Upper Body Ground Truth Data Acquisition and Verification

EE180DA/B

Group Member: Ryan Lenart, Vincent Ju and Tianxiang Xu

Date: 12/4/2015

## Context

In the field of stroke patient rehabilitation and treatment, there is a potential demand for devices that can act as monitoring and guidance tools to facilitate the rehab process for stroke patients. The monitoring functionality helps doctors to be more productive in treating multiple patients and to know patients’ exact progress of recovery. The device should provide answers to the following queries to doctors. Have patients done the exercises assigned to them according to schedules? Were the exercises completed correctly? Do patients show signs of improvement? The device should also act as a guidance tool to help patients do exercises correctly while being able to encourage patients to be motivated in completing the exercises assigned by doctors. It should be able to display performance data after patients completed the exercises and give feedback to patients. Audio and visual contents and a performance score might be of great incentives for patients faced with immense task of relearning simple movements and performing simple daily tasks. Without encouragements, they will get easily frustrated and lose hope.

From the point of view of a device developer, the issue is how to get ground truth data while producing a model that can have generalization capabilities of various body movements. In the case of the upper limb activities, the issue is to obtain correct trajectory information from a combination of sensors and devices, from which, a model can then be developed that can correctly category different upper arm movements. In the special case of stroke patients, their upper limb activities will be non-typical when compared to those of a normal healthy person and would display individual characteristics depending on the extent and location of damages to their nerve systems. In this case, device developers might utilize techniques like wearing impairment tools that might restrict joints’ movements or finger movements in order to imitate a particular upper arm condition of a stroke patient. But ideally, it would be a great advantage if first hand data can be obtained directly from patients.

## Theory, Software, and Hardware

Tools that will be employed include various forms of hardware and software working in conjunction to provide an easy to use system that will allow caregivers to monitor patient rehabilitation. The hardware includes IMU sensors, a pressure sensitive mat, an Intel Edison board, and any personal computer. The software being employed includes Matlab for data processing, and python for connecting all parts of the system together. Two main Python frameworks will be used. The first being MATLAB Engine for Python, which allows a python script to call Matlab functions. The second being Web.py, which provides tools for building a web interface. As the project progresses, we would like to move away from Matlab for data processing and have Python do the same thing, as an actual caregiver is unlikely to have Matlab on their computers.

The primary goal is to obtain the correct trajectory information from motion sensors like an IMU sensor. An IMU sensor can detect, store and broadcast three axis acceleration, angular accelerations, from which the displacement and orientation data of the sensor can be extracted. In the case of the WHI sensors provided, the sensor data are preprocessed with onboard DSPs so that the output signal has less noise. The WHI sensors also provide quaternion data that can be used to facilitate trajectory calculations.

Raw sensor data are usually subject to noise and are oscillating instead of being steady. The tiny errors in the accelerometer readings could add up leading to huge errors in the final trajectory. To avoid sensor drift, zero velocity update are essential. The zero velocity update is a series of algorithms to zero acceleration data if velocity of a sensor is known to have stayed zero for a short period of time. In the case of movements that show characteristics of repetition, this technique can be very useful in eliminating accumulated errors from calculating trajectory by the quaternion methods whereby the gravity readings are often incompletely removed due to sensor noises and inaccuracies of readings. This demands the help of a sensor mat.

To collect upper arm activity data, two WHI sensors were used initially. One were placed around the wrist like a watch and the other either on the side of the upper arm or on the elbow. The reason to move the elbow is that the Kinect measurement tool has a designated sensor node fixed on the elbow. This way, later calculated trajectory of the elbow sensor and the wrist sensor can be compared with those obtained by the Kinect and an error information can be derived using the Kinect as the judge.

Matlab was used extensively for processing data and calculating trajectory information for the moment until good results can be obtained. The algorithms can be rewritten later in other languages or the same Matlab program can be called to perform calculations from other programs like Python. In that case there is no need for rewriting.

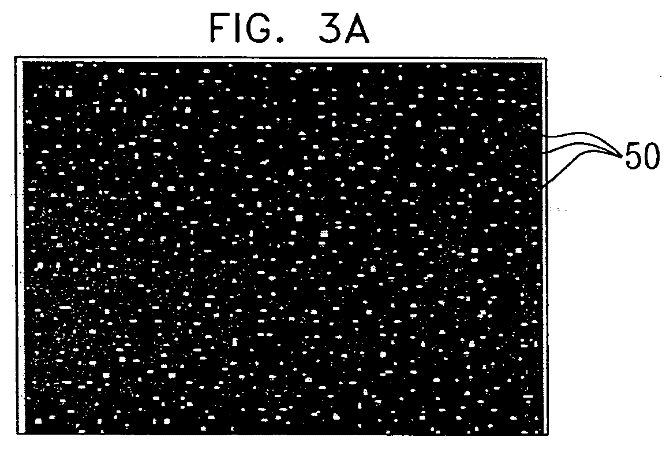
The way to calculate trajectory information is to first referencing all quaternions to an averaged starting value. Then rotate each and every acceleration value from sensor frame to the fixed initial reference frame. After this, an averaged gravity is subtracted from each and every acceleration data. The acceleration data now should be the correct data showing the actual acceleration of the sensor in the fixed initial reference frame.

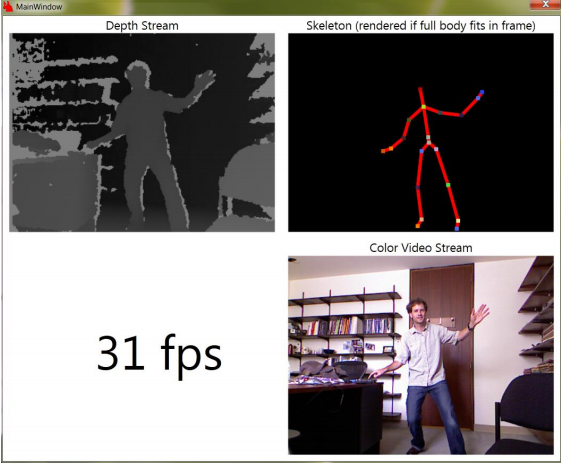
The pressure mat works primarily due to a material called velostat. Velostat is polymeric foil, infused with a form of paracrystalline carbon, which makes it conductive. It also has the property of changing resistance when pressure or deformations are applied to it. By having copper tape in between two layers of velostat placed on top of another layer of copper tape and a layer of velostat, you essentially create an open circuit when no pressure is applied, and a short circuit when pressure is applied. With one piece of copper tape soldered to a wire going to an analog input, and the other piece of tape soldered to a wire going to a power source, you create essentially a large button that would be easy for a patient to use.

