Eric Wu and Jeremy Kanovsky
ME-18, Final Project Report
Professor Felix Huang
May 1, 2020

Absolute Translational Position Sensing using Inertial Measurement Units

Introduction

Inertial measurement unit (IMU) sensors are widely used in a variety of applications, primarily due to their relatively low cost and availability. They are an inexpensive way to add gyroscopic and acceleration data collection to consumer devices, and research apparatuses. Most IMU devices are multi-axis, meaning they are capable of measuring several different metrics simultaneously, often in different dimensions. A three degree of freedom (DOF) IMU might measure acceleration in the x-, y-, and x-axis, or potentially gyroscopic motion in the same. A six degree of freedom IMU may measure both, or replace one or the other with a magnetometer. Higher DOF units are available, but less commonly used.

Unfortunately, the low cost of IMU devices often comes with drawbacks. Because the chips inside them are mass produced to be as cost-efficient as possible, they usually suffer from inaccuracies or drift. Drift is when the sensor begins to accumulate erroneous data points, even when stationary. This can come from a poorly calibrated sensor, or one that has been in use for an extended period of time (due to the nature of drift to accumulate almost exponentially).

Because of this particular issue, IMU data is less reliable than that obtained from an absolute position or absolute orientation sensor. Inertial measurement units generate data with inherent noise and drift, making them difficult to work with for precision applications, or even applications doing more than basic computation with the data.

Our research project is to investigate the viability of using raw data obtained from a low-cost IMU device to keep track of absolute translational position in an unknown reference frame. To this end, we will be collecting large amounts of data from an IMU, and attempting to use it to track the path of an IMU in space without any external sensors or input devices. The IMU will be totally isolated from its environment in that it will be unable to know where it is in space relative to other objects. Similar work has been done on this topic for orientation sensing using a gyroscope, but there has been little to no success finding translation using an accelerometer.

Background and Related Work

As mentioned above, some work has been done to find and track absolute orientation of a device using the gyroscope in an IMU. A study done by A.R. Jimenez and his colleges implemented dead-reckoning algorithms with an IMU to measure human strides[1]. This work ended with good results over a long walking distance, but unfortunately was not expanded upon to other applications. The algorithms developed and implemented were dependent on the assumption that the IMU was measuring a human walking stride, and therefore factored in estimations of stride length, and a fairly regular gait.

[1] A. R. Jimenez, F. Seco, C. Prieto and J. Guevara, "A comparison of Pedestrian Dead-Reckoning algorithms using a low-cost MEMS IMU," *2009 IEEE International Symposium on Intelligent Signal Processing*, Budapest, 2009, pp. 37-42.

Other work on the topic includes Farhad Aghilis paper on using two IMU sensors to determine and track attitude and position of mobile robots[2]. This work falls closer to the goals of our own, but relies on global positioning systems (GPS) to reduce drift in the model. The Kalman filter developed in this paper relied on input from the GPS as part of the state estimation input matrix.

Finally, work done by ETH Zurich combined monocular vision and the popular robotics pose estimation algorithm SLAM to estimate the absolute position with an IMU[3]. As with the other approaches mentioned, this relies heavily on other sensors. While the authors find sensor fusion between a camera and an IMU is extremely robust, it still relies on the robot having a sensor of where it is according to a visual reference frame. For systems that do not support a camera, or that wish to operate in conditions unsuitable to vision (such as in total darkness or low light conditions), it would still be largely beneficial to use only input from an IMU for translational pose estimation.

Experimental Setup

The IMU we chose for this experiment and our data collection was a MPU-6050 6-degree of freedom accelerometer and gyroscope. While originally made by InvenSense[4], we bought this sensor on a breakout board supplied by Adafruit[5]. This allowed us to leverage the STEMMA QT connector (developed by SparkFun) and communicate easily using an I^{2}C serial bus.

We chose to use a Raspberry Pi Zero microprocessor with Wi-Fi to power our IMU and provide data logging capabilities. This allowed us to write large amounts of data directly into a comma separated values (CSV) file. The benefit of the Raspberry Pi is that we could quickly and easily change the configuration of the data we chose to collect. Initially we logged all six axes of data produced from our IMU. However, we quickly realized that clock speed limitations meant we would only be capable of collecting data at approximately 55 Hz. By eliminating the gyroscope data in our logs, we were able to increase our sampling rate to approximately 100 Hz. We also chose to include a timestamp of each data point for purposes of data processing (discussed later).

