Specifications

Table 3.1  
Feedback Display specification Compliance Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Nominal | Max |
| Capture rate of Kinect (in fps) | 28 | 30 | 32 |
| Angle offset compared with Vicon (in degrees) | 3 | 5 | 15 |

As mentioned in the overview, this subsystem is tasked to provide posture control for the patient and aid them during their trials. To test the overall performance of this subsystem, the flex and the sagittal angles of the Kinect are compared against our golden standard of testing, which is our Vicon subsystem. By using the incredibly accurate vicon, we can check the accuracy of the data we are collecting and computing in order to ensure that the quantification metrics are as accurate as possible with the model of Lower Back Disorder we have created. In addition, we will ask the patient that performs the trials for feedback on how to improve the GUI for better understanding without compromising the focus of the patient but instead, help them.

## Subsystem Status

The Feedback Display subsystem is functionally complete. It tracks the body of the patient and calculates the sagittal and the flex angle to help the patient during test. It also provides a gesture tracker, which is based on Visual Gesture Builder. This was built into a database, which includes different positional gestures to indicate the patient on how do the test.

Although, it is functionally complete, it requires minor improvements since we would need to keep expanding the database with different environments, and body sizes to allow the Display Feedback to have flexibility while helping the patient. It also shows the patient whenever they are doing something wrong for them to correct it without major distraction.

## Subsystem Technical Details

As for the Feedback Display, we have to use several classes to be able to implement the gestures while helping the patient. In order to do that, we created a class called GestureDetector for which it gathers all the gestures from the gesture database and compares it with the patient. As shown in the next sample code:

Code 3.4 Initialization of GestureDetector.

public GestureDetector(KinectSensor kinectSensor, StandingResultView standingResultView, ArmsCrossedResultView armsCrossedResultView)

{

…

using (VisualGestureBuilderDatabase database = new VisualGestureBuilderDatabase(this.gestureDatabase))

{

this.vgbFrameSource.AddGestures(database.AvailableGestures);

}

}

As for the individual gestures, we had to create a separate class for each gesture as they contain a set of data structures to show the patient the level of confidence, as well as, an image to easily represent the gesture that the patient has to be on. For example in the next sample code:

Code 3.5 Standing Gesture.

public StandingResultView(int bodyIndex, bool isTracked, bool detected, float confidence)

{

this.BodyIndex = bodyIndex;

this.IsTracked = isTracked;

this.Detected = detected;

this.Confidence = confidence;

this.ImageSource = this.notTrackedImage;

}

In addition, to be able to track only one body during test, an algorithm was implemented in the main window, which can be seen in the next sample code:

Code 3.6 Get body index.

//MainWindow.xaml.cs

private int GetActiveBodyIndex()

{

...

for (int i = 0; i < maxBodies; ++i)

{

{

...

activeBodyIndex = i;

break;

}

}

return activeBodyIndex;

}

Finally, a quick check of parameters were set to notify the patient if they are out the boundaries that needs to be during test. However, those are not set in stone, as they do not have a specific reason to be that value. Although, it will change in the upcoming months with the integration of the other sub-systems.

## Subsystem Testing

# Feedback Display Visibility Test Overview

Multiple subjective visibility tests will be performed by members of the team and our mentors including Dr. Jafari and his team to ensure that the GUI for the feedback display is clear from 1 - 2 feet away.

#### Test Setup

We will use a standard monitor which is at least approximately 42 inches and with at least a refresh rate of 30 Hz and then bring up our C# program and run it to show the feedback display and then have multiple people on our team and Dr. Jafari’s to verify that it is visible within 1-2 ft away.

#### Data

There is currently no data that we plan to have to collect. However, we do plan on getting feedback from Dr. Jafari and his team in order to ensure that it meets their standards. This feedback will be later provided and the update Feedback Display.

#### Test Conclusion

Since the monitor described in the test will have a refresh rate of at least 30 Hz, we will have verified at least one specification. However, since the visibility is largely subjective, there is no way to ensure that is going to be accurate for all people. For most regular users though, we plan to use this test to ensure that is up to our specification standards.

# Subsystem Database And Analysis

Using the angular velocity and positional data provided by the IMUs and the kinect and our previous subsystem’s correction, the database and analysis’s main purpose is to use this data to accurately and precisely quantify the severity of Lower Back Disorder within the patient. Having received angular velocity which was converted into angular distance and corrected, this subsystem is required to compute an additional set of angular data which includes angular acceleration and angular jerk. Once these are computed, this set of data is analyzed according to a set of patient tests and a model of Lower Back Disorder described in The Quantification of Low Back Disorder using Motion Measures by Marras et. al. [1] to create a quantification of Low Back Disorder severity on a scale from 1 to 10 with 1 being the least severe and 10 the most. Lastly, this information is then desired for future analysis of the Lower Back Disorder model so it is stored within a patient database after computation and then this information is sent to our Data Display Interface (DDI) for reviewing the results.

## Significant Changes in Direction

Overall, the main purpose of this subsystem has remained unchanged after its definition in the proposal. However, we have decided to change the direction in which the overall quantification is calculated by utilizing multiple trials throughout a patient test in order to average out and potentially throw away any erroneous data collected throughout a single trial.

## Subsystem Specifications

Table 7.1  
Database and Analysis specification Compliance Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Nominal | Max |
| Angle, angular velocity, angular acceleration, and angular jerk error with respect to Vicon | N/A | <5 % | 25% |
| Severity of LBD quantification precision | 80% | 90% | 100% |
| Delay time for quantification | N/A | < 1min | 5 min |

As mentioned in the overview, this subsystem is tasked with computing another set of angular data which are angular acceleration, and angular jerk and to then use the corrected angular distance and angular velocity data with these to quantify the severity of Lower Back Disorder within the patient. To test the overall performance of this subsystem, these calculations are compared against our golden standard of testing which is our Vicon subsystem. By using the incredibly accurate vicon, we can check the accuracy of the data we are collecting and computing in order to ensure that the quantification metrics are as accurate as possible with the model of Lower Back Disorder we have created.

Other metrics include the precision of this quantification which should remain as constant as possible for the same patient in order to ensure that the same patient is not provided wildly different results and potentially give incorrect severity quantifications. Lastly, we wish to measure how long this overall process takes in order to ensure that patient testing can be performed with minimal time overhead for this subsystem.

## Subsystem Status

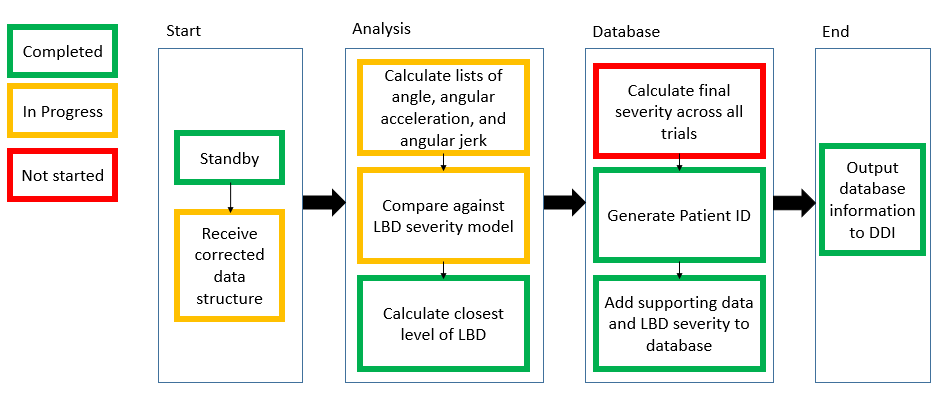


Fig 7.1 Database and Analysis Flowchart

Thus far, the Database and Analysis subsystem’s base functionality has been completed. Referring to the flow chart, everything which is outlined in green has been completed which includes the static standby state, the actual quantification of LBD, its database components, and the output to the DDI. However, what is yellow is currently being refined. As such, their quality according to their specifications still needs to be tested and ensure that they are met. The relative accuracy of the angular data calculations still needs to be verified. However, the algorithms have been written in C# and are currently output into text files for testing. The LBD model is in a similar status where it's been defined and written in C#, but its relative accuracy and thresholds for the checks defined by Dr. Marras still need to be worked on to ensure that they provide a precise and accurate measure of the severity of Lower Back Disorder. Lastly, without full integration with the correction units for our data, my subsystem is still working on receiving input from the correction unit which is a data structure called IMU Data. The data structure itself has been defined and its method for calculating angle from angular velocity has been implemented as well. The correction code just needs to be integrated for it to be completed. The only part of my subsystem which has yet to be implemented is the average severity calculation across trials since it was a relatively new change that we plan on assessing and implementing in the coming weeks.

## Subsystem Technical Details

Overall, this entire subsystem is implemented with code from C#. In terms of progress, most of the changes have been implemented with the function for implementing cumulative trapezoidal integration, data collection and storage of the IMU data for processing, and fixes to the step derivative function to define averaged steps for calculating the slope between each and every data point (a.k.a frame) in order to collect an accurate measure of the angular acceleration and angular jerk. However, I will also delve into the overall details for parts which have already been implemented such as the code for quantifying LBD according to the model previously mentioned and the database.

In order to implement the quantification of LBD, I first required a data structures to contain all of the sensor data being collected by the Kinect and wearable sensors. For the IMU’s, I needed to implement the ability to collect the sensor’s respective data. Both of the IMU’s have a bluetooth connection which are defined by the user to specify their COM ports, which are then used to create a connection and thread to decode the data being received by the IMUs, read them off a data queue, and then convert the bit stream from the IMUs into integers, create a timestamp, and then store them into what I called the IMU data structure. Below is an example code showing the generation of the time stamps inside the Processing1 function which is the function used by the first sensor’s respective thread.

Code 7.1 Database and Analysis code from MainWindow.cs

if (kinect\_start != 0)

{

//Create timestamp

DateTime datenow = DateTime.Now;

int hour = datenow.Hour;

int minute = datenow.Minute;

int second = datenow.Second;

int millisecond = datenow.Millisecond;

int timestampS = hour \* 3600 \* 1000 + minute \* 60 \* 1000 + second \* 1000 + millisecond;

// Use bit converter to convert the read bits into integers

gyroXMid.Add(BitConverter.ToInt16(convert, 16));

gyroYMid.Add((float)(BitConverter.ToInt16(convert, 14)/32.75));

gyroZMid.Add(BitConverter.ToInt16(convert, 12));

timeStampsMid.Add(timestampS);

}

The time stamp and gyro lists which are shown above are then passed into the IMU data structure after collection has been completed. Using these, the IMU data structure has an implemented function which performs cumulative trapezoidal integration over the gyroscope Y mid data and the collected time stamps in order to find the angular distance of the specific sensor. Shown below is the code for cumulative trapezoidal integration.

Code 7.2 Database and Analysis code from IMUData.cs

float angle = 0;

for(int i = 0; i < transposedTSMid.Count - 1; i++)

{

// Trapezoidal rule

// angle = (b - a)\*(f(b)+f(a)/2)

// Multiply by correction factor

float timeDiff = transposedTSMid[i + 1] - transposedTSMid[i];

float correctedDiff = (gyroYMid[i+1] + gyroYMid[i])/(2);

angle = angle + (timeDiff\*correctedDiff);

anglesMid.Add(angle + 90);

}

Referencing the code, the IMU data structure utilizes the time stamps, which are then converted from milliseconds to seconds and transposed to start at a time relative to 0 starting when data is collected. Using this, it can then simply perform cumulative trapezoidal integration through the formula:

CodeCogsEqn (1).gif

Fig 7.2 Cumulative Trapezoidal Integration Formula

Where n refers to the data point of the time stamp list, and f(t) corresponds to the data point corresponding to that timestamp and k is the total amount of data points collected by the gyroscope.

With the IMU data, time stamps, and angular distance implemented, I can delve into the integrating of this information into my Data Analysis. To implement the subsystem, I created a data structure specific to this entire subsystem called DataAnalysis. This structure is responsible for collecting specific patient data such as age and gender, and contain the calculated supporting data of angular acceleration, and angular jerk. In addition, it needed to define methods for calculating this supporting data and then analyzing this data to be used by its main Quantify LBD algorithm. To do so, I collected the supporting data as a set of lists corresponding to the tests they were collected under and defined the followed method.

Code 7.3 Step Derivative Function from DataAnalysis.cs

private List<float> CalcStepDerivative(List<float> floatList, float step)

{

List<float> derivative = new List<float>();

float derivValue = 0;

for(int i = 0; i < floatList.Count - 1; i ++)

{

if (floatList[i + 1] == floatList[i])

{

derivative.Add(0);

}

else

{

derivValue = ((floatList[i + 1] - floatList[i]) / step);

derivative.Add(derivValue);

}

}

return derivative;

}

Referring to the code, this function takes in the list of data that we wish to derive across and a step of which to use as a difference of time in order to essentially implement a simple slope between two points across the entire list going from i to i+1 until the end of the list (except -1 to ensure that we do not exceed bounds). The time step referred to are calculated using a slightly different method than previously for the angular distance due to a separate issue with the timestamps duplicating and causing the occasional step of 0. To rectify this issue, the timestamp differences are simply averaged out by taking the overall time lapse and then dividing them by the total number of frames collected in order to get an effective timestamp interval. Once this is done, the code above can simply be represented as the function shown below:

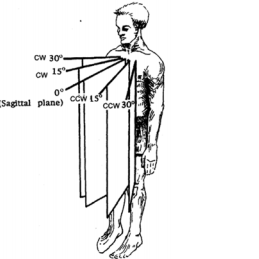
|  |  |
| --- | --- |
| DerivativeList*[i]=(List[i+1] -List[i])/step* | (7.3) |

This is essentially a simple derivation across the entire list and will be utilized later by the main quantification algorithm to essentially compute the following relations between angular velocity, angular acceleration, and angular jerk starting from angular velocity.