In order to merge every part of the system together, we decided to use Python. Python is extremely versatile, as we can use it for data processing, task automation, as well as web interface design.

In addition to the pressure mat and the IMU sensors providing 2D coordinates (XY) and raw data of 3D acceleration data points to be integrated as 3D coordinates (XYZ), the Kinect Sensor, inevitably became our final judge and ultimate standard for calibration due to its ability of directly gathering the 3D coordinates (XYZ) of specific major joints of body and its satisfactory resolution.

###### 

*Figure:* Components of Kinect Sensor (v2)

The first step is gathering the depth image through Light Coding. As shown above, the Kinect Sensor includes IR emitters and a Depth Sensor. The IR emitters project speckle pattern of IR laser light forwards as shown in the right picture. In this way, each coordinate of the 2D frame forwarding (YZ) as labeled by a specific pattern which could be received and recognized by the Depth Sensor. Meanwhile, the Kinect Sensor maintains the original pattern map projected. By comparing the projected pattern map and the received pattern map, the Kinect Sensor can utilize the algorithm of Stereo Triangulation to calculate the distance between each projected IR laser point and the Kinect Sensor. The resolution of the Kinect Sensor is satisfactory – 3mm spatial and 1cm depth at the distance of 2m.

*Figure:* speckle pattern of IR laser light

The second step is recognizing joints from the depth image. This step became feasible through training the algorithm through machine learning in the Microsoft lab. Starting with 100,000 depth images with known skeleton form a motion capture system, the algorithm had learnt a *randomized decision forest* mapping depth images to major body joints, outputs the 3D coordinates (XYZ) of joint position in meters.

*Figure:* RGB, depth and body tracking

To get any data points from the Kinect Sensor, we need to use the Kinect for Windows SDK v2. In addition, in order to show both RGB images and 3D coordinate points of selected joints on the screen, using OpenCV 2.4.10 would be the simplest way. To use these APIs properly, we choose VS2013 as the developing environment.

## Basic Design

The basic design of the system is as follows. First, a patient outfitted with IMU sensors on their arm will touch a pressure mat at a known location. This will signal the Edison board that an exercise has begun and sets the initial location to where the mat is. The patient will perform a prescribed exercise. This could be anything from combing their hair to drinking from a cup. Once the exercise has been performed, the patient will again touch the pressure mat. This will signal the end of the exercise. The data from the time the mat was first touched to when the mat was touched again will be sent from the Edison to a caregiver's computer via WIFI. A python script will receive the data and call Matlab to process it, where it will then update a website showing the caregiver how the patient did their exercise. The website will store a log of the various exercises done so that they may be monitored for improvement.

Matlab was chosen initially for processing because we all had prior experience with it and we were able to find helpful examples written in Matlab. However, due to Matlab really only being popular in academic circles, we hope to move the processing to be done in Python to minimize the software needed by a caregiver. Also, having Python call the MATLAB Engine framework is a very slow process even for a function as simple as adding two numbers.

The pressure mat made from velostat was chosen because it could be made to have a high degree of precision and we could know the exact position in the X and Y plane every time the patient touched the mat. However we have since refined this to being a smaller mat with a known location. As long as the patient's touch the mat at the beginning and end of every exercise, drift in the sensors will be accounted for.

Python will be the overarching language that will automate the entire process. Python was chosen because of its versatility. We can use it to call Matlab for processing, or we could even do the processing natively. We can also use Web.py to take our processed data and update a web interface.

IMU Sensors

Edison Board

Laptop

MATLAB

Website

Python Script

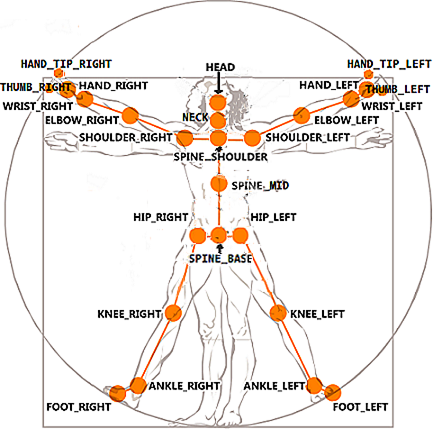
Python script calls Matlab functions.

Python script used to create and update website.

Python script receives data from Edison over WIFI.

Pressure Mat

*Diagram:* Basic project flow.

In the SDK, each joint was labeled as the JointType enumeration as shown in the figure below and table right.

The sample C++ code for acquiring data points from the Kinect Sensor is shown below:

|  |
| --- |
| for( int type = 0; type < JointType::JointType\_Count; type++ ){ |
| ColorSpacePoint colorSpacePoint = { 0 }; |
| pCoordinateMapper->MapCameraPointToColorSpace( joint[type].Position, &colorSpacePoint ); |
| int x = static\_cast<int>( colorSpacePoint.X ); |
| int y = static\_cast<int>( colorSpacePoint.Y ); |
| if( ( x >= 0 ) && ( x < width ) && ( y >= 0 ) && ( y < height ) ){ |
| cv::circle( bufferMat, cv::Point( x, y ), 5, static\_cast<cv::Scalar>( color[count] ), -1, CV\_AA );}} |

To record the movement of human skeleton, we need 5 variables per (frame-joint): # of frame (which would be upgraded to UTC timestamp in the next quarter), # of joint as enumerated in JointType, and 3D coordinates (XYZ) in meters gathered from the variable joint[type].Position. In addition, to record the joints we are interested in only, we can limit the variable type to the joints we specified. In the data points shown in the later section of Experimental Verification, we choose to track the movement of joint #9 ElbowRight and joint #10 WristRight.

## Problems Encountered and Workarounds:

The first problem we encountered was being able to map trajectory from the IMU sensors accurately. Working off of various examples, homework assignments, and help from our TA, we were able to develop a Matlab function that will plot the trajectory of an IMU sensor.

The next problem encountered was ways to synchronize the sensor data, the pressure mat data and Kinect data so that together they can be used to reference the same activity along the time axis. The way to work around this issue is to have data from pressure mat and Kinect carry with it the UTC time stamp which describe a unique point in time that has only one instance. Data with the same UTC stamp should correspond to the exact same moment in history and therefore must be the same data describing only one and same instance of the sensor characteristics.