The IMU and Raspberry Pi boards were mounted on a 3D printed fixture (shown below). In this fixture we also included a lithium polymer battery that would power both boards. This allowed us to create a small, and wire-free device for measuring acceleration. The fixture could be mounted on a robotic car, or held in hand and moved freely through space. The wireless nature of the Raspberry Pi allowed us to execute data collection programs and retrieve the data without requiring a wired connection to the board. All data was uploaded directly to GitHub for processing[6].

[2] F. Aghili and A. Salerno, "Driftless 3-D Attitude Determination and Positioning of Mobile Robots By Integration of IMU With Two RTK GPSs," in *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 1, pp. 21-31, Feb. 2013.
[3] Nützi, G., Weiss, S., Scaramuzza, D. *et al.* Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM. *J Intell Robot Syst* 61, 287–299 (2011). https://doi.org/10.1007/s10846-010-9490-z
[4] https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
[5] https://www.adafruit.com/product/3886
[6] https://github.com/0xJeremy/ME-18-Final/tree/master/data

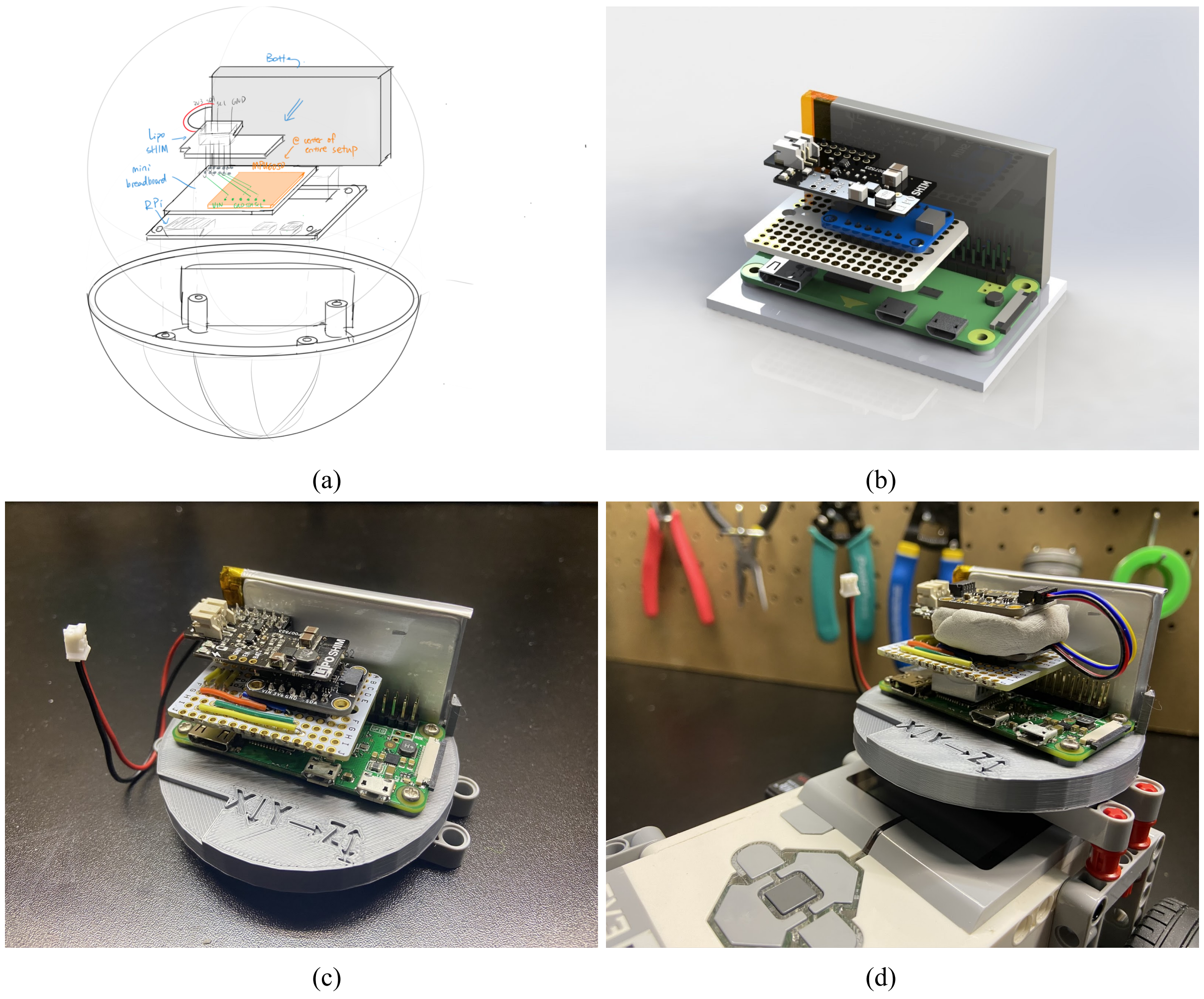


Figure 1 a. Early concept sketch of the sensor system; b. Sensor system designed in SolidWorks; c. Sensor system with LEGO Technic compatibility; d. Complete experimental setup with two MPU 6050 connected.

<u>Data Collection Methods</u>