Report DA Formulas Pic.PNG

*Figure 7.4: Derivation relations between angle, angular velocity, angular acceleration, and angular jerk*

One of the main tests for determining whether the patient has LBD is to have the patient rotate across the sagittal plane and perform trunk flexions at specific points. Shown below is a figure referring to the sagittal plane rotation that I am referring to.



*Figure 7.5: Sagittal Plane Rotations*

At each of these points, the patient will flex and then extend the lower back and during these, I collect angle from the Kinect and IMU with respect to the sagittal plane and to the flex angle and the angular velocity of the trunk using the motion sensors. Currently, the motion sensors are only partially integrated in order to perform calculations of only the flex angle while the Kinect utilizes and computes the sagittal plane angle. The most important point of data within each of these supporting data lists computed by the step derivative are the peaks. This allows us to check whether the patient is having any issues while performing the test since they will be told to flex and rotate to certain degrees as fast as they comfortably can. The peak algorithm itself is a simple linear search through the list making a comparison in order to find the max. You may refer to DataAnalysis.cs to see the algorithm itself. Below is the quantification of the peaks algorithm.

Code 7.4 Quantification of Supporting Data Peaks Function from DataAnalysis.cs

private double QuantifyPeak(float max, float peakMeasurement, double peakFactor)

{

double quantifiedPeak = 0;

if(max < peakMeasurement)

{

return 0;

}

quantifiedPeak = (peakMeasurement / max);

quantifiedPeak = (quantifiedPeak \* peakFactor);

quantifiedPeak = peakFactor - quantifiedPeak;

return quantifiedPeak;

}

The max is a float which denotes a limiter on the peak which will be defined based on my model of LBD which denotes whether the patient was able to reach a certain maximum threshold to ensure they do not have Lower Back Disorder. Otherwise, we simply weigh down the peak measurement to find a number from 0 to 1 in which they fall on a scale and then multiply it by a peak factor that is passed in.

After defining these key structures and their required functions, I started defining the main quantification algorithm which has a snippet shown below.

Code 7.5 Quantification of LBD from DataAnalysis.cs

angularSPVel = CalcStepDerivative(kinectSPAngleAt0, step);

angularFlexVel = CalcStepDerivative(kinectFlexAngleAt0, step);

angularSPAccel = CalcStepDerivative(angularSPVel, step);

angularFlexAccel = CalcStepDerivative(angularFlexVel, step);

angularSPJerk = CalcStepDerivative(angularSPAccel, step);

angularFlexJerk = CalcStepDerivative(angularFlexAccel, step);

peakSPAngle = FindMax(kinectSPAngleAt0);

peakFlexAngle = FindMax(kinectFlexAngleAt0);

peakSPAngVelocityAt0 = FindMax(angularSPVel);

peakSPAngAccelerationAt0 = FindMax(angularSPAccel);

peakSPAngJerkAt0 = FindMax(angularSPJerk);

peakFlexAngVelocityAt0 = FindMax(angularSPVel);

peakFlexAngAccelerationAt0 = FindMax(angularSPAccel);

peakFlexAngJerkAt0 = FindMax(angularSPJerk);

minFlexAngle = FindMin(kinectFlexAngleAt0);

maxSPCWAngle = peakSPAngle;

maxSPCCWAngle = FindMin(kinectSPAngleAt0);

//Normalize peaks to a corresponding rating factor

rating += QuantifyPeak(maxVel, peakSPAngVelocityAt0, peakAngVelFactor);

rating += QuantifyPeak(maxAcc, peakSPAngAccelerationAt0, peakAngAccFactor);

rating += QuantifyPeak(maxJerk, peakSPAngJerkAt0, peakAngJerkFactor);

//Calculate twisting ROM

twistingROM = maxSPCWAngle - maxSPCCWAngle;

if (peakSPAngle < 30)

{

rating += spROMFactor\*.5;

spROM30 = false;

if (peakSPAngle < 15)

{

rating += spROMFactor \* .5;

spROM15 = false;

}

}

if(minFlexAngle > flexAngleROM)

{

rating += fpROMFactor;

fpROM = false;

}

//Still needs definition

if(asymComplete == false)

{

rating += asymCompleteFactor;

}

//If unable to twist more than 90 degrees total

if(twistingROM < 45)

{

rating += twistingROMFactor;

}

//Factor in age

ageFactor = 1 / ((float)age);

rating += (ageFactor \* 10);

//Normalize accoridng to gender factor

if (gender == true)

{

rating = rating \* maleGenderFactor;

}

else

{

rating = rating \* femaleGenderFactor;

}

severityLBD = rating;

return rating;

}

As you can see, I initially utilize the algorithms I defined earlier to find the supporting data lists. I proceeded to use these lists to find the peak within each and then quantified the respective severity based off the peaks. The following checks are simple range of motion checks utilizing these peaks in order to ensure the patient has reached certain benchmarks during testing such were they able to rotate across the sagittal plane at least 30 degrees, or were they able to flex a total 55 degrees. Lastly, the rating is then normalized towards age and gender according to specific factors defined for each.

As mentioned previously, the factors are currently in the process of being defined since they are somewhat arbitrary as of right now. As a result, I integrated a new form into our MainWindow.xaml.cs file which acts as a preliminary display for testing and debugging purposes. In this form, I defined the ability to calibrate the Kinect, define sensor ports, start collecting data, and stop collecting. Most importantly, I defined a set of text boxes to allow for user input for defining the quantification factors to allow for testing and modifying the quantification algorithm.

The final requirement for the subsystem was to implement the database. To do so, I used visual studio to create an SQL database called PatientData and defined its set of columns and ported it to locally connect to my computer. The set of columns will need updating and it will eventually need to be made external for access and use by the Spine Research Institute. However, the basis code for connecting to the database, inserting SQL commands to insert data into the database, and closing the connection is shown below.

Code 7.6 Database code from MainWindow.cs

SqlConnection connection = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\Alex\Documents\ECEN403Github\AlexDubois\BodyBasics-WPF-Database\PatientData.mdf;Integrated Security=True");

try

{

connection.Open();

SqlCommand cmdRead = new SqlCommand();

SqlCommand cmd = new SqlCommand();

SqlDataReader dr;

cmd.CommandText = "SELECT TOP 1 PatientNum FROM Patients ORDER BY PatientNum DESC";

cmd.CommandType = CommandType.Text;

cmd.Connection = connection;

dr = cmd.ExecuteReader();

// Read first entry corresponding to patient number

int patientIDNum = 0;

if (dr.Read())

{

patientIDNum = Convert.ToInt32(dr["PatientNum"]);

// Increment to get next patient ID

patientIDNum++;

}

connection.Close();

connection.Open();

cmd.Connection = connection;

//@TODO: Fix potential SQL Code Injections

if (dataAnalysis.gender == true)

{

cmd.CommandText = "insert into Patients (PatientNum,Age,Gender,PeakSPVelocityAt0,PeakSPAccelerationAt0,PeakSPJerkAt0,FPROM,SPROM15,SPROM30,AsymComplete,TwistingROM) values ('" + patientIDNum.ToString() + "','" + ageBox.Text + "', '" + "1" + "','" + dataAnalysis.peakSPAngVelocityAt0.ToString() + "', '" + dataAnalysis.peakSPAngAccelerationAt0.ToString() + "', '" + dataAnalysis.peakSPAngJerkAt0.ToString() + "', '" + dataAnalysis.fpROM.ToString() + "', '" + dataAnalysis.spROM15.ToString() + "', '" + dataAnalysis.spROM30.ToString() + "', '" + dataAnalysis.asymComplete.ToString() + "', '" + dataAnalysis.twistingROM.ToString() + "')";

}

else

{

cmd.CommandText = "insert into Patients (PatientNum,Age,Gender,PeakSPVelocityAt0,PeakSPAccelerationAt0,PeakSPJerkAt0,FPROM,SPROM15,SPROM30,AsymComplete,TwistingROM) values ('" + patientIDNum.ToString() + "','" + ageBox.Text + "', '" + "0" + "','" + dataAnalysis.peakSPAngVelocityAt0.ToString() + "', '" + dataAnalysis.peakSPAngAccelerationAt0.ToString() + "', '" + dataAnalysis.peakSPAngJerkAt0.ToString() + "', '" + dataAnalysis.fpROM.ToString() + "', '" + dataAnalysis.spROM15.ToString() + "', '" + dataAnalysis.spROM30.ToString() + "', '" + dataAnalysis.asymComplete.ToString() + "', '" + dataAnalysis.twistingROM.ToString() + "')";

}

cmd.ExecuteNonQuery();

connection.Close();

}

catch (Exception ex)

{

MessageBox.Show("Can not open connection ! ");

}

Initially, I defined a connection string which is utilized to create a connection to the database. I proceeded to use function calls from an SQL library available in C# to create and execute SQL commands. In order to generate the primary key required for creating each individual unique entry, I denoted this as a Patient ID and use the SQLdatareader to sort the database in descending order and read the top entry and add 1 to the integer value to create a new patient ID value which is guaranteed to be unique. Lastly, this also allowed me to programmatically define a string of an SQL insert command to create a new row within the database for new patient entries. It is written by denoting all of its fields and then defining the values as strings to be inserted and then executing the command. Lastly, a close connection is called. If any of the steps fail in between this, then an exception is caught and we display a message saying that there was an error while attempting to connect to the database.

## Subsystem Testing

### Accuracy and Timing Test Overview

The main test which will be performed on this system will be done in coordination with the signal correction unit and Vicon. The signal correction unit will retrieve angular velocity and the Vicon will retrieve position in the x, y, and z plane. These will then be used to calculate angle using a trapezoidal integration algorithm, and angular acceleration and angular jerk using a step derivative function. These will be compared to the same functions calculated in Matlab with recreated angular data from the Vicon. The RMS percent error will then be calculated to determine its overall relative accuracy and timing will be captured through timestamps to verify how long the total quantification took.

#### Test Setup

The first and foremost step is to set up both the Vicon and our data collection system. To set up the Vicon, a skeletal frame is created out of little reflective markers placed on the body. These are then placed on our two sensors and the sensors are then attached to the spine middle and spine base and the last 2 markers are placed on points of interest to create a model of the back (in our case, these points include the spine shoulder and right shoulder). Once the Vicon is properly calibrated, our system needs to be set up. The Kinect is placed in front of the subject far enough away in order to capture their whole body. Now, we bring up our C# program through Visual Studio and run it. The user hits start recording in order to collect data into a set of text files and stop record to close them and start performing our angle, angular acceleration, and angular jerk calculations.

Once stop recording is hit, we then take a timestamp immediately and another timestamp after the quantification algorithm ends to time how long it takes to calculate all of our data. With all of the data we require collected, we begin processing through a custom Matlab program. The Matlab program parses all of the Vicon positional data, converts it to angle, angular velocity, angular acceleration, and angular jerk and then it is compared to the same set of data collected by our system. The RMS percent error is then calculated between the two and output by the Matlab program.

#### Data

Currently, our subsystems are not fully integrated in order to allow for the correction unit to supply synchronized and corrected data. However, much of this code is currently implemented in Matlab and simply requires porting into C#. As a result, we can provide the root mean square error of the angle, angular velocity, and angular acceleration with respect to the Vicon of the synchronized IMU’s data as processed through Matlab. The step derivative functions which were used to calculate the angular velocity, angular acceleration, and angular jerk will be recreated in C# and will ultimately be used to compare against the Vicon.

Shown below are our plots of data. The first shows the side by side comparison of the angular data collected by the Vicon, IMU’s, and the Kinect. The next three plots shows the RMSE of angular velocity, angular acceleration, and then angular jerk vs. Frames (data point number collected).