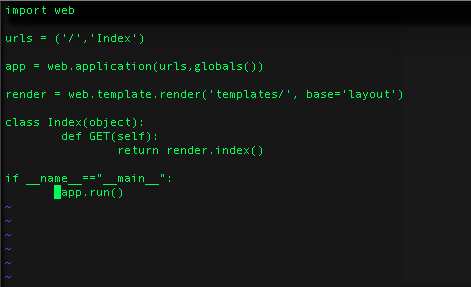
Another issue we ran into was that the pressure mat is extremely sensitive to any deformation in it. So the mat may conduct even though no one is touching it, due to wrinkles in the mat itself. The plan to overcome this, is to pin the mat flatly on some kind of rigid board, such as a corkboard. This would allow the board to be flat and not conduct when it shouldn’t be.

For a while we could not figure out how to connect every stage of the system together. We could get raw data from the IMU sensors, but then we would have to manually connect the sensors via Bluetooth to our computers and then manually start the functions in Matlab to process the data. We eventually settled on using Python. As stated before, Python can be used to automate all of these tasks. Python also allowed us to create our web interface.

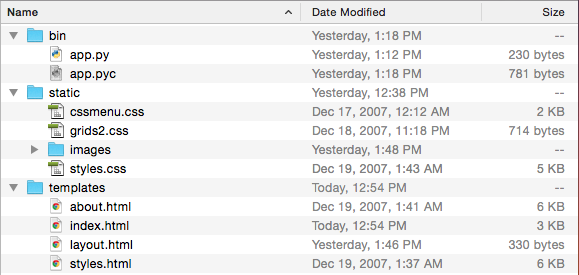
## Experimental Verification

We first wanted to test out Python for creating our website. Below is a screenshot of the basic site. At the moment it has no link to our Matlab processing script. Setting up a web application with Web.py was very easy and only required the following code.

The Web.py application references HTML and CSS to render the site with the following organization.

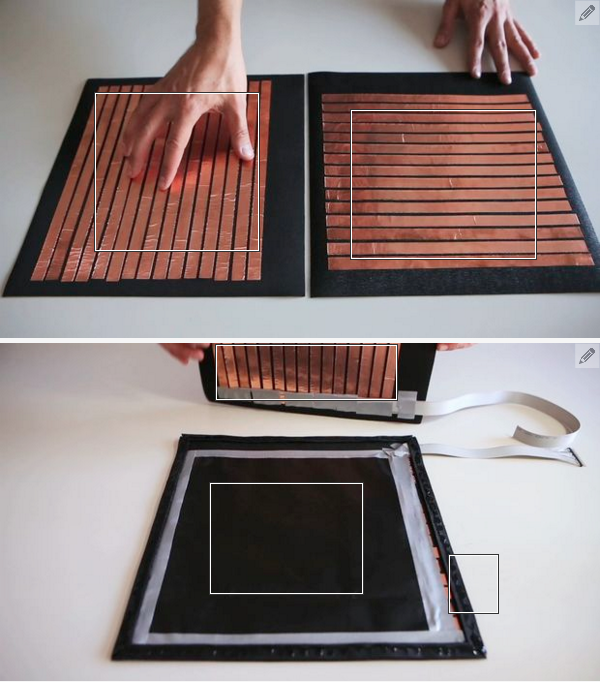
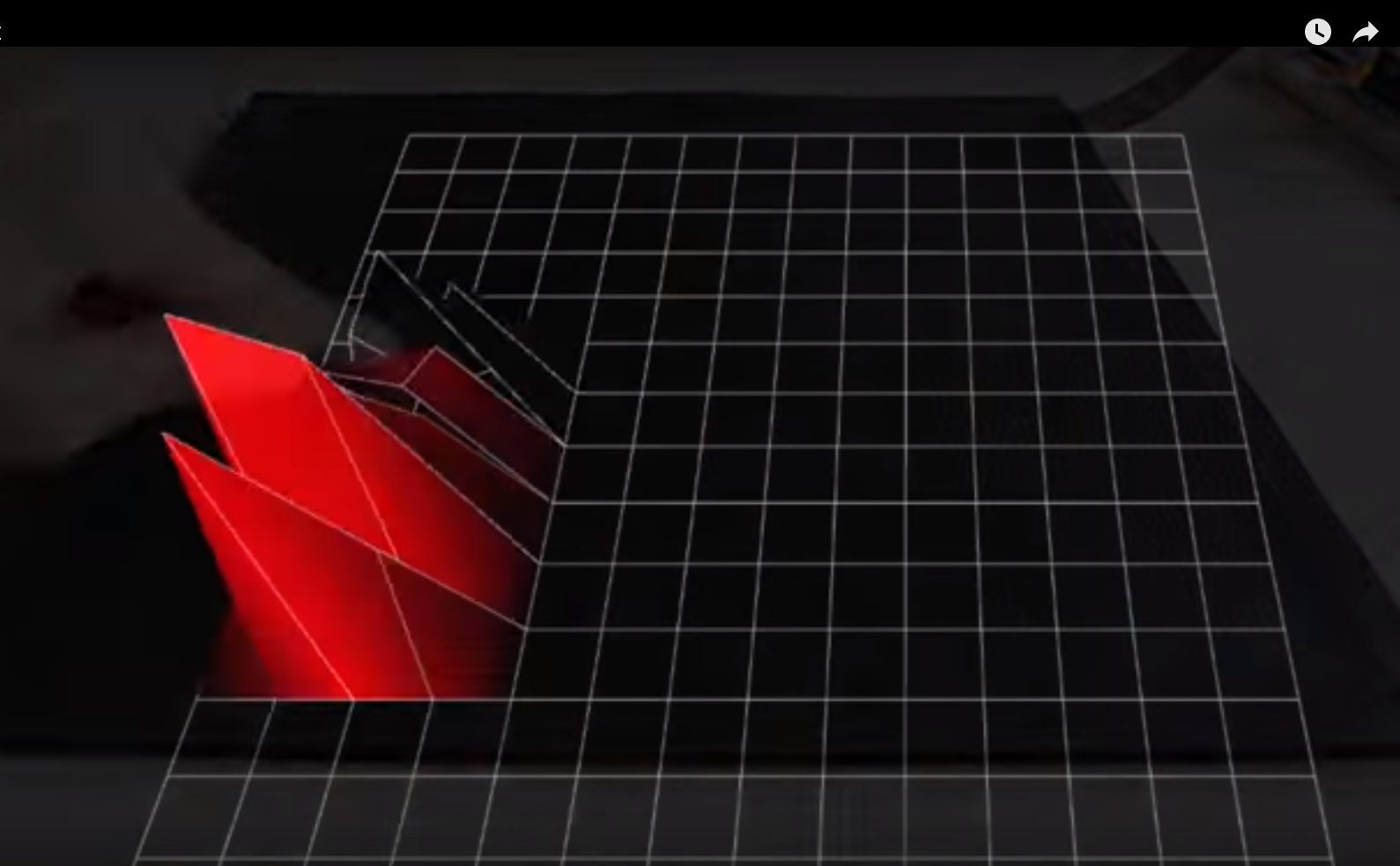


*Figure:* Code to start web a locally hosted web application.