The data collection code was written in python based on the existing MPU 6050 library provided by Adafruit industries. The completed code can be found in the Github repository under the "data_logging" folder.

In order to simplify the data behavior and avoid introducing unnecessary noise during the experiment, we decided to limit the movement of the set up to 1D translational movement. With the materials we have on hand in my house, we use a LEGO MINDSTORMS EV3 kit to construct an unpowered car with no ability to steer, simulating an environment with a linear slide rail.

The experiment is done by placing the apparatus on a clean, flat surface. A tape measure is secured on the surface next to the car along the direction of movement. For each trial, the Raspberry Pi would record the raw acceleration and gyroscopic reading for 10 seconds. The program automatically writes the data to a file in CSV format. Finally, all recorded data can be pushed to the Github repository for later access and manipulation. A picture of the experimental setup is shown below as Figure 2.

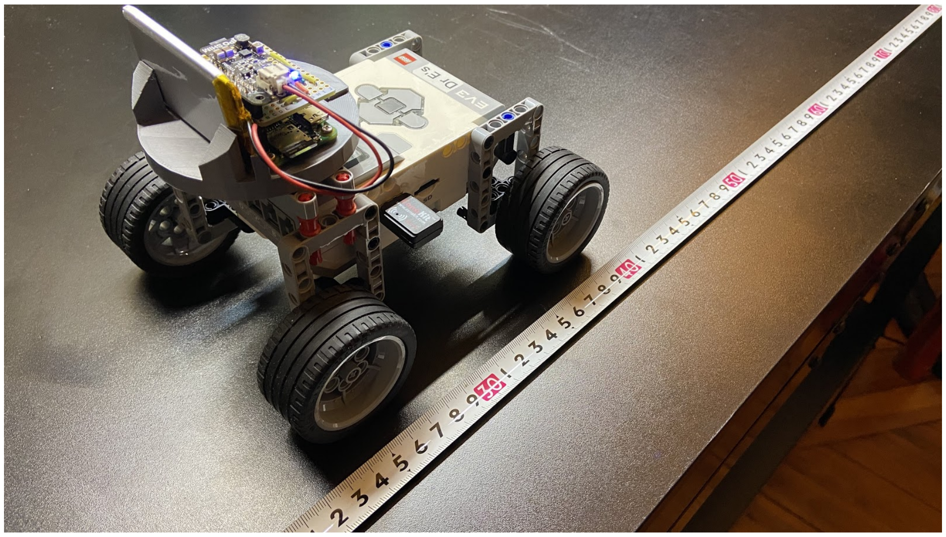


Figure 2: Experimental setup

Over the course of the project, we performed multiple sets of trials with different movements. The first set of data is performed by moving the car to a specified position and then back to the origin. The second set of data is performed by moving the car to a set of distinct locations. The third set of data is performed by repeating similar movements as the second set of trials with another MPU6050 sensor added to the apparatus.