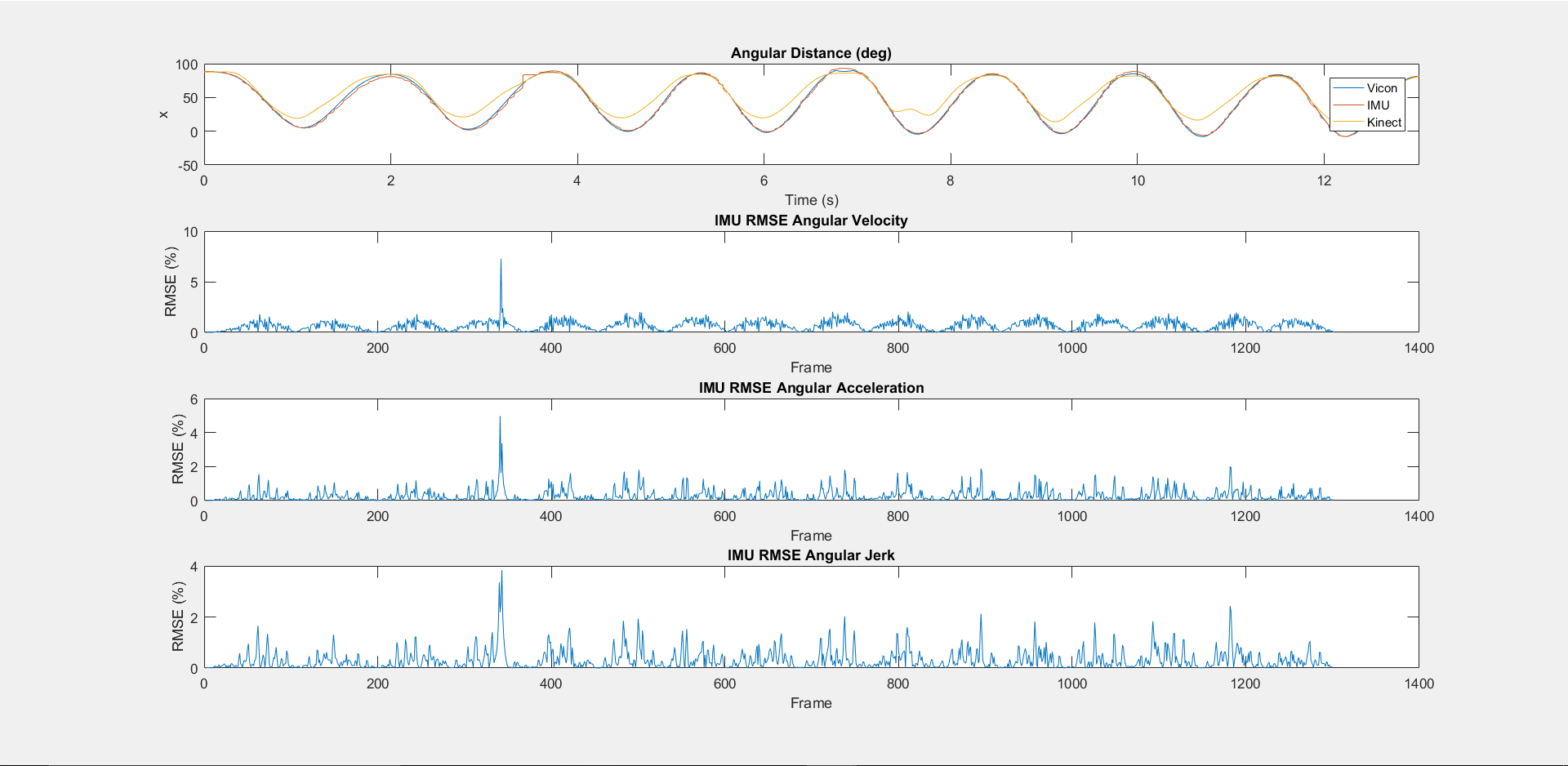


Fig 7.6 RMSE of Angular Data With Respect to Vicon

In order to get a better look at the overall accuracy of the system, I calculated the total mean RMSE of each of these which came out to be:

Angular Velocity of IMU vs. Vicon Angular Velocity = .8227 deg/s

Angular Acceleration of IMU vs. Vicon Angular Acceleration = .4661 deg/s^2

Angular Jerk of IMU vs. Vicon Angular Acceleration = .4661 deg/s^3

However, as you can see in the plot, there is data that still needs to be assessed such as the random spike seen at around frame 350. The RMSE of our actual fully integrated system is expected to be at least a bit higher than how it is in the data and figure shown above.shown above due to the usage of the same Matlab algorithms for both the Vicon and the IMU’s in the provided data. However, we plan to recreate the same plots and mean RMSE and have them be at approximately the same level of accuracy. If it is equally accurate, then we will end up meeting the specs mentioned by being within the accuracy denoted.

#### Test Conclusion

Currently, the amount of time it takes to finish collecting and processing the closest level of LBD is under a minute and as thus meets spec. More system integration is required in order to get a completely verified spec requirement for the accuracy of the calculations of angular velocity, angular acceleration, and angular jerk. However, the raw data after synchronization with provided Matlab functions seems to prove its feasibility. We will recreate all of the algorithms we require in C# and aim to achieve the same level of accuracy. We currently have Matlab programs ready and set up in order to perform the same analysis as shown in the Data section to prove its accuracy vs. the FSR at that time.

### Precision Test Overview

Using our integrated system, we plan to test the precision of the quantification of LBD by using our system to perform multiple real trials on ourselves and crafted dummy tests (or patients if we end up getting permission in the future to do so). Using the mean and standard deviation across all trials and multiple tests by using multiple people and dummy tests, we plan to verify that it is within a minimum 80% precision range of the mean for each individual trial.

#### Test Setup

For precision testing, we no longer need the Vicon. However, we still require the partially integrated system to supply accurate and synchronized data. Once our system is set up similarly to the accuracy test minus the Vicon, we begin doing testing. Depending on whether we get permission to do patient testing, we currently plan to use ourselves as example subjects. We will test each person on our team approximately 5 times and collect data and LBD severity. Once this is completed, we plan to take the average and standard deviation of each person and verify that the standard deviation from the mean of their ratings still keeps them within range of the specs provided. In addition, to fully test our system, we will also create a dummy test case where we will fake conditions for having a high severity of LBD to ensure that it stays precise under different test cases as well.

#### Data

Currently, there is no data that we have collected for precision testing since we are waiting to complete integration and accuracy testing.

#### Test Conclusion

Without data, we are unable to provide a definite conclusion. However, if the test described above passes it will be able to verify the precision spec of minimum 80% if the mean minus or plus standard deviation still stays within 80% of the rating of the mean.

# Subsystem Data Display Interface

After retrieving the quantification of the severity of Lower Back Disorder, patient information, and all of its supporting data, the Data Display is required to allow the user to display all of this information. In addition, the data display works as an interface for the user to input information required to perform testing, collect patient information, and set up the system. As such, it is required to provide button functionality for system set up.

## Significant Changes in Direction

Overall, this subsystem has had no major changes in direction.

## Subsystem Specifications

Table 8.1  
Data Display Interface specification Compliance Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| Specification | Min | Nominal | Max |
| Clear Display from 1-2ft | 1ft | 2ft | 2ft |
| Refresh Rate | 30 Hz | >60 Hz | N/A |
| Update upon completion of quantification | N/A | N/A | N/A |

Overall, this subsystem’s performance metrics are relative to the user. However, considering average human reaction, we can measure whether there is a noticeable time it takes to update. The display is relative, however, we will perform user tests on ourselves and asking for input from Dr. Jafari’s associates to ensure that it is clear enough to meet our specs. The refresh rate is simply dependant on the computer.

## Subsystem Status

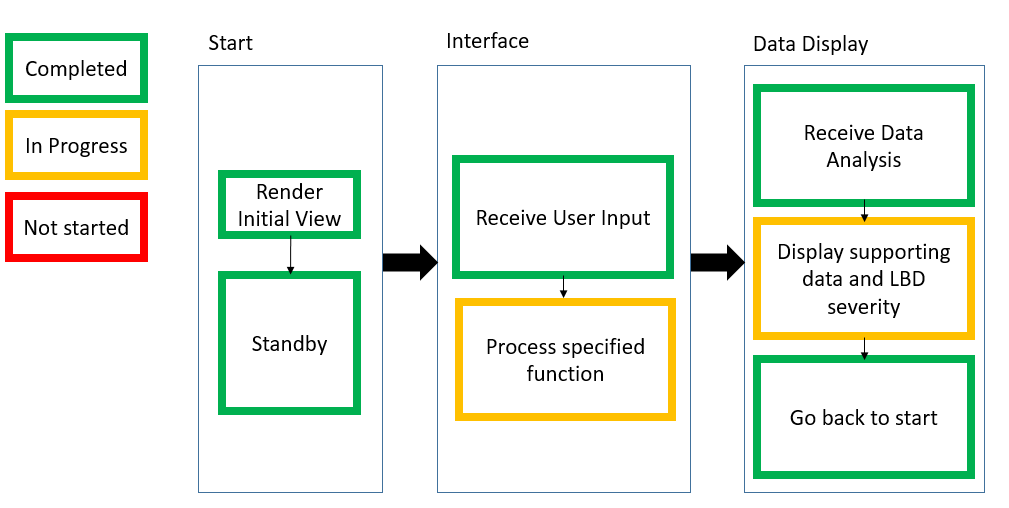


Fig 8.1 Data Display Flowchart

The Data Display Interface has currently been mostly completed. It can render the GUI and wait for user input, process most of the required functionality through buttons and text boxes to set up the IMU sensors, record data, and allow for patient data inputs required for the quantification, as well as provide easier testing of the lower back disorder model through setting the model’s respective factors. Lastly, it can properly receive and display this analysis and go back to continue testing. However, the specified functions of the user input is still being refined to ensure that it meets all of the needs for testing, such as implementing multiple trials, and the display is still being refined in order to make it more user friendly and clean to read.

## Subsystem Technical Details

The Data Display Interface is simply implemented as a display to be shown on a computer monitor and designed through the Windows Forms framework available through C#. To access this framework, I simply used visual studio designer to view and create an xml file for the data display and defined a set of text boxes to display the data and a set of text to denote what all of the data represents. Below is the code for writing the values to the form.

Code 8.1 Data Display code from DDI.cs

ageTxt.Text = "Age: " + ApplicationState.dataAnalysis.age.ToString();

if (ApplicationState.dataAnalysis.gender == true)

{

genderTxt.Text = "Gender: " + "Male";

}

else

{

genderTxt.Text = "Gender: " + "Female";

}

severityLBDTxt.Text = "LBD Severity: " + ApplicationState.dataAnalysis.severityLBD.ToString();

spROM15Txt.Text = "SP ROM15: " + ApplicationState.dataAnalysis.spROM15.ToString();

spROM30Txt.Text = "SP ROM30: " + ApplicationState.dataAnalysis.spROM30.ToString();

fpROMTxt.Text = "FP ROM: " + ApplicationState.dataAnalysis.fpROM.ToString();

eakVelTxt.Text = "Peak Angular Velocity: " + ApplicationState.dataAnalysis.peakSPAngVelocityAt0.ToString();

eakAccTxt.Text = "Peak Angular Acceleration: " + ApplicationState.dataAnalysis.peakSPAngAccelerationAt0.ToString();

peakJerkTxt.Text = "Peak Angular Jerk: " + ApplicationState.dataAnalysis.peakSPAngJerkAt0.ToString();

twistingROMTxt.Text = "Twisting ROM: " + ApplicationState.dataAnalysis.twistingROM.ToString();

I use a static data structure in C# in order to create what I call the ApplicationState structure which stores all of the information required for the display by passing in the data from the Database and Analysis structure and then write it to the text boxes using a simple ToString() function and assignment.

For the user input for testing, I created another xml file through the Main Window which allowed me to create text boxes and buttons to allow for the user to input the functionality such as specifying the bluetooth COM ports for the IMUs, the LBD model’s factors, and buttons for starting a trial, and ending a trial. The functions of these are processed and explained in more detail under the Database and Analysis which requires most of this set up and can be seen in section 7.4.

## Subsystem Testing

### Visibility Testing

Multiple subjective visibility tests will be performed by members of the team and our mentors including Dr. Jafari and his team to ensure that the GUI for the data display is clear from 1-2 ft away.

#### Test Setup

We will use a standard monitor which is at least approximately 16x16 inches and with at least a refresh rate of 30 Hz and then bring up our C# program and run it to show the Windows Forms display of our GUI and then have multiple people on our team and Dr. Jafari’s to verify that it is visible within 1-2 ft away.

#### Data

There is currently no data that we plan to have to collect. However, we do plan on getting feedback from Dr. Jafari and his team in order to ensure that it meets their standards. This feedback will be later provided and the updated Data Display after listening to said feedback will be captured. We can provide a current capture of the data display which shows the display of the information and the user interface for inputs.

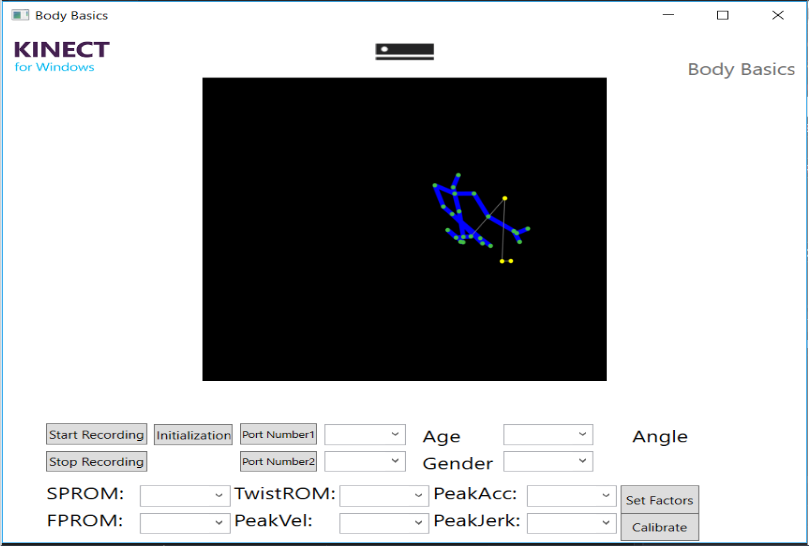


Fig 8.2 Data Display User Input Interface

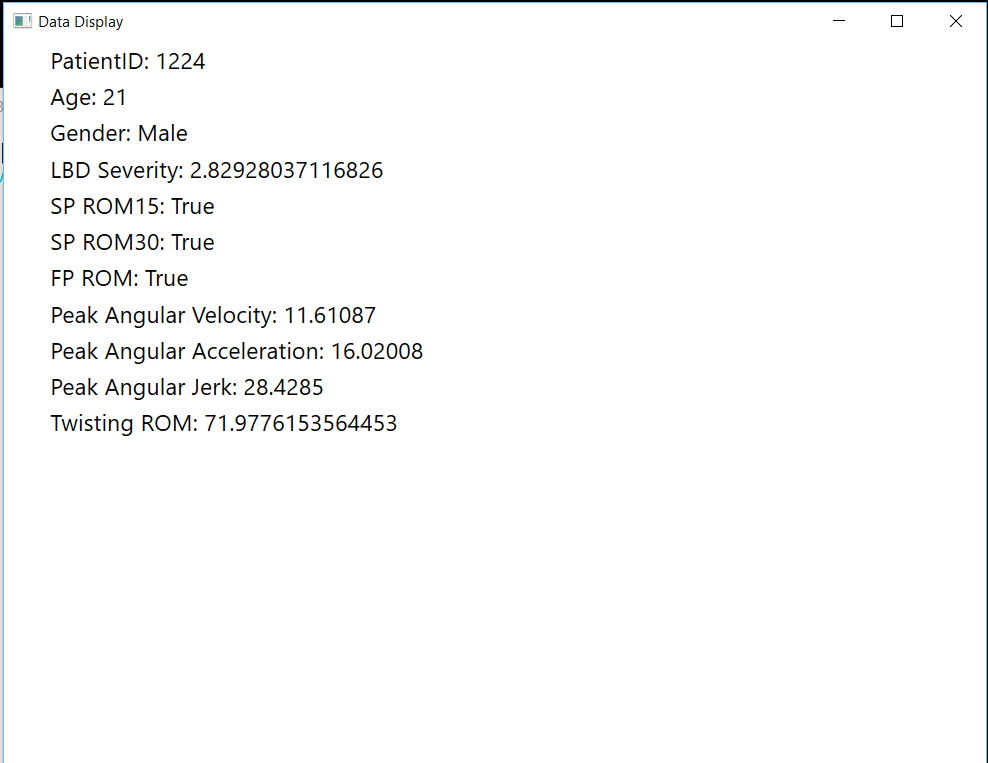


Fig 8.3 Data Display Interface

#### Test Conclusion

Since the monitor described in the test will have a refresh rate of at least 30 Hz, we will have verified at least one specification. However, since the visibility is largely subjective, there is no way to ensure that is going to be true for all people. For most regular users though, we plan to use this test to ensure that is up to our specification standards.