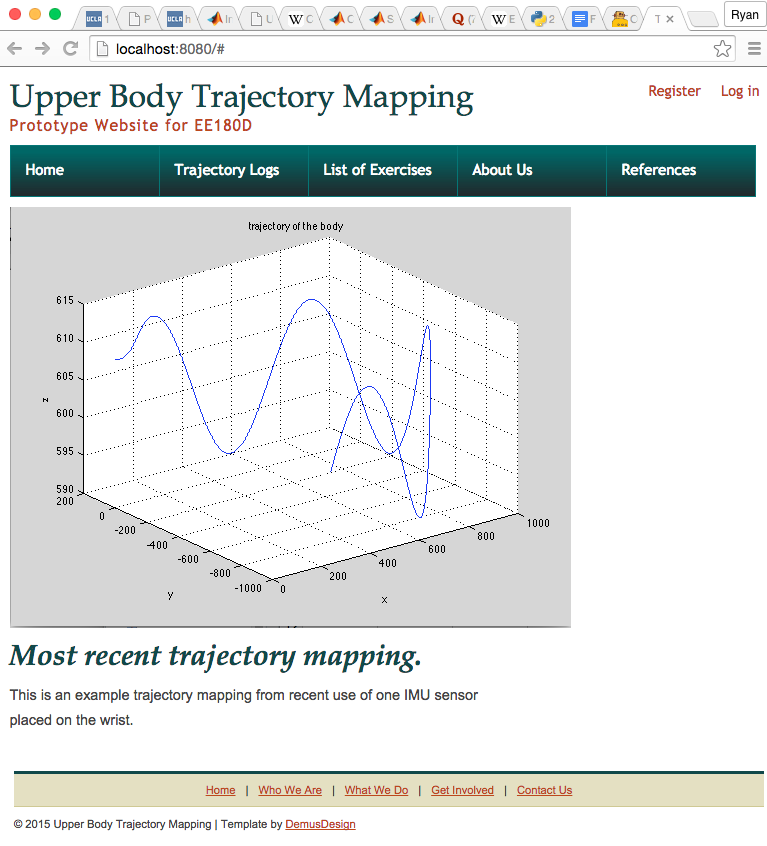
*Figure:* Web application file structure.

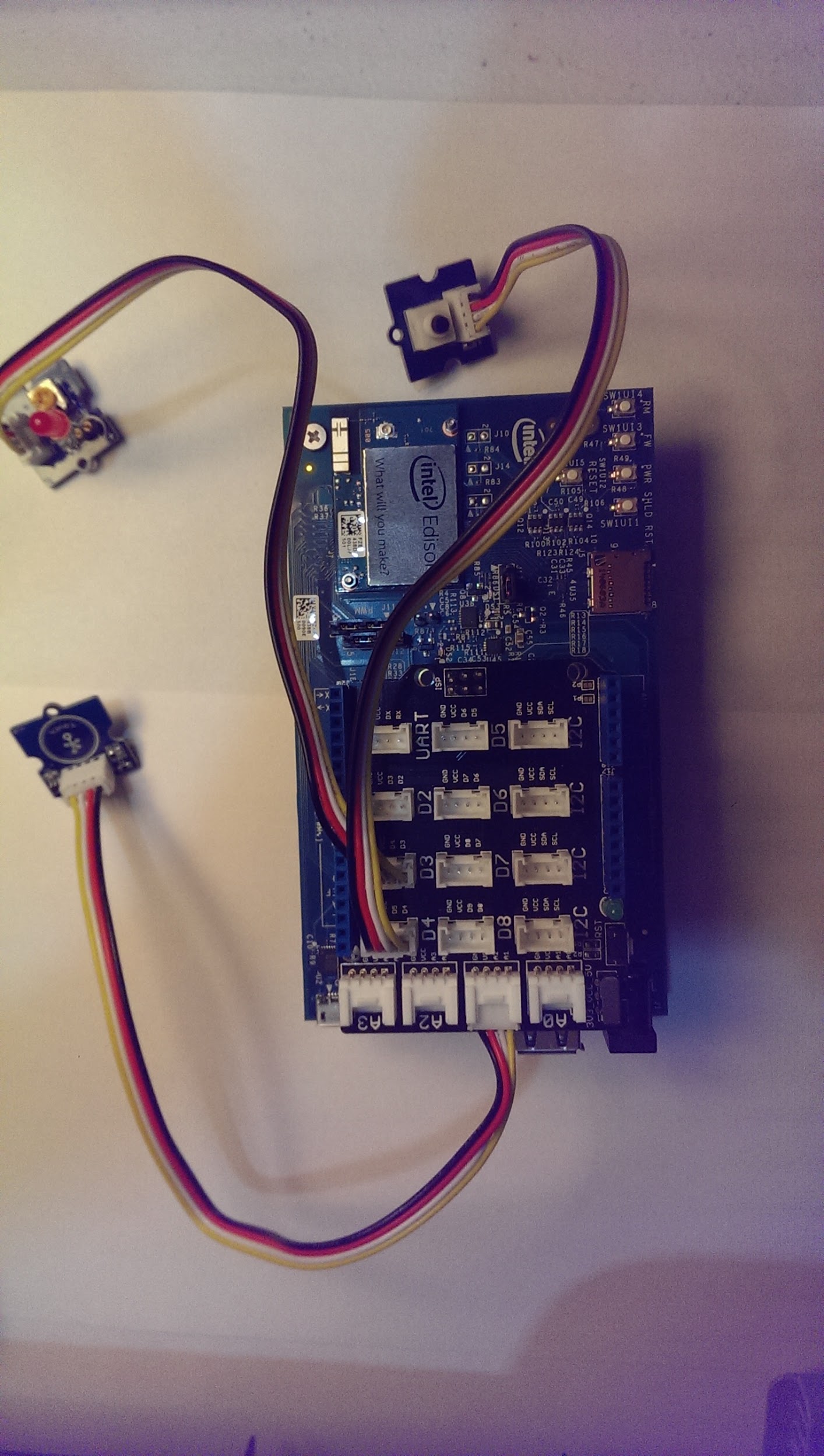
App.py is our main application and will control every step of the system as well as render the web interface. Web.py allows us to create a layout.html file which makes referencing CSS and image files easy to do.