<u>Data Pre-Processing Techniques</u>

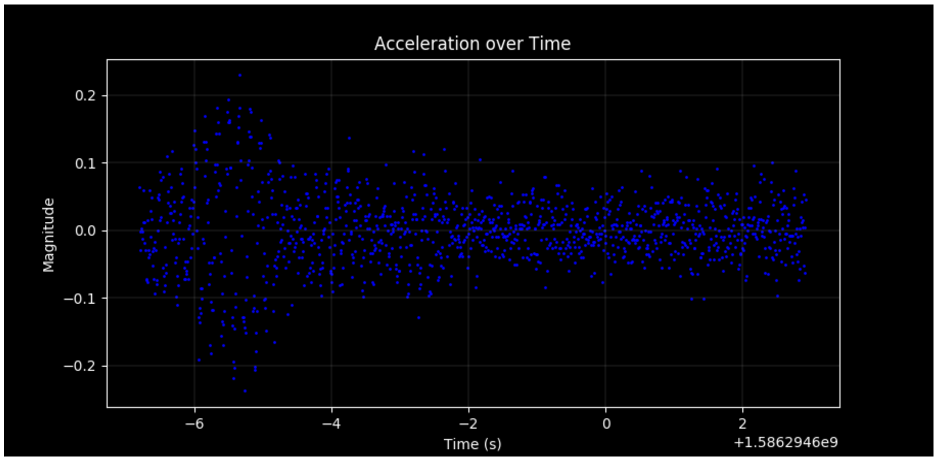
A large portion of the success of our project predicates on having meaningful data to use for analysis. Because low cost IMU devices are notorious for having sporadic data points and are susceptible to drift, we needed robust data processing techniques that would help us smooth and process the data before we began working on deriving the absolute translational position. To this end, we experimented with a number of well known filtering techniques. Figure 3(a) shows the raw data set we used to evaluate the metrics of each filter.

The first technique we applied was a convolution to the raw data. We choose to use a convolution of variable size with a total area of 1.0. By varying the size of our function window, we were able to define a metric for how much we wanted the data smoothed. The smaller the convolution window the less smoothing would occur. By defining a window of one, we essentially applied no smoothing to the data. Figure 3(b) shows the results of this technique, with window sizes of five, ten, and 100.

The next filter implementation we developed was the Savitzky-Golay filter. This is a digital filter designed to increase precision without distorting the original signal. It utilized convolutional filtering and least squares on subsets of the data to give the final result a smoothed signal. By defining the order of the polynomial used to fit the data, we were able to also define the “smoothing coefficient” so to speak of this filter. Figure 3(c) shows the results of such technique using polynomials of fifth, 11th, and 101th order.

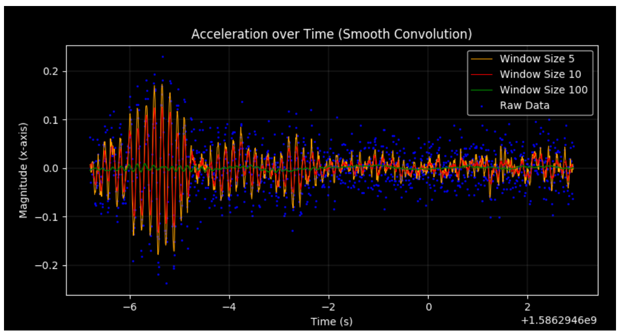
We then developed a rolling window filter. Using the rolling window function from the Pandas data processing library, we experimented with windows of different sizes. Figure 3(d) shows the results of this technique using window sizes of one, ten, and 100.

The last filter technique we implemented was a forward-background digital filter using second-order sections. This filter applies a linear digital filter twice over the data, once in the forward direction and once in the reverse direction. We began by creating a lowpass Butterworth filter with different orders and critical frequencies. The Butterworth filter was then applied over the data. We used the filter order and cutoff frequency of the Butterworth filter as our metric for how much the data would be filtered. Figure 3(e) shows the results of this technique with a constant third order fit, and cutoff frequencies of 0.75, 0.25, and 0.1.

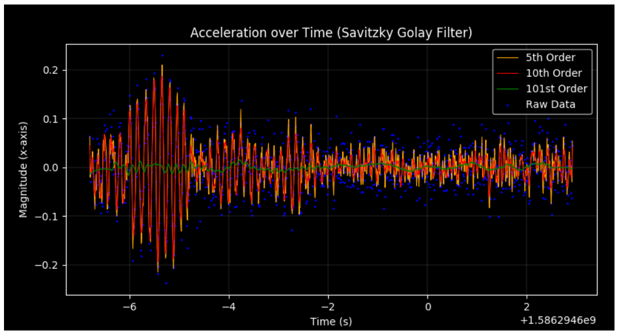


(a)