### Input and Updating Test

We plan to do user testing on the buttons we will provide on the Data Display and ensure that when they are pressed, they are responsive and perform the functionality we specify.

#### Test Setup

The Data Display will be set up as mentioned in the previous test. However, this time we are largely testing the inputs on the screen which will include buttons for parsing through the patient’s data such as display patient information and display patient test results. Once these are prompted for, we plan on taking a timestamp at the beginning of the prompt and at the end to ensure that it is responsive within noticeable human delay (within a few frames of the monitor’s refresh rate which will generally be approximately 8-16 ms depending on the monitor). In addition, we also plan to take a timestamp when the quantification completes and to when the data is displayed and update on the screen to see that it updates upon completion.

#### Data

Currently, there is no data to provide. However, the timestamps as mentioned will be recorded and the difference will be calculated to verify that the data display updates within noticeable human delay (a frame at the monitor’s refresh rate).

#### Test Conclusion

If the timestamps mentioned manage to update within a frame of the monitor’s refresh rate, then we will have verified that it updates upon completion. In addition, if we verify the specifications and check that the buttons display the information we specify properly, we will also have verified that it takes user input through buttons properly.

# Conclusions

This project is moving an appropriate pace to be completed by the end of the semester. The integration process was started ahead of schedule, as such some subsystems are still in the process of being refined for the final integration testing, but are on track to be finished successfully. Additionally, the early integration we’ve achieved has allowed for some potential system issues to be identified early rather than running into these at a later time. Moving forward, continued integrated system testing will be performed to further characterize the overall system error for reporting to the Spine Research Institute at Ohio State. Once subsystems have been integrated completely, further polishing will be attempted to bring this error as low as is possible to achieve the best results.

# 

# References

1. Marras, William S., et al. "The quantification of low back disorder using motion measures: Methodology and validation." Spine 24.20 (1999): 2091.

##### Project Budget

Table A-1   
Project Budget

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Quantity | Amount | Shipping | Line Total |
| Kinect V2 | 1 | $100.00 | N/A | $100.00 |
| Kinect Adapter for PC | 1 | $40.00 | N/A | $40.00 |
| Wet Suits | 2 | $20.00 | N/A | $40.00 |
|  |  |  | TOTAL | $180.00 |

As this project is a research project, this budget does not reflect the IMU sensors that were provided to us by Dr. Roozbeh Jafari. Inquiries into this technology should be directed towards him. It should also be noted that the funds used to purchase items in the above table were provided by Dr. Roozbeh Jafari for use in this project, and future use in the Embedded Signal Processing Labs.

##### Code

**<System Testing Code>**

%%%%%%%%%%%%%%%%%%%%%%%%%%%% VICON %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

filename = 'Ben\_Johnston Cal 01.csv';

V\_Data = xlsread(filename, 'A12:N1582');

pnts\_base(:,1) = V\_Data(:,3); %base points

pnts\_base(:,2) = V\_Data(:,4);

pnts\_base(:,3) = V\_Data(:,5);

pnts\_upper(:,1) = V\_Data(:,9); %upper points

pnts\_upper(:,2) = V\_Data(:,10);

pnts\_upper(:,3) = V\_Data(:,11);

v\_zunit = ([0 0 1]); %create z unit vector

Vic\_frames = V\_Data(:,1);

v\_pntpnt = pnts\_upper - pnts\_base; %point to point vector

iterator\_a=1;

v\_length = size(v\_pntpnt);

l = v\_length(1,1);

while iterator\_a<l

v\_pnt\_norm = v\_pntpnt(iterator\_a,:)./norm(v\_pntpnt(iterator\_a,:));

iterator\_a = iterator\_a+1;

theta(iterator\_a,:) = acos(dot(v\_pnt\_norm,v\_zunit));

end

alpha = (pi/2)-theta;

alpha\_deg = alpha.\*(180/pi);

%Syncing

[Vic\_pks, Vic\_locs] = findpeaks(alpha\_deg, 'MinPeakProminence', .5);

Vic\_peak\_beg = Vic\_locs(1)-200;

Vic\_peak\_end = Vic\_locs(end);

Frames\_used = Vic\_frames(Vic\_peak\_end)-Vic\_frames(Vic\_peak\_beg);

Vic\_time = (Frames\_used)/100;

Vic\_plot\_yaxis = alpha\_deg(Vic\_peak\_beg:Vic\_peak\_end);

Vic\_plot\_xaxis = 0:Vic\_time/(Frames\_used):Vic\_time;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% IMUs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%NOTE:

%Need to import GyroZ and Ltime columns from Bapgui

filenameSMid = 'T1S1.txt';

filenameSBase = 'T1S2.txt';

delimiterIn = ' ';

headerlinesIn\_IMU = 1;

SMid = importdata(filenameSMid, delimiterIn, headerlinesIn\_IMU);

SBase = importdata(filenameSBase, delimiterIn, headerlinesIn\_IMU);

%Program reports data using the z-axis of the gyroscope

%including angular position, velocity, acceleration and jerk,

%includes correction factor

%import data

gyroMid(:,1) = (SMid.data(:,4))./32.75;

gyroMid(:,2) = (SMid.data(:,5))./32.75;

gyroMid(:,3) = (SMid.data(:,6))./32.75; %Gyroscope correction factor

gyroBase(:,1) = (SBase.data(:,4))./32.75;

gyroBase(:,2) = (SBase.data(:,5))./32.75;

gyroBase(:,3) = (SBase.data(:,6))./32.75; %Gyroscope correction factor

LtimeMid = (SMid.data(:,10));

LtimeBase = (SBase.data(:,10));

tMid = transpose((LtimeMid-LtimeMid(1))./1000); %relative to start time, ms to s

tBase = transpose((LtimeBase-LtimeBase(1))./1000);

%\*\*\*\*\*\*\*low pass filter\*\*\*\*\*

x\_filter = designfilt('lowpassiir','FilterOrder',3,...

'PassbandFrequency',10e3,'PassbandRipple',0.5,...

'SampleRate',200e3);

gyroMid = filtfilt(x\_filter,gyroMid);

gyroBase = filtfilt(x\_filter,gyroBase);

%best fit code

%t3(:,1) = transpose(t);

%t3(:,2) = transpose(t);

%t3(:,3) = transpose(t);

%p1 = polyfit(t3,gyro,1);

%c\_velocity = transpose(polyval(p1,t)); %best fit line of velocity data

%kinect corrections

%IMU\_coeff = polyfit(transpose(IMU\_plot\_x),IMU\_plot\_func,1);

%IMU\_bestfit = transpose(polyval(IMU\_coeff,IMU\_plot\_x));

%mean\_kinect = mean(Kin\_plot\_func);

%mean\_kinect\_line = ones([length(Kin\_plot\_func),1]);

%mean\_kinect\_line = mean\_kinect\_line .\* mean\_kinect;

%IMU\_fusion = IMU\_bestfit.\*(-1);

%IMU\_fusion = IMU\_fusion + mean\_kinect;

%IMU\_corrected\_func = IMU\_plot\_func + IMU\_fusion;

%mean\_IMU\_line = ones([length(IMU\_plot\_func),1]).\*mean\_kinect;

%differentiation and integration

accelerationMid = diff(gyroMid); % vel to accel

accelerationMid = [0,[1 3];accelerationMid];

jerkMid = diff(accelerationMid); %accel to jerk

jerkMid = [0,[1 3];jerkMid];

positionMid = trapz(tMid,gyroMid);

distanceMid = cumtrapz(tMid,gyroMid); % vel to distance

distanceMid(:,2) = distanceMid(:,2) + 90;

accelerationBase = diff(gyroBase); % vel to accel

accelerationBase = [0,[1 3];accelerationBase];

jerkBase = diff(accelerationBase); %accel to jerk

jerkBase = [0,[1 3];jerkBase];

positionBase = trapz(tBase,gyroBase);

distanceBase = cumtrapz(tBase,gyroBase); % vel to distance

distanceBase(:,2) = distanceBase(:,2) + 90;

[SMid\_pks , SMid\_locs] = findpeaks(distanceMid(:,2), 'MinPeakProminence', 2);

% plot(tMid, distanceMid(:,2), tMid(SMid\_locs), SMid\_pks, 'or');

IMU\_str2end\_frame = SMid\_locs(end)-SMid\_locs(1);

IMU\_str2end\_time = tMid(SMid\_locs(end))-tMid(SMid\_locs(1));

IMU\_timesteps = IMU\_str2end\_time/IMU\_str2end\_frame;

IMU\_prev\_time = IMU\_str2end\_time+2;

IMU\_prev\_frms = 2/IMU\_timesteps;

SMid\_y\_axis = distanceMid(:,2);

SMid\_peak\_beg = SMid\_locs(1)-IMU\_prev\_frms;

SMid\_peak\_end = SMid\_locs(end);

%

% Frames\_used = Vic\_frames(Vic\_peak\_end)-Vic\_frames(Vic\_peak\_beg);

%

SMid\_time = IMU\_prev\_time;

SMid\_plot\_yaxis = SMid\_y\_axis(109:SMid\_peak\_end);

SMid\_plot\_xaxis = 0:SMid\_time/(SMid\_peak\_end-SMid\_peak\_beg):SMid\_time;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% KINECT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

filename1\_Kin = 'spinebaseT1.txt';

filename2\_Kin = 'spinemidT1.txt';

delimiterIn = ' ';

headerlinesIn\_Kin = 0;

spinebaseData = importdata(filename1\_Kin, delimiterIn, headerlinesIn\_Kin);

spinemidData = importdata(filename2\_Kin, delimiterIn, headerlinesIn\_Kin);

time = spinebaseData.data(:,1);

time = time - time(1);

time = transpose(time./1000);

pnts\_base\_Kin(:,1) = str2double(spinebaseData.textdata(:,1)); %base points

pnts\_base\_Kin(:,2) = str2double(spinebaseData.textdata(:,2));

pnts\_base\_Kin(:,3) = str2double(spinebaseData.textdata(:,3));

pnts\_upper\_Kin(:,1) = str2double(spinemidData.textdata(:,1)); %upper points

pnts\_upper\_Kin(:,2) = str2double(spinemidData.textdata(:,2));

pnts\_upper\_Kin(:,3) = str2double(spinemidData.textdata(:,3));

kinect\_yunit = ([0 1 0]); %create z unit vector

kinect\_pntpnt = pnts\_upper\_Kin - pnts\_base\_Kin; %point to point vector

iterator\_a=1;

kinect\_length = size(kinect\_pntpnt);

l = kinect\_length(1,1);

while iterator\_a<l

kinect\_pnt\_norm = kinect\_pntpnt(iterator\_a,:)./norm(kinect\_pntpnt(iterator\_a,:));

iterator\_a = iterator\_a+1;

theta\_Kin(iterator\_a,:) = acos(dot(kinect\_pnt\_norm,kinect\_yunit));

end

alpha\_Kin = (pi/2)-theta\_Kin;

alpha\_deg\_Kin = alpha\_Kin.\*(180/pi);

kin\_filter = designfilt('lowpassiir','FilterOrder',3,...

'PassbandFrequency',15e3,'PassbandRipple',0.5,...