The pressure mat was built initially without any sort of structural support. This allowed creases and divots to register unwanted signals in the mat output. The process for building the mat comes from an Instructables.com tutorial. Our pressure mat has currently been disassembled in order to make adjustments, but the basic structure can be seen in an image from the Instructions web page. A program provided by the authors of the Instuctables.com article allowed us to visualize the output of the mat. At the moment our mat is essentially unusable due to noise created from deformations.

*Figure:* Pressure mat

*Figure:* Data from the pressure mat

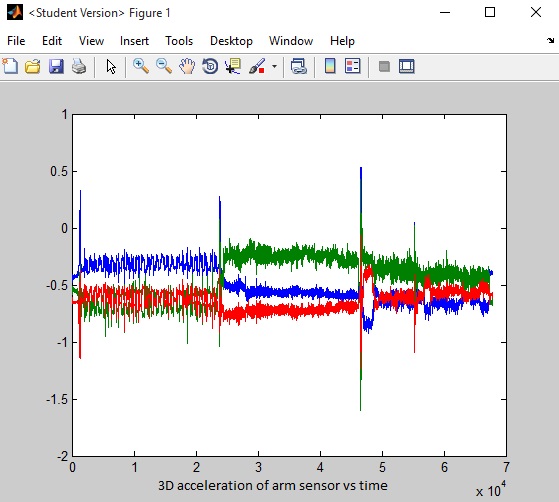
*Figure:* Basic Web app design.

Combining a touch sensor, a push button and a LED diode to build a device that could act as a touch indicator during actual ground truth data collection. If the touch sensor is touched the led indicator lights up. The process can be terminated by the push button.

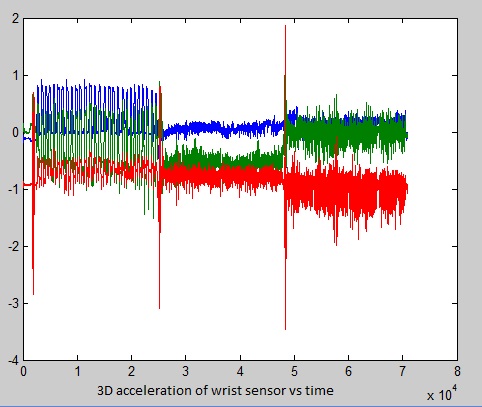
The device can be later further developed combining the Bluetooth function and processing power of Edison board. The wireless capability of the Edison board could be used to upload data onto designated website. The data can be analyzed by different methods.

First the Arduino codes were written and tested to be working. Then they were translated and made available on Edison board under Linux OS. This simple implementation can be later utilized to control pressure mat or some other functions yet to be defined.

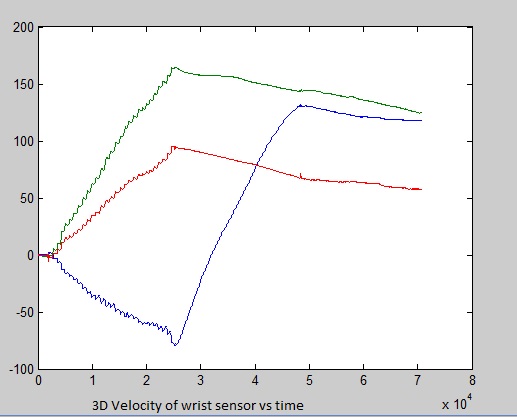
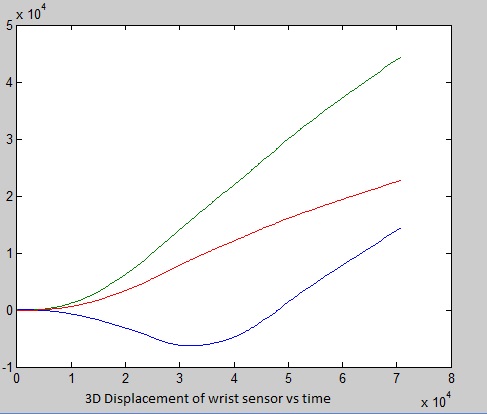
The six minute drinking, writing and typing experiment was performed to get familiar with working with WHI sensors and calculating trajectory information for upper limb. Matlab codes were written to calculate trajectory information of the two sensors.

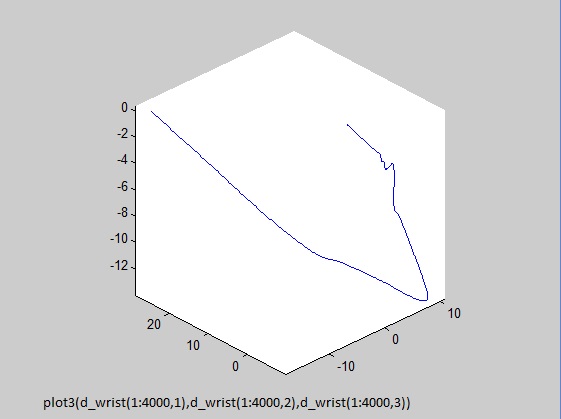
The results are illustrated in the following set of graphs. The sharp peaks on the upper left graph represent rapid hand waving as a way to signal the beginning of each activity. From the processed acceleration data it is not hard to see that three activities are captured as three distinctive segments.

The lower left graph is the processed acceleration data from the wrist WHI sensor which showed similar characteristics.

On the top of the next page, the upper left graph show the velocity vs time of the wrist sensor over the six minutes calculated from acceleration data. The upper right graph showed the trajectory of the wrist sensor over the six minutes time. The trajectory data are two large. Obviously, there are issues either in calculation or in the fact that the no ZUPT were used in the calculation of trajectory. It could also be because other experiment problems. The rapid hand shaking could have introduced some large errors. We are yet to look into the cause of this large error.