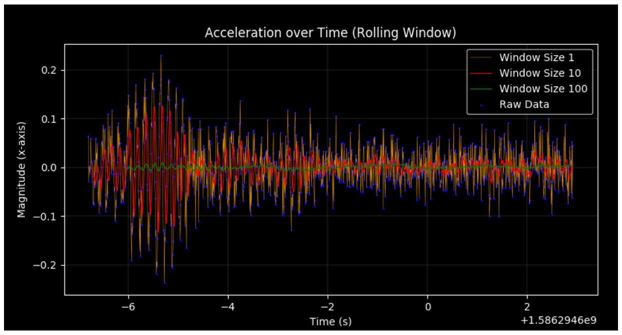
Figure 3
Data pre-processing techniques used in this project: a. Raw data used for filter evaluation; b. Convolution Filter; c. Savitzky-Golay filter; d. The rolling window function; e. Double digital filter



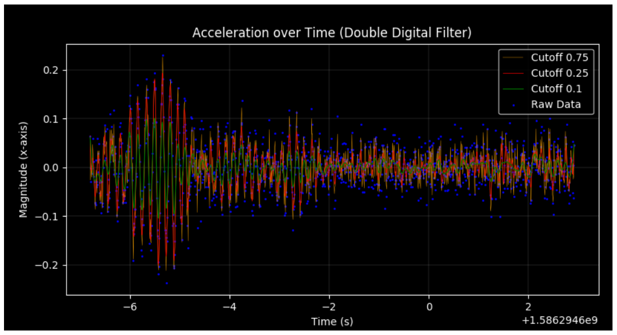
(b)



(c)



(d)



(e)

As might be apparent in the charts for each technique, they are remarkably similar and can generate much of the same results depending on the input filter parameters. It is for this reason we believe that the filtering technique is less important to the overall experiment than it's existence. The data must be filtered before use due to the high variance observed, but this filter is of less importance than the post-processing techniques described below. Implementations of these data filtering techniques can be found on the project's GitHub page[7].

Data Post-Processing Techniques

Much of our data post-processing techniques relied on trial and error. Wr quickly found that a large problem with our data was drift in the data. When we performed time integrations, the calculated velocity of the sensor would accumulate error exponentially. Zeroing all the data according to the mean of the entire dataset helped linearize the error, but it was still problematic when trying to draw conclusions. We found that one of the most successful methods for eliminating this drift was to perform the integration in pieces. This would allow the error to reset every time we split the data, and would allow us to choose arbitrary break points where we observed drift becoming a major problem. Below is a plot showing our original integration based velocity, and then our transformed data after segmentation.