'SampleRate',200e3);

alpha\_deg\_Kin\_filt = filtfilt(kin\_filter, alpha\_deg\_Kin);

[Kin\_pks , Kin\_locs] = findpeaks(alpha\_deg\_Kin\_filt, 'MinPeakProminence', 2);

Kin\_Frames=Kin\_locs(end)-Kin\_locs(1);

Kin\_time\_diff = time(Kin\_locs(end))-time(Kin\_locs(1));

Kin\_time\_step = Kin\_time\_diff/Kin\_Frames;

Kin\_added\_time = round(2/.0333);

Kin\_pks\_begin = Kin\_locs(1)-Kin\_added\_time;

Kin\_pks\_end = Kin\_locs(end);

Kin\_plot\_time = 0:Kin\_time\_diff/(Kin\_locs(end)-Kin\_locs(1)):Kin\_time\_diff+2;

Kin\_plot\_time(end+1) = 13.06;

Kin\_plot\_y = alpha\_deg\_Kin\_filt(Kin\_pks\_begin:Kin\_pks\_end);

%plot(Kin\_plot\_time, Kin\_plot\_y)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ENTER NAME %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%% PLOTTING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

subplot(4,1,1)

plot(Vic\_plot\_xaxis,Vic\_plot\_yaxis,SMid\_plot\_xaxis, SMid\_plot\_yaxis, Kin\_plot\_time, Kin\_plot\_y)

xlim([0 Vic\_time])

title('Angular Distance (deg)')

ylabel('x'),xlabel('Time (s)')

legend('Vicon','IMU', 'Kinect')

% subplot(3,1,2)

% xlim([0 SMid\_time])

% ylabel('y'),xlabel('Time (s)')

%subplot(3,1,3)

%plot(tMid,distanceMid(:,3), tBase,distanceBase(:,3))

%ylabel('z'),xlabel('Time (s)')

SMidVelocity = diff(SMid\_plot\_yaxis);

SMidAcceleration = diff(SMidVelocity);

SMidJerk = diff(SMidAcceleration);

VicVelocity = diff(Vic\_plot\_yaxis);

VicAccel = diff(VicVelocity);

VicJerk = diff(VicAccel);

KinVelocity = diff(Kin\_plot\_y);

KinAccel = diff(KinVelocity);

KinJerk = diff(KinAccel);

SMidVelocity = resample(SMidVelocity, length(VicVelocity),length(SMidVelocity));

SMidAcceleration = resample(SMidAcceleration, length(VicAccel),length(SMidAcceleration));

SMidJerk = resample(SMidJerk, length(VicJerk),length(SMidJerk));

SMid\_plot\_yaxis = resample(SMid\_plot\_yaxis,length(Vic\_plot\_yaxis),length(SMid\_plot\_yaxis));

SMid\_plot\_xaxis = resample(SMid\_plot\_xaxis,length(Vic\_plot\_xaxis),length(SMid\_plot\_xaxis));

Kin\_plot\_y = resample(Kin\_plot\_y,length(Vic\_plot\_yaxis),length(Kin\_plot\_y));

Kin\_plot\_time = resample(Kin\_plot\_time,length(Vic\_plot\_xaxis),length(Kin\_plot\_time));

angleRMSE\_IMU = sqrt(mean((Vic\_plot\_yaxis - SMid\_plot\_yaxis).^2))

angleRMSE\_Kin = sqrt(mean((Vic\_plot\_yaxis - Kin\_plot\_y).^2))

velRMSE\_IMU = sqrt(mean((VicVelocity - SMidVelocity).^2))

accelRMSE\_IMU = sqrt(mean((VicAccel - SMidAcceleration).^2))

jerkRMSE\_IMU = sqrt(mean((VicJerk - SMidJerk).^2))

velRMSEPlot = sqrt((VicVelocity - SMidVelocity).^2);

accelRMSEPlot = sqrt((VicAccel - SMidAcceleration).^2);

jerkRMSEPlot = sqrt((VicJerk - SMidJerk).^2);

%velRMSE\_plot\_xaxis = resample(velRMSEPlot,length(Vic\_plot\_xaxis),length(SMid\_plot\_xaxis));

subplot(4,1,2)

plot(velRMSEPlot)

title('IMU RMSE Angular Velocity')

ylabel('RMSE (%)'),xlabel('Frame')

subplot(4,1,3)

plot(accelRMSEPlot)

title('IMU RMSE Angular Acceleration')

ylabel('RMSE (%)'),xlabel('Frame')

subplot(4,1,4)

plot(jerkRMSEPlot)

title('IMU RMSE Angular Jerk')

ylabel('RMSE (%)'),xlabel('Frame')

**<Kinect and Feedback Code>**

using System;

using System.Collections.Generic;

using System.Windows;

using System.Text;

using System.Globalization;

using System.IO;

public class KinectFeedback

{

public bool isInitial;

public List<float> sagittalAngle;

public List<float> flexAngle;

public float currentSagittalAngle;

public float currentFlexAngle;

public List<float> initialPosRS;

public List<float> initialPosSS;

public List<float> initialPosSM;

public List<float> initialPosSB;

public string flexAngleTxt;

public string sagittalAngleTxt;

public string isZero;

public string isFifteen;

public string isThirty;

public string isFlex;

public KinectFeedback()

{

isInitial = true;

isZero = "False";

isFifteen = "False";

isThirty = "False";

isFlex = "False";

initialPosRS = new List<float>();

initialPosSS = new List<float>();

initialPosSM = new List<float>();

initialPosSB = new List<float>();

sagittalAngle = new List<float>();

flexAngle = new List<float>();

}

public float CalcSagittalAngleWithRespectToInitialPos(List<float> jointPos)

{

List<float> vector0 = new List<float>();

List<float> vector1 = new List<float>();

for (int i = 0; i < jointPos.Count; i++)

{

float posDiff = initialPosRS[i] - initialPosSS[i];

vector0.Add(posDiff);

}

for (int i = 0; i < jointPos.Count; i++)

{

float posDiff = jointPos[i] - initialPosSS[i];

vector1.Add(posDiff);

}

float dotProduct = DotProduct(vector0, vector1);

float magnitudeVector0 = CalcMagnitude(vector0);

float magnitudeVector1 = CalcMagnitude(vector1);

float cos = dotProduct / (magnitudeVector0 \* magnitudeVector1);

sagittalAngle.Add((float)(Math.Acos(cos) \* (180 / Math.PI)));

double angle = Math.Acos(cos) \* (180 / Math.PI);

sagittalAngleTxt = ((float)(angle)).ToString();

currentSagittalAngle = (float)(angle);

return (float)(angle);

}

public float CalcFlexAngleWithRespectToInitialPos(List<float> jointPos)

{

List<float> vector0 = new List<float>();

List<float> vector1 = new List<float>();

for (int i = 0; i < jointPos.Count; i++)

{

float posDiff = jointPos[i] - initialPosSB[i];

vector0.Add(posDiff);

}

for (int i = 0; i < jointPos.Count; i++)

{

//0 is the reference point of the Kinect

float posDiff = 0 - initialPosSM[i];

vector1.Add(posDiff);

}

float dotProduct = DotProduct(vector0, vector1);

float magnitudeVector0 = CalcMagnitude(vector0);

float magnitudeVector1 = CalcMagnitude(vector1);

float cos = dotProduct / (magnitudeVector0 \* magnitudeVector1);

flexAngle.Add((float)(Math.Acos(cos) \* (180 / Math.PI)));

double angle = Math.Acos(cos) \* (180 / Math.PI);

flexAngleTxt = ((float)(angle)).ToString();

currentFlexAngle = (float)(angle);

return (float)(angle);

}

private float DotProduct(List<float> vector1, List<float> vector2)

{

float dotProduct = 0;

for (int i = 0; i < vector1.Count; i++)

{

dotProduct += (vector1[i] \* vector2[i]);

}

return dotProduct;

}

private float CalcMagnitude(List<float> vector)

{

double magnitude = 0;

foreach (float pos in vector)

{

magnitude += pos \* pos;

}

magnitude = Math.Sqrt(magnitude);

return (float)(magnitude);

}

}

**<Signal Correction and Control Code>**

using System;

using System.IO;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace TestProgram

{

class Program

{

static void Main(string[] args)

{

// Initializations

double CorrFact;

List<double> Timestamps1 = new List<double>();

List<double> Timestamps2 = new List<double>();

List<double> Input1 = new List<double>();

List<double> Input2 = new List<double>();

List<double> Peaks\_Kinect = new List<double>();

List<double> Peaks\_IMU1 = new List<double>();

List<double> Peaks\_IMU2 = new List<double>();

int I1Eletracker = 0;

int I2Eletracker = 0;

double I1max = 0;

double I1cont = 0;

int I1Ele = 0;

double I2max = 0;

double I2cont = 0;

int I2Ele = 0;

double TimeI1 = 0;

double TimeI2 = 0;

int I1CorEle = 0;

int I2CorEle = 0;

string path1 = @"C:\Users\burns\Desktop\403\MATLAB Codes\test1\_1.txt";

string path2 = @"C:\Users\burns\Desktop\403\MATLAB Codes\test2\_1.txt";

string line;

//Read in txt file into lists for manipulation

/\*TextReader rdr = File.OpenText(path);

while ((line = rdr.ReadLine()) != null)

{

string text = rdr.ReadLine();

/\*string[] bits = text.Split(' '); //Incorrect read-in

double x = double.Parse(bits[0]);

double y = double.Parse(bits[1]);

Console.WriteLine(text);

}

rdr.Close();\*/

using (TextReader reader = File.OpenText(path1))

{

string readin1;

while ((readin1 = reader.ReadLine()) != null)

{

//string text = reader.ReadLine();

string[] bits = readin1.Split(new [] {' '}, StringSplitOptions.RemoveEmptyEntries);

double x = double.Parse(bits[0]);

double y = double.Parse(bits[1]);

Console.WriteLine(x);

Timestamps1.Add(x);

Console.WriteLine(y);

Input1.Add(y);

}

Console.WriteLine("To read in next file press any key");

System.Console.ReadKey();

}

using (TextReader reader2 = File.OpenText(path2))

{

string readin2;

while ((readin2 = reader2.ReadLine()) != null)

{

// string text2 = reader2.ReadLine();

string[] bits = readin2.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

double x2 = double.Parse(bits[0]);

double y2 = double.Parse(bits[1]);

Console.WriteLine(x2);

Timestamps2.Add(x2);

Console.WriteLine(y2);

Input2.Add(y2);

}

Console.WriteLine("Both Files read in, press any key");

System.Console.ReadKey();

}

//dummy test

/\* Timestamps1.Add(1);

Timestamps1.Add(2);

Timestamps1.Add(3);

Timestamps1.Add(4);

Timestamps1.Add(5);

Timestamps1.Add(6);

Timestamps1.Add(7);

Timestamps1.Add(8);

Timestamps2.Add(1);

Timestamps2.Add(2);

Timestamps2.Add(3);

Timestamps2.Add(4);

Timestamps2.Add(5);

Timestamps2.Add(6);

Timestamps2.Add(7);

Timestamps2.Add(8);

Input1.Add(1);

Input1.Add(2);

Input1.Add(3);

Input1.Add(4);

Input1.Add(6);

Input1.Add(3);

Input1.Add(2);

Input1.Add(1);

Input2.Add(1);

Input2.Add(2);

Input2.Add(3);

Input2.Add(4);

Input2.Add(5);

Input2.Add(6);

Input2.Add(10);

Input2.Add(5); \*/

//Math Algorithm

//First input Max Ele tracker

for (int i1iter = 0; i1iter < Input1.Count; i1iter++)

{

I1cont = Input1[I1Ele];

if (Math.Abs(I1cont) > I1max)

{

I1max = I1cont;

I1Eletracker = I1Ele;

}

I1Ele = I1Ele + 1;

}

Console.WriteLine("I1 Max is "+I1max);

Console.WriteLine("The timestamp for I1 at max is " + Timestamps1[I1Eletracker]);

//Second input Max Ele tracker

for (int i2iter = 0; i2iter < Input2.Count; i2iter++)

{

I2cont = Input2[i2iter];

if (Math.Abs(I2cont) > I2max)

{

I2max = I2cont;

I2Eletracker = I2Ele;

}

I2Ele = I2Ele + 1;

}

Console.WriteLine("I2 Max is "+I2max);

Console.WriteLine("The timestamp for I2 at max is " + Timestamps2[I2Eletracker]);

//Now have the elements for each input that the maximum resides in

//Next will extract timestamps

TimeI1 = Timestamps1[I1Eletracker];

TimeI2 = Timestamps2[I2Eletracker];

//Determines correction factor

CorrFact = Math.Abs(TimeI1 - TimeI2);

Console.WriteLine("Correction Factor is "+CorrFact);

Console.WriteLine("Press any key to see the new timestamps");

System.Console.ReadKey();

//Cases for sync, 1) TimeI1 longer than TimeI2 2)Time I2 longer than TimeI1

if (TimeI1 > TimeI2)

{

Console.WriteLine("The new timestamps for input two are ");

for (int i = 0; i < Input2.Count-1; i++)

{

Timestamps2[I2CorEle] = Timestamps2[I2CorEle] + CorrFact;

Console.WriteLine(Timestamps2[I2CorEle]);

I2CorEle = I2CorEle + 1;

}

Timestamps2[I2CorEle] = Timestamps2[I2CorEle] + CorrFact;

Console.WriteLine(Timestamps2[I2CorEle]);

}

else if (TimeI1 < TimeI2)

{

Console.WriteLine("The new timestamps for input one are ");

for (int i2 = 0; i2 < Input1.Count-1; i2++)

{

Timestamps1[I1CorEle] = Timestamps1[I1CorEle] + CorrFact;

Console.WriteLine(Timestamps1[I1CorEle]);

I1CorEle = I1CorEle + 1;

}

Timestamps1[I1CorEle] = Timestamps1[I1CorEle] + CorrFact;

Console.WriteLine(Timestamps1[I1CorEle]);

}

else

{

Console.WriteLine("No correction needed, signals synced.");

}

//Read Out after corrections

//Under construction... reading to data structure from data analysis

Console.WriteLine("Press any key to exit");

System.Console.ReadKey();

}

}

}

**<Database, Analysis, and Display Code>**

//------------------------------------------------------------------------------

// <copyright file="MainWindow.xaml.cs" company="Microsoft">

// Copyright (c) Microsoft Corporation. All rights reserved.