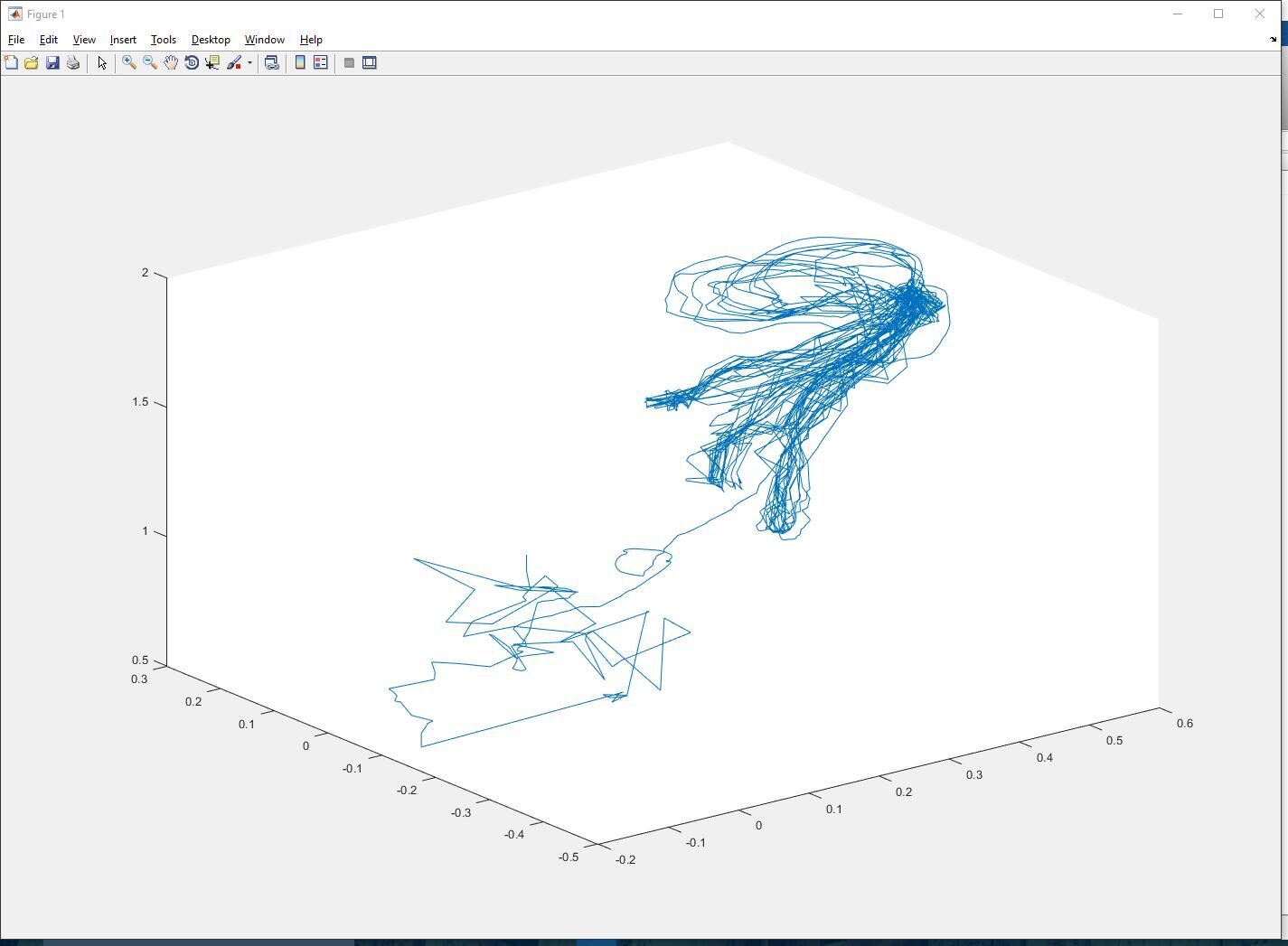
To break down things a little, we can look at what happened at the beginning and over a shorter period of time. The bottom 3D graph on the next page is a plot of the first 4000 data points which represents the first 20 seconds of trajectory at a sample rate 200 samples/sec.

From the trajectory diagram, it seemed that the wrist sensor started from zero on the upper right corner and went into a long and circular path. Obviously, there was some issues.