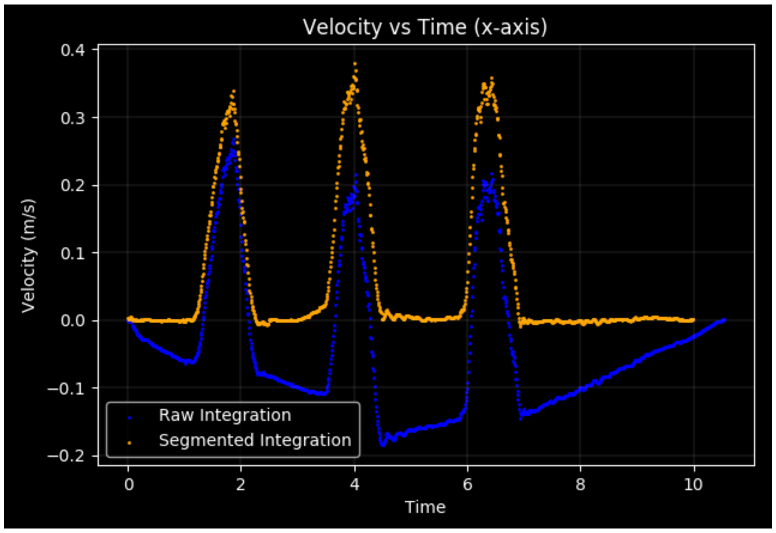


Figure 4: Post processing the data using segmentation

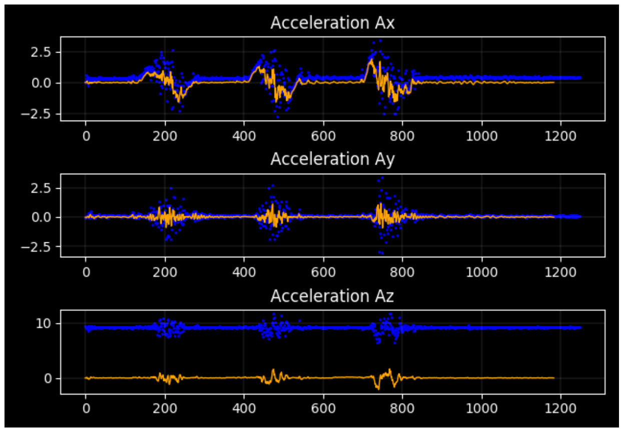
This method makes it fairly obvious that drift is a relatively major problem. With segmented integration, however, we are able to perform another direct integration to get the position. Overall, the following steps are performed on the data from start to finish:

1. Data is loaded into a Python dictionary, and the metadata is retrieved from the filename (which stores the travel distance of the IMU).
2. A time differential is added as an additional column to the data. Then, all the original timestamps are zeroed to begin at the start of the trial period (instead of timestamping from the beginning of the Unix Epoch).

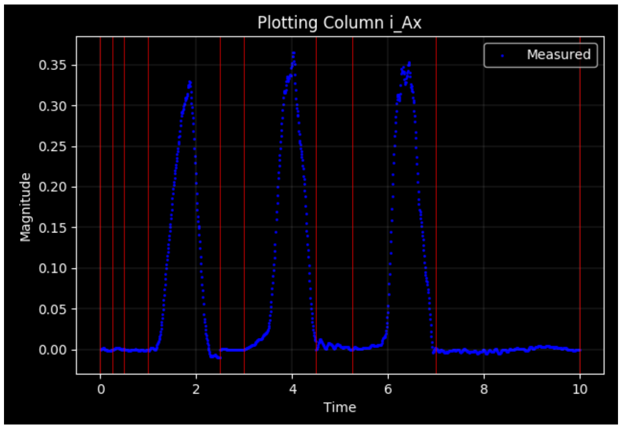
[7] https://github.com/0xJeremy/ME-18-Final/tree/master/pre_processing

3. The data is sliced into various sections, by time, where drift is present. This includes sections containing a spike of data where the IMU was moved, and flat sections where there was no movement, but major drift is present in the data. For each section of data we:
    a. Remove all bias from the data by calculating the mean of the segment, and removing that from all data points.
    b. Apply a smoothing technique (outlined in the pre-processing section). For most datasets, we find a convolution or double digital filter yields the best results. The convolution window size or digital filter order and cutoff can be modified per dataset for the best results.
    c. Perform a time-based double integration of the data to yield velocity and position columns.
4. We then zip all the data slices back into a single contiguous whole. We offset each slice by the magnitude of the last item in the previous slice. In this manner, accumulations in velocity are removed, but accumulations in position still exist.

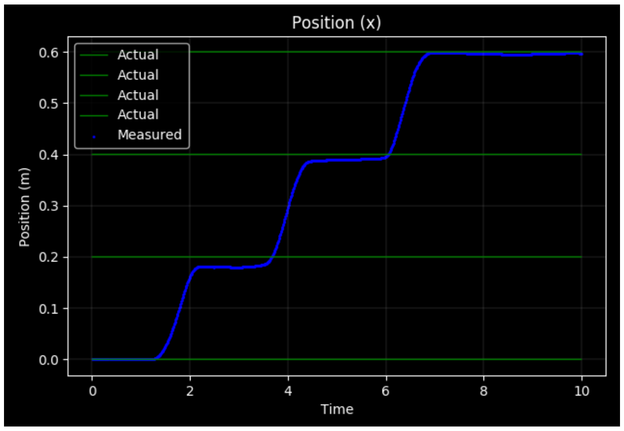
Below are some of our results from applying this method. Figure 5(a) is the raw data from the trials, with the smoothing filter applied over it. Figure 5(b) is the calculated velocity at each point, with the red lines indicating where the data was cut. Figure 5(c) is the calculated position of the IMU over time, with the green lines indicating the points along the axis the unit was moved.



(a)



(b)



(c)

Figure 5
Graphs of final results after post-processing: a. Raw data from trials with smoothing and filter applied; b. Calculated velocity at each point; c. Calculated position over time form the IMU.