// </copyright>

//------------------------------------------------------------------------------

namespace Microsoft.Samples.Kinect.BodyBasics

{

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Globalization;

using System.IO;

using System.Windows;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using Microsoft.Kinect;

using System.IO.Ports;

using System.Data;

using System.Collections.Generic;

using System.Threading;

/// <summary>

/// Interaction logic for MainWindow

/// </summary>

public partial class MainWindow : Window, INotifyPropertyChanged

{

/// <summary>

/// Radius of drawn hand circles

/// </summary>

private const double HandSize = 30;

/// <summary>

/// Thickness of drawn joint lines

/// </summary>

private const double JointThickness = 3;

/// <summary>

/// Thickness of clip edge rectangles

/// </summary>

private const double ClipBoundsThickness = 10;

/// <summary>

/// Constant for clamping Z values of camera space points from being negative

/// </summary>

private const float InferredZPositionClamp = 0.1f;

/// <summary>

/// Brush used for drawing hands that are currently tracked as closed

/// </summary>

private readonly Brush handClosedBrush = new SolidColorBrush(Color.FromArgb(128, 255, 0, 0));

/// <summary>

/// Brush used for drawing hands that are currently tracked as opened

/// </summary>

private readonly Brush handOpenBrush = new SolidColorBrush(Color.FromArgb(128, 0, 255, 0));

/// <summary>

/// Brush used for drawing hands that are currently tracked as in lasso (pointer) position

/// </summary>

private readonly Brush handLassoBrush = new SolidColorBrush(Color.FromArgb(128, 0, 0, 255));

/// <summary>

/// Brush used for drawing joints that are currently tracked

/// </summary>

private readonly Brush trackedJointBrush = new SolidColorBrush(Color.FromArgb(255, 68, 192, 68));

/// <summary>

/// Brush used for drawing joints that are currently inferred

/// </summary>

private readonly Brush inferredJointBrush = Brushes.Yellow;

/// <summary>

/// Pen used for drawing bones that are currently inferred

/// </summary>

private readonly Pen inferredBonePen = new Pen(Brushes.Gray, 1);

/// <summary>

/// Drawing group for body rendering output

/// </summary>

private DrawingGroup drawingGroup;

/// <summary>

/// Drawing image that we will display

/// </summary>

private DrawingImage imageSource;

private SerialPort serialPort1 = new SerialPort();

private SerialPort serialPort2 = new SerialPort();

private SerialPort serialPort3 = new SerialPort();

private SerialPort serialPort4 = new SerialPort();

private byte[] RxPkt1 = new byte[50];

private byte[] RxPkt2 = new byte[50];

private byte[] RxPkt3 = new byte[50];

private byte[] RxPkt4 = new byte[50];

private int stop = 0;

private int keystart = 0;

private int kinect\_start = 0;

//private string move = "Kicking";

/\*private static FileStream fs\_kinect = new FileStream("C:/Users/Jian/01. Personal research/01. Sensor location calibration using kinect/data\_collection/Jian/Kinect.txt", FileMode.Create);

private StreamWriter sw1 = new StreamWriter(fs\_kinect);

private static FileStream fs\_sensor1 = new FileStream("C:/Users/Jian/01. Personal research/01. Sensor location calibration using kinect/data\_collection/Jian/Sensor1.txt", FileMode.Create);

private StreamWriter sw2 = new StreamWriter(fs\_sensor1);

private static FileStream fs\_sensor2 = new FileStream("C:/Users/Jian/01. Personal research/01. Sensor location calibration using kinect/data\_collection/Jian/Sensor2.txt", FileMode.Create);

private StreamWriter sw3 = new StreamWriter(fs\_sensor2);

private static FileStream fs\_sensor3 = new FileStream("C:/Users/Jian/01. Personal research/01. Sensor location calibration using kinect/data\_collection/Jian/Sensor3.txt", FileMode.Create);

private StreamWriter sw4 = new StreamWriter(fs\_sensor3);

private static FileStream fs\_sensor4 = new FileStream("C:/Users/Jian/01. Personal research/01. Sensor location calibration using kinect/data\_collection/Jian/Sensor4.txt", FileMode.Create);

private StreamWriter sw5 = new StreamWriter(fs\_sensor4);\*/

private static FileStream fs\_kinect\_spinebase;

private StreamWriter spineBaseSW;

private static FileStream fs\_kinect\_spinemid;

private StreamWriter spinemidSW;

private static FileStream fs\_kinect\_spineshoulder;

private StreamWriter spineshoulderSW;

private static FileStream fs\_kinect\_rightshoulder;

private StreamWriter rightshoulderSW;

private static FileStream fs\_sensor1;

private StreamWriter sw2;

private static FileStream fs\_sensor2;

private StreamWriter sw3;

private static FileStream fs\_sensor3;

private StreamWriter sw4;

private static FileStream fs\_sensor4;

private StreamWriter sw5;

//private long timestamp = 0;

private int bytesRead1 = 0;

private int bytesRead2 = 0;

private int bytesRead3 = 0;

private int bytesRead4 = 0;

private Queue<byte> data1 = new Queue<byte>(); //Queue structure to store all the bytes received

private byte[] buffer1 = new byte[4096]; //Buffer to hold data received in serial port

private Queue<byte> data2 = new Queue<byte>(); //Queue structure to store all the bytes received

private byte[] buffer2 = new byte[4096]; //Buffer to hold data received in serial port

private Queue<byte> data3 = new Queue<byte>(); //Queue structure to store all the bytes received

private byte[] buffer3 = new byte[4096]; //Buffer to hold data received in serial port

private Queue<byte> data4 = new Queue<byte>(); //Queue structure to store all the bytes received

private byte[] buffer4 = new byte[4096]; //Buffer to hold data received in serial port

private int threadCount = 4;

Thread[] ProcessSensorData;

/// <summary>

/// Active Kinect sensor

/// </summary>

private KinectSensor kinectSensor = null;

/// <summary>

/// Coordinate mapper to map one type of point to another

/// </summary>

private CoordinateMapper coordinateMapper = null;

/// <summary>

/// Reader for body frames

/// </summary>

private BodyFrameReader bodyFrameReader = null;

/// <summary>

/// Array for the bodies

/// </summary>

private Body[] bodies = null;

/// <summary>

/// definition of bones

/// </summary>

private List<Tuple<JointType, JointType>> bones;

/// <summary>

/// Width of display (depth space)

/// </summary>

private int displayWidth;

/// <summary>

/// Height of display (depth space)

/// </summary>

private int displayHeight;

/// <summary>

/// List of colors for each body tracked

/// </summary>

private List<Pen> bodyColors;

/// <summary>

/// Current status text to display

/// </summary>

private string statusText = null;

private SensorData sensorData = new SensorData();

private DataAnalysis dataAnalysis;

private KinectFeedback kinectFeedback = new KinectFeedback();

/// <summary>

/// Initializes a new instance of the MainWindow class.

/// </summary>

public MainWindow()

{

// one sensor is currently supported

this.kinectSensor = KinectSensor.GetDefault();

// get the coordinate mapper

this.coordinateMapper = this.kinectSensor.CoordinateMapper;

// get the depth (display) extents

FrameDescription frameDescription = this.kinectSensor.DepthFrameSource.FrameDescription;

// get size of joint space

this.displayWidth = frameDescription.Width;

this.displayHeight = frameDescription.Height;

// open the reader for the body frames

this.bodyFrameReader = this.kinectSensor.BodyFrameSource.OpenReader();

// a bone defined as a line between two joints

this.bones = new List<Tuple<JointType, JointType>>();

// Torso

this.bones.Add(new Tuple<JointType, JointType>(JointType.Head, JointType.Neck));

this.bones.Add(new Tuple<JointType, JointType>(JointType.Neck, JointType.SpineShoulder));

this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineShoulder, JointType.SpineMid));

this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineMid, JointType.SpineBase));

this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineShoulder, JointType.ShoulderRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineShoulder, JointType.ShoulderLeft));

this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineBase, JointType.HipRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.SpineBase, JointType.HipLeft));

// Right Arm

this.bones.Add(new Tuple<JointType, JointType>(JointType.ShoulderRight, JointType.ElbowRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.ElbowRight, JointType.WristRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.WristRight, JointType.HandRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.HandRight, JointType.HandTipRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.WristRight, JointType.ThumbRight));

// Left Arm

this.bones.Add(new Tuple<JointType, JointType>(JointType.ShoulderLeft, JointType.ElbowLeft));

this.bones.Add(new Tuple<JointType, JointType>(JointType.ElbowLeft, JointType.WristLeft));

this.bones.Add(new Tuple<JointType, JointType>(JointType.WristLeft, JointType.HandLeft));

this.bones.Add(new Tuple<JointType, JointType>(JointType.HandLeft, JointType.HandTipLeft));

this.bones.Add(new Tuple<JointType, JointType>(JointType.WristLeft, JointType.ThumbLeft));

// Right Leg

this.bones.Add(new Tuple<JointType, JointType>(JointType.HipRight, JointType.KneeRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.KneeRight, JointType.AnkleRight));

this.bones.Add(new Tuple<JointType, JointType>(JointType.AnkleRight, JointType.FootRight));

// Left Leg

this.bones.Add(new Tuple<JointType, JointType>(JointType.HipLeft, JointType.KneeLeft));

this.bones.Add(new Tuple<JointType, JointType>(JointType.KneeLeft, JointType.AnkleLeft));

this.bones.Add(new Tuple<JointType, JointType>(JointType.AnkleLeft, JointType.FootLeft));

// populate body colors, one for each BodyIndex

this.bodyColors = new List<Pen>();

this.bodyColors.Add(new Pen(Brushes.Red, 6));

this.bodyColors.Add(new Pen(Brushes.Orange, 6));

this.bodyColors.Add(new Pen(Brushes.Green, 6));

this.bodyColors.Add(new Pen(Brushes.Blue, 6));

this.bodyColors.Add(new Pen(Brushes.Indigo, 6));

this.bodyColors.Add(new Pen(Brushes.Violet, 6));

// set IsAvailableChanged event notifier

this.kinectSensor.IsAvailableChanged += this.Sensor\_IsAvailableChanged;

// open the sensor

this.kinectSensor.Open();

// set the status text

this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText

: Properties.Resources.NoSensorStatusText;

// Create the drawing group we'll use for drawing

this.drawingGroup = new DrawingGroup();

// Create an image source that we can use in our image control

this.imageSource = new DrawingImage(this.drawingGroup);

// use the window object as the view model in this simple example

this.DataContext = this;

// initialize the components (controls) of the window

this.InitializeComponent();

ProcessSensorData = new Thread[threadCount];

(ProcessSensorData[0] = new Thread(Processing1)).Start();

(ProcessSensorData[1] = new Thread(Processing2)).Start();

(ProcessSensorData[2] = new Thread(Processing3)).Start();

(ProcessSensorData[3] = new Thread(Processing4)).Start();

}

/// <summary>

/// INotifyPropertyChangedPropertyChanged event to allow window controls to bind to changeable data

/// </summary>

public event PropertyChangedEventHandler PropertyChanged;

/// <summary>

/// Gets the bitmap to display

/// </summary>

public ImageSource ImageSource

{

get

{

return this.imageSource;

}

}

/// <summary>

/// Gets or sets the current status text to display

/// </summary>

public string StatusText

{

get

{

return this.statusText;

}

set

{

if (this.statusText != value)

{

this.statusText = value;

// notify any bound elements that the text has changed

if (this.PropertyChanged != null)

{

this.PropertyChanged(this, new PropertyChangedEventArgs("StatusText"));

}

}

}

}

/// <summary>

/// Execute start up tasks

/// </summary>

/// <param name="sender">object sending the event</param>

/// <param name="e">event arguments</param>

private void MainWindow\_Loaded(object sender, RoutedEventArgs e)

{

if (this.bodyFrameReader != null)

{

this.bodyFrameReader.FrameArrived += this.Reader\_FrameArrived;

}

}

/// <summary>

/// Execute shutdown tasks

/// </summary>

/// <param name="sender">object sending the event</param>

/// <param name="e">event arguments</param>

private void MainWindow\_Closing(object sender, CancelEventArgs e)

{

if (this.bodyFrameReader != null)

{

// BodyFrameReader is IDisposable

this.bodyFrameReader.Dispose();

this.bodyFrameReader = null;

}

if (this.kinectSensor != null)

{

this.kinectSensor.Close();

this.kinectSensor = null;

}

}

/// <summary>

/// Handles the body frame data arriving from the sensor

/// </summary>

/// <param name="sender">object sending the event</param>

/// <param name="e">event arguments</param>

///

List<List<float>> kinectData = default(List<List<float>>);

private void Reader\_FrameArrived(object sender, BodyFrameArrivedEventArgs e)

{

bool dataReceived = false;

using (BodyFrame bodyFrame = e.FrameReference.AcquireFrame())

{

if (bodyFrame != null)

{

if (this.bodies == null)

{

this.bodies = new Body[bodyFrame.BodyCount];

}

// The first time GetAndRefreshBodyData is called, Kinect will allocate each Body in the array.