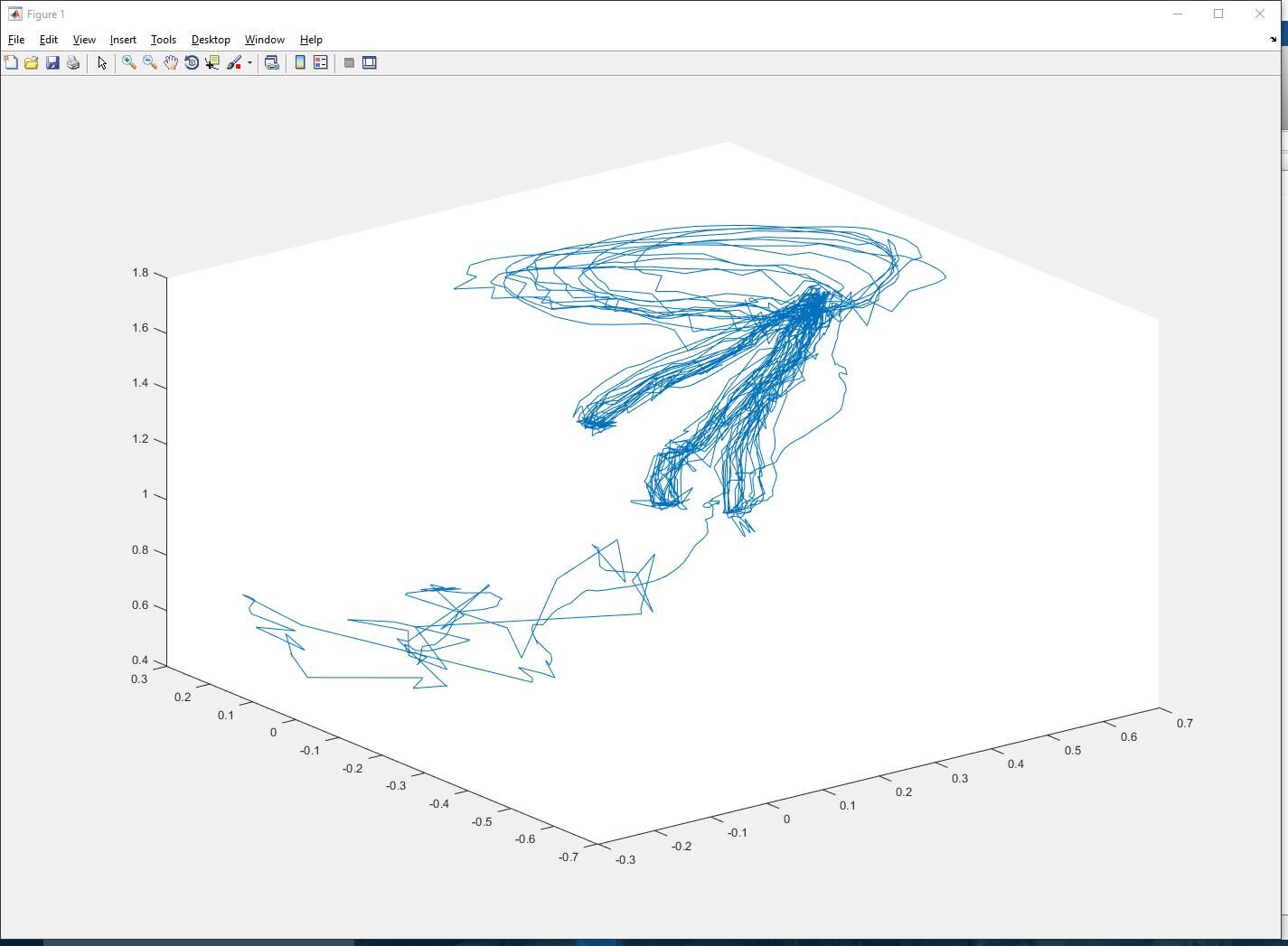


After the coding, gathering 3D coordinate (XYZ) data points from the Kinect Sensor would be straightforward. Sample trajectory as the model moving is shown below, with comparison with data points collected by IMU.

Using the plot3 function in Matlab: plot3(xyz(:, 1), xyz(:, 2), xyz(:, 3))



*Figure:* Movements recorded for joint #9 ElbowRight



*Figure:* Movements recorded for joint #10 WristRight.

## Future plan

The Matlab program needs to be fully developed to correctly reproduce trajectory information first. Then the window methods can be used to search for a particular type of trajectory to differentiate different types of activity. The number of repetitions for each activity can be calculated and so do the duration of each activity.

Then comes the part to combine WHI sensors, pressure mat to collect ground truth data and use pressure mat data to implement ZUPT to get a better trajectory information for ground truth. Then tests can be performed having a subject wearing just one wrist sensor and do some designated sequence of activity in combination with the sensor mat. The data can then be processed by an application and result can be compared to the actual experiment to see how well the prediction is, i.e. what activity and how many repetitions.

The pressure mat needs to be refined in order to get a clear reading with noise created from deformations in the mat. We may look into simplifying this stage of the system even further, being as we only need the pressure mat as a sort of button.

Much of next quarter will be devoted to further developing the overarching Python application. This means integrating our Matlab scripts with the web app and further refining and developing the web interface.

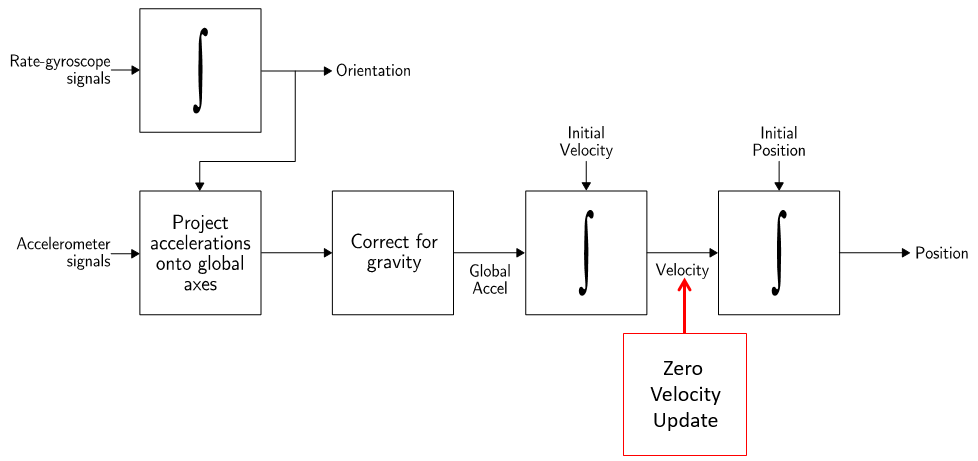
Calibration for the IMU sensor is the central task of the entire project. As the data point of each timestamp gathered by the IMU sensor has an error range provided by product specification, each data point we gathered is not a point, but a range of coordinates. By calibration, we can derive the algorithm learning which point within the error range is acceptable, and hopefully our product could achieve a commercial quality.

In next quarter, we planned to caliber the IMU sensor in two ways: using constraints, using absolute coordinates gathered by the Kinect Sensor and using zero velocity update.

Using constraints would be the most self-evident method for calibration. At first the data points collected by our IMU sensor is totally senseless. My shoulder is in the fourth floor, while my wrist is in the seventh floor. The length of my arm is a constant (with an error range indeed) and can become an absolute constraint for the 3D coordinates we collected from sensors.