Results

Overall this experiment was reasonably successful in terms of meeting its goals. We were able to calculate, with millimeter level accuracy, the position of the IMU as it moved across our predefined track. Unfortunately, this method relies on manual tuning of the time breakpoints by hand. This is non-optimal because it requires different breakpoints per dataset, and therefore this system cannot be applied in real time. Further work will focus on peak detection of a system, and detection of when the sensor begins to accumulate exponential drift. While methods exist to compute both of these automatically, it begins to exit the scope of this project to continue further work into that area. We have also found that our system is not infallible to all the datasets we collected. The charts above are from a particular subset of our data in which the IMU was moved at speed across the table. When the movement is slower, the data becomes exponentially noiser, and more difficult to parse and process accurately. At times, the IMU readings appear meaningless. This is likely due to our sampling rate and low-cost sensor, however, and not our processing techniques.

Moving forward, there are several areas of research that could stem from these experiments. The first would be to apply our existing system in real-time. This may prove difficult because much of our analysis relies on being able to apply transforms over the entire dataset. Another would be peak detection in real-time, and the ability to calculate inherent drift in the sensor. Perhaps finding ways to reset the sensor in real-time would also be a useful avenue to investigate. Finally, expanding our system to one that functions across a wider range of movements, and in more axis than outs. We confined our experiments to a single dimension in most cases due to the difficulty that poses already.

Conclusion

Inertial measurement units (IMUs) are extremely robust in a wide range of applications. We investigated how they might also be used as absolute translational position sensors. During the course of our investigation and experiments, we found that with some information about the ground truth state of movement they can supply reasonably accurate information about translation. This data, however, is only useful after the fact when processed with our methods. Further work might be performed to improve the usability of this system in real-time, and expand on our findings. The implementations of all our algorithms and data processing tools, and the raw data from our trials can be found here: https://github.com/0xJeremy/ME-18-Final.

References

A. R. Jimenez, F. Seco, C. Prieto and J. Guevara, "A comparison of Pedestrian Dead-Reckoning algorithms using a low-cost MEMS IMU," *2009 IEEE International Symposium on Intelligent Signal Processing*, Budapest, 2009, pp. 37-42.

F. Aghili and A. Salerno, "Driftless 3-D Attitude Determination and Positioning of Mobile Robots By Integration of IMU With Two RTK GPSs," in *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 1, pp. 21-31, Feb. 2013.

Nützi, G., Weiss, S., Scaramuzza, D. *et al.* Fusion of IMU and Vision for Absolute Scale Estimation in Monocular SLAM. *J Intell Robot Syst* 61, 287–299 (2011). https://doi.org/10.1007/s10846-010-9490-z

Appendix A: Updated Project Proposal and Milestones

Project Proposal:

Objective: Investigate the possibility to turn a 6 DOF Inertial Measurement Unit into an absolute positional sensor using various data processing methods.

Abstract: Inertial Measurement Unit (IMU) sensors are widely used in various applications. Some major challenges in working with a low-cost IMU sensor is noise and drifts in data due to the sensor itself, and the difficulty to deduce orientation and positional information from the on-board accelerometer and gyroscopes. In the past, people have been using sensor fusion algorithms to get clean absolute orientation data from IMU sensors. For our project, we are interested in investigating the feasibility of getting absolute x, y, and z positional data from a low-cost IMU.

Our data collection method will involve setting up a battery-powered Raspberry Pi that communicates with an MPU6050 IMU sensor, with the intention to create a cable-free environment and ensure smooth and clean translational movements during data collection. We will be moving the IMU fixture in a series of predefined paths using the LEGO MINDSTORMS EV3 kit we have on hand. For each trial, we will save the data as a .csv file on the Raspberry Pi for later post-processing.

We will then try to use various methods to post-process the raw IMU data. These methods will include traditional methods of time based double integration and sensor fusion, and also more modern techniques such as convolutional and recurrent neural networks, and n-th degree polynomial curve fitting. We will then try to compare the calculated path with the actual path traveled by the sensor and see how close we are to the true data. An example metric for evaluating the accuracy could be to calculate the sum of deviation from the actual path at different locations along the path.

Hypothesis: We hypothesize that by using double-integration, band-pass filters, machine learning, or a combination of various methods, we can acquire translational position data from the 6 DOF IMU sensor.