// As long as those body objects are not disposed and not set to null in the array,

// those body objects will be re-used.

bodyFrame.GetAndRefreshBodyData(this.bodies);

dataReceived = true;

}

}

if (kinect\_start != 0)

{

foreach (Body body in bodies)

{

if (body.IsTracked == true)

{

DateTime datenow = DateTime.Now;

int hour = datenow.Hour;

int minute = datenow.Minute;

int second = datenow.Second;

int millisecond = datenow.Millisecond;

int timestamp = hour \* 3600 \* 1000 + minute \* 60 \* 1000 + second \* 1000 + millisecond;

List<float> spineBaseData = new List<float>();

List<float> spineMidData = new List<float>();

List<float> spineShoulderData = new List<float>();

List<float> rightShoulderData = new List<float>();

List<List<float>> kinectData = new List<List<float>>();

if(kinectFeedback.isInitial == true)

{

kinectFeedback.initialPosRS.Add(body.Joints[JointType.ShoulderRight].Position.X);

kinectFeedback.initialPosSS.Add(body.Joints[JointType.SpineShoulder].Position.X);

kinectFeedback.initialPosRS.Add(body.Joints[JointType.ShoulderRight].Position.Y);

kinectFeedback.initialPosSS.Add(body.Joints[JointType.SpineShoulder].Position.Y);

kinectFeedback.initialPosRS.Add(body.Joints[JointType.ShoulderRight].Position.Z);

kinectFeedback.initialPosSS.Add(body.Joints[JointType.SpineShoulder].Position.Z);

kinectFeedback.isInitial = false;

}

else

{

List<float> rightShoulderPos = new List<float>();

rightShoulderPos.Add(body.Joints[JointType.ShoulderRight].Position.X);

rightShoulderPos.Add(body.Joints[JointType.ShoulderRight].Position.Y);

rightShoulderPos.Add(body.Joints[JointType.ShoulderRight].Position.Z);

kinectFeedback.CalcAngleWithRespectToInitialPos(rightShoulderPos);

angleTxt.Text = kinectFeedback.CalcAngleWithRespectToInitialPos(rightShoulderPos).ToString();

}

//Collect Spine Base Data

spineBaseData.Add(body.Joints[JointType.SpineBase].Position.X);

spineBaseData.Add(body.Joints[JointType.SpineBase].Position.Y);

spineBaseData.Add(body.Joints[JointType.SpineBase].Position.Z);

if(body.Joints[JointType.SpineBase].TrackingState == TrackingState.Tracked)

{

spineBaseData.Add(1);

}

else

{

spineBaseData.Add(0);

}

spineBaseData.Add(timestamp);

//Collect Spine Mid Data

spineMidData.Add(body.Joints[JointType.SpineMid].Position.X);

spineMidData.Add(body.Joints[JointType.SpineMid].Position.Y);

spineMidData.Add(body.Joints[JointType.SpineMid].Position.Z);

if (body.Joints[JointType.SpineMid].TrackingState == TrackingState.Tracked)

{

spineMidData.Add(1);

}

else

{

spineMidData.Add(0);

}

spineMidData.Add(timestamp);

//Collect Spine Shoulder Data

spineShoulderData.Add(body.Joints[JointType.SpineShoulder].Position.X);

spineShoulderData.Add(body.Joints[JointType.SpineShoulder].Position.Y);

spineShoulderData.Add(body.Joints[JointType.SpineShoulder].Position.Z);

if (body.Joints[JointType.SpineShoulder].TrackingState == TrackingState.Tracked)

{

spineShoulderData.Add(1);

}

else

{

spineShoulderData.Add(0);

}

spineShoulderData.Add(timestamp);

//Collect Right Shoulder Data

rightShoulderData.Add(body.Joints[JointType.ShoulderRight].Position.X);

rightShoulderData.Add(body.Joints[JointType.ShoulderRight].Position.Y);

rightShoulderData.Add(body.Joints[JointType.ShoulderRight].Position.Z);

if (body.Joints[JointType.ShoulderRight].TrackingState == TrackingState.Tracked)

{

rightShoulderData.Add(1);

}

else

{

rightShoulderData.Add(0);

}

rightShoulderData.Add(timestamp);

//Collect lists of kinect data

kinectData.Add(spineBaseData);

kinectData.Add(spineMidData);

kinectData.Add(spineShoulderData);

kinectData.Add(rightShoulderData);

//Write Kinect Joint Data to text files

spineBaseSW.WriteLine(body.Joints[JointType.SpineBase].Position.X + " " + body.Joints[JointType.SpineBase].Position.Y + " " + body.Joints[JointType.SpineBase].Position.Z + " " + body.Joints[JointType.SpineBase].TrackingState + " " + timestamp);

spinemidSW.WriteLine(body.Joints[JointType.SpineMid].Position.X + " " + body.Joints[JointType.SpineMid].Position.Y + " " + body.Joints[JointType.SpineMid].Position.Z + " " + body.Joints[JointType.SpineMid].TrackingState + " " + timestamp);

spineshoulderSW.WriteLine(body.Joints[JointType.SpineShoulder].Position.X + " " + body.Joints[JointType.SpineShoulder].Position.Y + " " + body.Joints[JointType.SpineShoulder].Position.Z + " " + body.Joints[JointType.SpineShoulder].TrackingState + " " + timestamp);

rightshoulderSW.WriteLine(body.Joints[JointType.ShoulderRight].Position.X + " " + body.Joints[JointType.ShoulderRight].Position.Y + " " + body.Joints[JointType.ShoulderRight].Position.Z + " " + body.Joints[JointType.ShoulderRight].TrackingState + " " + timestamp);

}

}

}

if (dataReceived)

{

using (DrawingContext dc = this.drawingGroup.Open())

{

// Draw a transparent background to set the render size

dc.DrawRectangle(Brushes.Black, null, new Rect(0.0, 0.0, this.displayWidth, this.displayHeight));

int penIndex = 0;

foreach (Body body in this.bodies)

{

Pen drawPen = this.bodyColors[penIndex++];

if (body.IsTracked)

{

this.DrawClippedEdges(body, dc);

IReadOnlyDictionary<JointType, Joint> joints = body.Joints;

// convert the joint points to depth (display) space

Dictionary<JointType, Point> jointPoints = new Dictionary<JointType, Point>();

foreach (JointType jointType in joints.Keys)

{

// sometimes the depth(Z) of an inferred joint may show as negative

// clamp down to 0.1f to prevent coordinatemapper from returning (-Infinity, -Infinity)

CameraSpacePoint position = joints[jointType].Position;

if (position.Z < 0)

{

position.Z = InferredZPositionClamp;

}

DepthSpacePoint depthSpacePoint = this.coordinateMapper.MapCameraPointToDepthSpace(position);

jointPoints[jointType] = new Point(depthSpacePoint.X, depthSpacePoint.Y);

}

this.DrawBody(joints, jointPoints, dc, drawPen);

this.DrawHand(body.HandLeftState, jointPoints[JointType.HandLeft], dc);

this.DrawHand(body.HandRightState, jointPoints[JointType.HandRight], dc);

}

}

// prevent drawing outside of our render area

this.drawingGroup.ClipGeometry = new RectangleGeometry(new Rect(0.0, 0.0, this.displayWidth, this.displayHeight));

}

}

}

/// <summary>

/// Draws a body

/// </summary>

/// <param name="joints">joints to draw</param>

/// <param name="jointPoints">translated positions of joints to draw</param>

/// <param name="drawingContext">drawing context to draw to</param>

/// <param name="drawingPen">specifies color to draw a specific body</param>

private void DrawBody(IReadOnlyDictionary<JointType, Joint> joints, IDictionary<JointType, Point> jointPoints, DrawingContext drawingContext, Pen drawingPen)

{

// Draw the bones

foreach (var bone in this.bones)

{

this.DrawBone(joints, jointPoints, bone.Item1, bone.Item2, drawingContext, drawingPen);

}

// Draw the joints

foreach (JointType jointType in joints.Keys)

{

Brush drawBrush = null;

TrackingState trackingState = joints[jointType].TrackingState;

if (trackingState == TrackingState.Tracked)

{

drawBrush = this.trackedJointBrush;

}

else if (trackingState == TrackingState.Inferred)

{

drawBrush = this.inferredJointBrush;

}

if (drawBrush != null)

{

drawingContext.DrawEllipse(drawBrush, null, jointPoints[jointType], JointThickness, JointThickness);

}

}

}

/// <summary>

/// Draws one bone of a body (joint to joint)

/// </summary>

/// <param name="joints">joints to draw</param>

/// <param name="jointPoints">translated positions of joints to draw</param>

/// <param name="jointType0">first joint of bone to draw</param>

/// <param name="jointType1">second joint of bone to draw</param>

/// <param name="drawingContext">drawing context to draw to</param>

/// /// <param name="drawingPen">specifies color to draw a specific bone</param>

private void DrawBone(IReadOnlyDictionary<JointType, Joint> joints, IDictionary<JointType, Point> jointPoints, JointType jointType0, JointType jointType1, DrawingContext drawingContext, Pen drawingPen)

{

Joint joint0 = joints[jointType0];

Joint joint1 = joints[jointType1];

// If we can't find either of these joints, exit

if (joint0.TrackingState == TrackingState.NotTracked ||

joint1.TrackingState == TrackingState.NotTracked)

{

return;

}

// We assume all drawn bones are inferred unless BOTH joints are tracked

Pen drawPen = this.inferredBonePen;

if ((joint0.TrackingState == TrackingState.Tracked) && (joint1.TrackingState == TrackingState.Tracked))

{

drawPen = drawingPen;

}

drawingContext.DrawLine(drawPen, jointPoints[jointType0], jointPoints[jointType1]);

}

/// <summary>

/// Draws a hand symbol if the hand is tracked: red circle = closed, green circle = opened; blue circle = lasso

/// </summary>

/// <param name="handState">state of the hand</param>

/// <param name="handPosition">position of the hand</param>

/// <param name="drawingContext">drawing context to draw to</param>

private void DrawHand(HandState handState, Point handPosition, DrawingContext drawingContext)

{

switch (handState)

{

case HandState.Closed:

drawingContext.DrawEllipse(this.handClosedBrush, null, handPosition, HandSize, HandSize);

break;

case HandState.Open:

drawingContext.DrawEllipse(this.handOpenBrush, null, handPosition, HandSize, HandSize);

break;

case HandState.Lasso:

drawingContext.DrawEllipse(this.handLassoBrush, null, handPosition, HandSize, HandSize);

break;

}

}

/// <summary>

/// Draws indicators to show which edges are clipping body data

/// </summary>

/// <param name="body">body to draw clipping information for</param>

/// <param name="drawingContext">drawing context to draw to</param>

private void DrawClippedEdges(Body body, DrawingContext drawingContext)

{

FrameEdges clippedEdges = body.ClippedEdges;

if (clippedEdges.HasFlag(FrameEdges.Bottom))

{

drawingContext.DrawRectangle(

Brushes.Red,

null,

new Rect(0, this.displayHeight - ClipBoundsThickness, this.displayWidth, ClipBoundsThickness));

}

if (clippedEdges.HasFlag(FrameEdges.Top))

{

drawingContext.DrawRectangle(

Brushes.Red,

null,

new Rect(0, 0, this.displayWidth, ClipBoundsThickness));

}

if (clippedEdges.HasFlag(FrameEdges.Left))

{

drawingContext.DrawRectangle(

Brushes.Red,

null,

new Rect(0, 0, ClipBoundsThickness, this.displayHeight));

}

if (clippedEdges.HasFlag(FrameEdges.Right))

{

drawingContext.DrawRectangle(

Brushes.Red,

null,

new Rect(this.displayWidth - ClipBoundsThickness, 0, ClipBoundsThickness, this.displayHeight));

}

}

/// <summary>

/// Handles the event which the sensor becomes unavailable (E.g. paused, closed, unplugged).