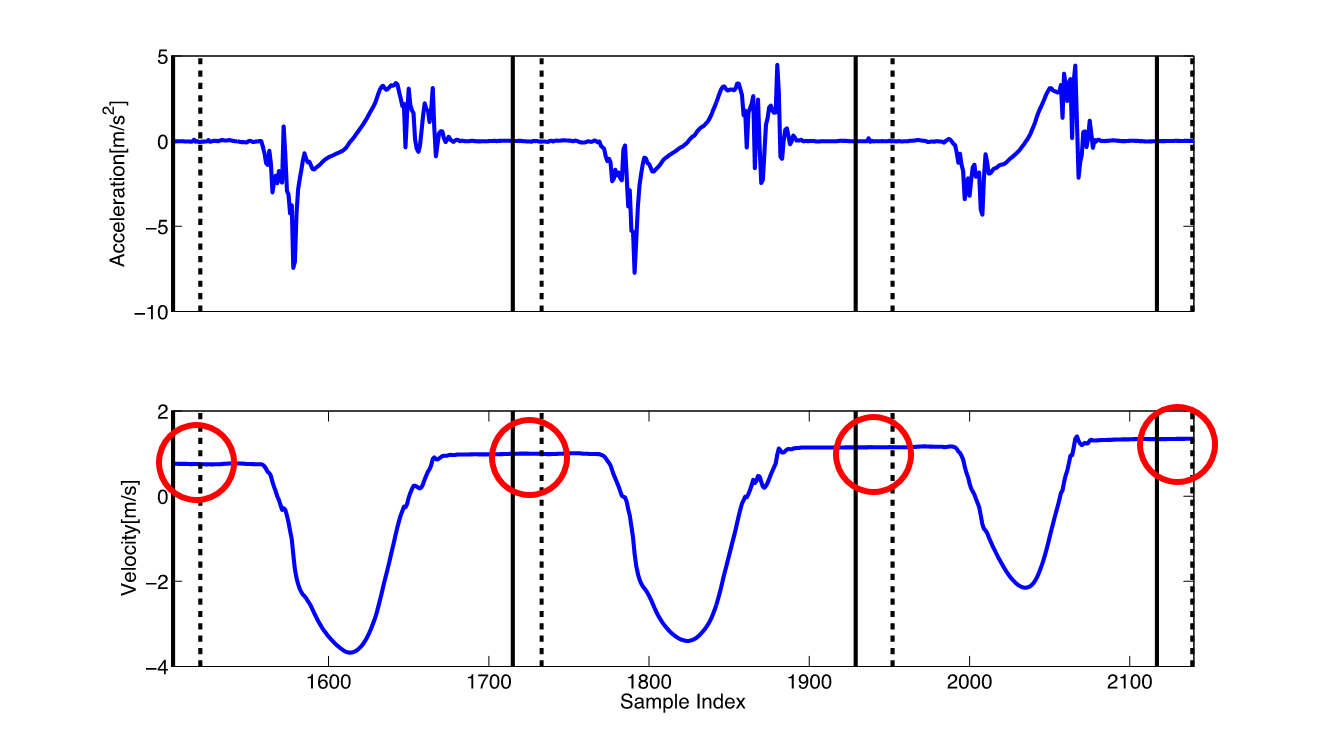
The Kinect Sensor, directly gathering position data with a resolution of centimeters, is the ultimate judge of the accuracy of the pressure mat - IMU sensor combination in trajectory calculations to yield a performance parameter of ground truth data acquisition. However, to synchronize the data points gathered by different sensors, the Kinect Sensor is required to provide data points with a UTC time stamp.

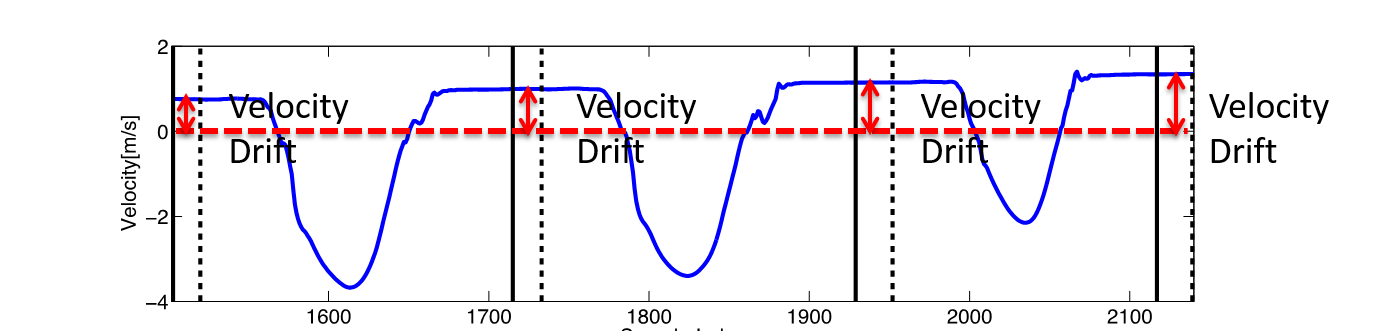
Furthermore, we could utilize the absolute coordinates acquired by the Kinect Sensor to calibrate the IMU sensor. We need to derive an algorithm for the IMU sensor learning from the absolute coordinates acquired by the Kinect Sensor, to train our sensor combination to become commercial quality. Deriving this algorithm could be the hardest part in next quarter, but it could be the most successful method of calibration.



*Diagram:* The additional step of Zero Velocity Update in integrating the coordinates

The third calibration method we are going to use is Zero Velocity Update. The IMU sensors are with errors constantly due to constant bias, temperature noise, etc. However, we can set the velocity to zero when foot is stationary, and smooth the velocity in between before being integrated for position. We need to detect whether it is at Zero Velocity by threshold the motion acceleration energy calculated from gathered data points from accelerometers and gyroscopes.





*Figure:* The projected results of the method of Zero Velocity

## References:

[1] Mathworks Documentation for MATLAB Engine for Python. [Online]. Available: <http://www.mathworks.com/help/matlab/matlab_external/call-matlab-functions-from-python.html>

[2] Install MATLAB Engine for Python. [Online] Availiable: <http://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html#bujdbfi>

[3] O-Mat instuctions. [Online] Availiable: <http://www.instructables.com/id/O-mat/?ALLSTEPS>

[4] Web.py tutorial [Online] Availiable: <http://webpy.org/docs/0.3/tutorial>

[5] Learn with MATLAB and Simulink Tutorial [Online] available: <http://www.mathworks.com/support/learn-with-matlab-tutorials.html>

[6] Introduction for Kinect for Windows SDK v2 [Online, Traditional Chinese] available: <https://kheresy.wordpress.com/2014/12/29/kinect-for-windows-sdk-v2-basic/>

[7] Depth-varying light fields for three dimensional sensing [Online, US Patent 2008/0106746 A1] available: <http://www.google.com/patents/US20080106746>

[8] How does the Kinect work? [Online] available: <http://users.dickinson.edu/~jmac/selected-talks/kinect.pdf>

[9] Reference, Kinect for Windows SDK [Online] available: <https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx>

[10] Zero Velocity Update Tutorial Slides provided by our TA.