/// </summary>

/// <param name="sender">object sending the event</param>

/// <param name="e">event arguments</param>

private void Sensor\_IsAvailableChanged(object sender, IsAvailableChangedEventArgs e)

{

// on failure, set the status text

this.StatusText = this.kinectSensor.IsAvailable ? Properties.Resources.RunningStatusText

: Properties.Resources.SensorNotAvailableStatusText;

}

// start create files for Kinect and sensor and start to record data

private void button1\_Click(object sender, RoutedEventArgs e)

{

//string fpath = "C:/Users/Alex/Documents/BodyBasics-WPF-IntegratedSensors/SD01/";

//string fpath = "C:/Users/BennyChan/OneDrive/Documentos/ECEN 403/Team7/BennyChan/BodyBasics-WPF-IntegratedSensorsUpdated/SD01";

//string fpath = "C:/Users/BennyChan/Documents/BodyBasics-WPF-IntegratedSensors -MatlabUpdate/BodyBasics-WPF-IntegratedSensors/SD01/";

string fpath = "E:/OneDrive/Documentos/ECEN 403/Team7/AlexDubois/BodyBasics-WPF-IntegratedSensorsUpdated/SD01/";

string[] dirs = Directory.GetFiles(fpath);

int num = dirs.Length;

fs\_kinect\_spinebase = new FileStream(string.Concat(fpath,"spinebase.txt"), FileMode.Create);

spineBaseSW = new StreamWriter(fs\_kinect\_spinebase);

fs\_kinect\_spinemid = new FileStream(string.Concat(fpath,"spinemid.txt"), FileMode.Create);

spinemidSW = new StreamWriter(fs\_kinect\_spinemid);

fs\_kinect\_spineshoulder = new FileStream(string.Concat(fpath, "spineshoulder.txt"), FileMode.Create);

spineshoulderSW = new StreamWriter(fs\_kinect\_spineshoulder);

fs\_kinect\_rightshoulder = new FileStream(string.Concat(fpath, "rightshoulder.txt"), FileMode.Create);

rightshoulderSW = new StreamWriter(fs\_kinect\_rightshoulder);

fs\_sensor1 = new FileStream(string.Concat(fpath, string.Concat((num + 2).ToString(), ".txt")), FileMode.Create);

sw2 = new StreamWriter(fs\_sensor1);

fs\_sensor2 = new FileStream(string.Concat(fpath, string.Concat((num + 3).ToString(), ".txt")), FileMode.Create);

sw3 = new StreamWriter(fs\_sensor2);

//fs\_sensor3 = new FileStream(string.Concat("C:/Users/Jian/01. Personal research/01. Sensor location calibration using kinect/data\_collection/Jian/Kevin/", string.Concat((num + 4).ToString(), ".txt")), FileMode.Create);

//sw4 = new StreamWriter(fs\_sensor3);

//fs\_sensor4 = new FileStream(string.Concat("C:/Users/Jian/01. Personal research/01. Sensor location calibration using kinect/data\_collection/Jian/Kevin/", string.Concat((num + 5).ToString(), ".txt")), FileMode.Create);

//sw5 = new StreamWriter(fs\_sensor4);

kinect\_start = 1;

button1.IsEnabled = false;

ButtonStop.IsEnabled = true;

}

private void button3\_Click(object sender, RoutedEventArgs e)

{

kinect\_start = 0;

stop = 1;

spineBaseSW.Close();

spinemidSW.Close();

spineshoulderSW.Close();

rightshoulderSW.Close();

sw2.Close();

sw3.Close();

//sw4.Close();

//sw5.Close();

fs\_kinect\_spinebase.Close();

fs\_kinect\_spinemid.Close();

fs\_kinect\_spineshoulder.Close();

fs\_kinect\_rightshoulder.Close();

fs\_sensor1.Close();

fs\_sensor2.Close();

//fs\_sensor3.Close();

//fs\_sensor4.Close();

button1.IsEnabled = true;

ButtonStop.IsEnabled = false;

}

private void DataReceivedHandler1(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)

{

bytesRead1 = serialPort1.Read(buffer1, 0, buffer1.Length);

lock (data1)

{

for (int i = 0; i < bytesRead1; i++)

{

data1.Enqueue(buffer1[i]);

}

Monitor.Pulse(data1);

}

}

private void DataReceivedHandler2(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)

{

bytesRead2 = serialPort2.Read(buffer2, 0, buffer2.Length);

lock (data2)

{

for (int i = 0; i < bytesRead2; i++)

{

data2.Enqueue(buffer2[i]);

}

Monitor.Pulse(data2);

}

}

private void DataReceivedHandler3(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)

{

bytesRead3 = serialPort3.Read(buffer3, 0, buffer3.Length);

lock (data3)

{

for (int i = 0; i < bytesRead3; i++)

{

data3.Enqueue(buffer3[i]);

}

Monitor.Pulse(data3);

}

}

private void DataReceivedHandler4(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)

{

bytesRead4 = serialPort4.Read(buffer4, 0, buffer4.Length);

lock (data4)

{

for (int i = 0; i < bytesRead4; i++)

{

data4.Enqueue(buffer4[i]);

}

Monitor.Pulse(data4);

}

}

List<List<Int16>> wearableData = default(List<List<Int16>>);

private void Processing1()

{

while (true)

{

lock (data1)

{

while (data1.Count < 50)

{

Monitor.Wait(data1);

}

int i = 1;

while (i == 1)

{

RxPkt1[0] = data1.Dequeue();

if (RxPkt1[0] == 16)

{

RxPkt1[1] = data1.Dequeue();

if (RxPkt1[1] == 1)

{

int j = 1;

int k = 2;

while (j == 1)

{

RxPkt1[k] = data1.Dequeue();

if (RxPkt1[k] == 16)

{

RxPkt1[k + 1] = data1.Dequeue();

if (RxPkt1[k + 1] == 4)

{

j = 0;

i = 0;

}

}

k++;

}

}

}

}

byte[] convert = new byte[26];

for (i = 0; i < 26; i++)

{

convert[i] = RxPkt1[i];

}

Array.Reverse(convert);//operating system is little endians while the package is big endians, reverse the array.

if (kinect\_start != 0)

{

DateTime datenow = DateTime.Now;

List<Int16> wearable = default(List<Int16>);

int hour = datenow.Hour;

int minute = datenow.Minute;

int second = datenow.Second;

int millisecond = datenow.Millisecond;

int timestampS = hour \* 3600 \* 1000 + minute \* 60 \* 1000 + second \* 1000 + millisecond;

wearable.Add(BitConverter.ToInt16(convert, 22));

wearable.Add(BitConverter.ToInt16(convert, 20));

wearable.Add(BitConverter.ToInt16(convert, 18));

wearable.Add(BitConverter.ToInt16(convert, 16));

wearable.Add(BitConverter.ToInt16(convert, 14));

wearable.Add(BitConverter.ToInt16(convert, 12));

wearable.Add(BitConverter.ToInt16(convert, 10));

wearable.Add(BitConverter.ToInt16(convert, 8));

wearable.Add(BitConverter.ToInt16(convert, 6));

wearable.Add((Int16)(timestampS));

wearableData.Add(wearable);

sw2.WriteLine(BitConverter.ToInt16(convert, 22) + " " + BitConverter.ToInt16(convert, 20) + " " + BitConverter.ToInt16(convert, 18) + " " + BitConverter.ToInt16(convert, 16) + " " + BitConverter.ToInt16(convert, 14) + " " + BitConverter.ToInt16(convert, 12) + " " + BitConverter.ToInt16(convert, 10) + " " + BitConverter.ToInt16(convert, 8) + " " + BitConverter.ToInt16(convert, 6) + " " + timestampS);

}

}

}

}

private void Processing2()

{

while (true)

{

lock (data2)

{

while (data2.Count < 50)

{

Monitor.Wait(data2);

}

int i = 1;

while (i == 1)

{

RxPkt2[0] = data2.Dequeue();

if (RxPkt2[0] == 16)

{

RxPkt2[1] = data2.Dequeue();

if (RxPkt2[1] == 1)

{

int j = 1;

int k = 2;

while (j == 1)

{

RxPkt2[k] = data2.Dequeue();

if (RxPkt2[k] == 16)

{

RxPkt2[k + 1] = data2.Dequeue();

if (RxPkt2[k + 1] == 4)

{

j = 0;

i = 0;

}

}

k++;

}

}

}

}

byte[] convert = new byte[26];

for (i = 0; i < 26; i++)

{

convert[i] = RxPkt2[i];

}

Array.Reverse(convert);//operating system is little endians while the package is big endians, reverse the array.

if (kinect\_start != 0)

{

DateTime datenow = DateTime.Now;

int hour = datenow.Hour;

int minute = datenow.Minute;

int second = datenow.Second;

int millisecond = datenow.Millisecond;

int timestampS2 = hour \* 3600 \* 1000 + minute \* 60 \* 1000 + second \* 1000 + millisecond;

sw3.WriteLine(BitConverter.ToInt16(convert, 22) + " " + BitConverter.ToInt16(convert, 20) + " " + BitConverter.ToInt16(convert, 18) + " " + BitConverter.ToInt16(convert, 16) + " " + BitConverter.ToInt16(convert, 14) + " " + BitConverter.ToInt16(convert, 12) + " " + BitConverter.ToInt16(convert, 10) + " " + BitConverter.ToInt16(convert, 8) + " " + BitConverter.ToInt16(convert, 6) + " " + timestampS2);

}

}

}

}

private void Processing3()

{

while (true)

{

lock (data3)

{

while (data3.Count < 50)

{

Monitor.Wait(data3);

}

int i = 1;

while (i == 1)

{

RxPkt3[0] = data3.Dequeue();

if (RxPkt3[0] == 16)

{

RxPkt3[1] = data3.Dequeue();

if (RxPkt3[1] == 1)

{

int j = 1;

int k = 2;

while (j == 1)

{

RxPkt3[k] = data3.Dequeue();

if (RxPkt3[k] == 16)

{

RxPkt3[k + 1] = data3.Dequeue();

if (RxPkt3[k + 1] == 4)

{

j = 0;

i = 0;

}

}

k++;

}

}

}

}

byte[] convert = new byte[26];

for (i = 0; i < 26; i++)

{

convert[i] = RxPkt3[i];

}

Array.Reverse(convert);//operating system is little endians while the package is big endians, reverse the array.

if (kinect\_start != 0)

{

DateTime datenow = DateTime.Now;

int hour = datenow.Hour;

int minute = datenow.Minute;

int second = datenow.Second;

int millisecond = datenow.Millisecond;

int timestampS3 = hour \* 3600 \* 1000 + minute \* 60 \* 1000 + second \* 1000 + millisecond;

sw4.WriteLine(BitConverter.ToInt16(convert, 22) + " " + BitConverter.ToInt16(convert, 20) + " " + BitConverter.ToInt16(convert, 18) + " " + BitConverter.ToInt16(convert, 16) + " " + BitConverter.ToInt16(convert, 14) + " " + BitConverter.ToInt16(convert, 12) + " " + BitConverter.ToInt16(convert, 10) + " " + BitConverter.ToInt16(convert, 8) + " " + BitConverter.ToInt16(convert, 6) + " " + timestampS3);

}

}

}

}

private void Processing4()

{

while (true)

{

lock (data4)

{

while (data4.Count < 50)

{

Monitor.Wait(data4);

}

int i = 1;

while (i == 1)

{

RxPkt4[0] = data4.Dequeue();

if (RxPkt4[0] == 16)

{

RxPkt4[1] = data4.Dequeue();

if (RxPkt4[1] == 1)

{

int j = 1;

int k = 2;

while (j == 1)

{

RxPkt4[k] = data4.Dequeue();

if (RxPkt4[k] == 16)

{

RxPkt4[k + 1] = data4.Dequeue();

if (RxPkt4[k + 1] == 4)

{

j = 0;

i = 0;

}

}

k++;

}

}

}

}

byte[] convert = new byte[26];

for (i = 0; i < 26; i++)

{

convert[i] = RxPkt4[i];

}

Array.Reverse(convert);//operating system is little endians while the package is big endians, reverse the array.

if (kinect\_start != 0)

{

DateTime datenow = DateTime.Now;

int hour = datenow.Hour;

int minute = datenow.Minute;

int second = datenow.Second;

int millisecond = datenow.Millisecond;

int timestampS4 = hour \* 3600 \* 1000 + minute \* 60 \* 1000 + second \* 1000 + millisecond;

sw5.WriteLine(BitConverter.ToInt16(convert, 22) + " " + BitConverter.ToInt16(convert, 20) + " " + BitConverter.ToInt16(convert, 18) + " " + BitConverter.ToInt16(convert, 16) + " " + BitConverter.ToInt16(convert, 14) + " " + BitConverter.ToInt16(convert, 12) + " " + BitConverter.ToInt16(convert, 10) + " " + BitConverter.ToInt16(convert, 8) + " " + BitConverter.ToInt16(convert, 6) + " " + timestampS4);

}

}

}

}

private void Window\_KeyDown(object sender, System.Windows.Input.KeyEventArgs e)

{

if (keystart == 0 && e.Key == System.Windows.Input.Key.PageDown)

{

kinect\_start = 1;

button1.IsEnabled = false;

keystart = 1;

ButtonStop.IsEnabled = true;

}

else if (keystart == 1 && e.Key == System.Windows.Input.Key.PageDown)

{

kinect\_start = 0;

keystart = 0;

stop = 1;

spineBaseSW.Close();

spinemidSW.Close();

spineshoulderSW.Close();

sw2.Close();

sw3.Close();

sw4.Close();

sw5.Close();

fs\_kinect\_spinebase.Close();

fs\_kinect\_spinemid.Close();

fs\_kinect\_spineshoulder.Close();

fs\_kinect\_rightshoulder.Close();

fs\_sensor1.Close();

fs\_sensor2.Close();

fs\_sensor3.Close();

fs\_sensor4.Close();

ButtonStop.IsEnabled = false;

button1.IsEnabled = true;

}

}

private void button4\_Click(object sender, RoutedEventArgs e)

{

if (comboBox1.Text != "")

{

serialPort1.PortName = "COM" + comboBox1.Text;

serialPort1.BaudRate = 115200;

serialPort1.Parity = Parity.None;

serialPort1.StopBits = StopBits.One;

serialPort1.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler1);

serialPort1.Open();

}

if (comboBox2.Text != "")

{

serialPort2.PortName = "COM" + comboBox2.Text;

serialPort2.BaudRate = 115200;

serialPort2.Parity = Parity.None;

serialPort2.StopBits = StopBits.One;

serialPort2.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler2);

serialPort2.Open();

}

if (comboBox3.Text != "")

{

serialPort3.PortName = "COM" + comboBox3.Text;

serialPort3.BaudRate = 115200;

serialPort3.Parity = Parity.None;

serialPort3.StopBits = StopBits.One;

serialPort3.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler3);

serialPort3.Open();

}

if (comboBox4.Text != "")

{

serialPort4.PortName = "COM" + comboBox4.Text;

serialPort4.BaudRate = 115200;

serialPort4.Parity = Parity.None;

serialPort4.StopBits = StopBits.One;

serialPort4.DataReceived += new SerialDataReceivedEventHandler(DataReceivedHandler4);

serialPort4.Open();

}

button4.IsEnabled = false;

}

}

}