Milestones:

Week of March 30th:

- Finalize our revised project plans and submit project proposal
- With access to a 3D-printer and LEGO kits, Eric will finish building a test fixture design for a cable-free Raspberry Pi and IMU setup
- Jeremy will write a data logging program to record all the IMU data on the Raspberry Pi.
- We will begin brainstorming possible translational paths to use for collection of IMU sensor data.

Week of April 6th:

- We will finish designing and building the IMU fixture and the LEGO setup to control it (a 2 to 3 DOF robot arm or a 3-wheeled rover) that will move our IMU fixture in a predefined, translational path.
- We will begin collecting IMU sensor data and perform multiple trials of the same path.

Week of April 13th:

- We will finish collecting the raw IMU data and begin processing the .csv files
- We will investigate pre-processing methods to smooth the data (band-pass filters, moving averages, and gaussian smoothing)
- We will investigate post-processing methods of computing the absolute translational difference as observed by the IMU sensor.
- These tasks will be divided evenly so that we can investigate more ways overall of applying transforms to the data.

Week of April 20th:

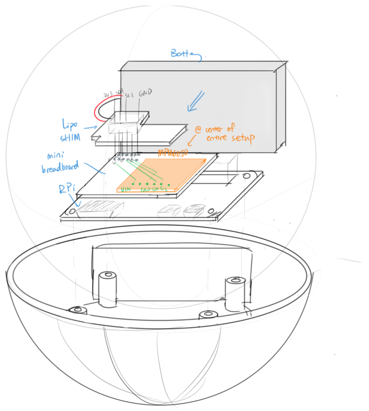
- We will finish applying post-processing methods to the data and aggregate the data according to our pre-defined metric of success.
- We will organize our finds and finish the project presentation and report with the collected data.

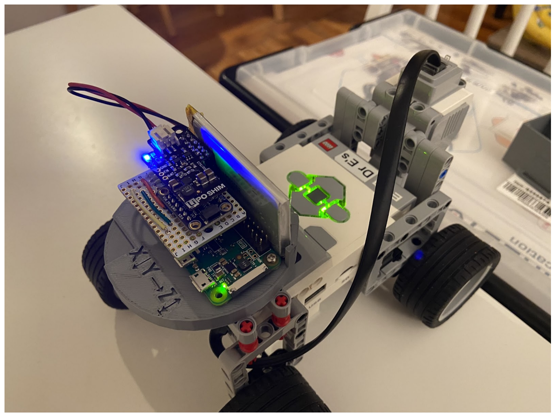
Appendix B: Eric Wu Logbook Entries

Project Checkpoint - Week 2 Date: 03/30/20 - 04/05/20

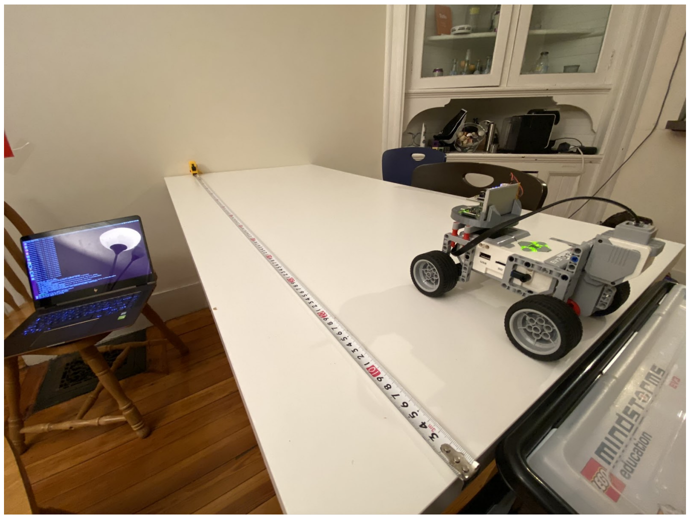
Accomplished:

1. Test Fixture Design and Building
   With the equipment I have on hand, I was able to quickly come up with a cable-free test fixture by soldering an MPU 6050 IMU sensor to a mini breadboard and attaching the breadboard to the raspberry Pi, which is powered by a 1-cell Lipo battery. I also designed and 3D-printed a jig that allows me to attach the setup to LEGO pieces (see attached pictures of design and prototypes.)





2. Pilot Data Acquisition
   With the test fixture I built last week, I started collecting pilot data using one IMU sensor. The first test procedure we came up with is to build a 1D LEGO car that will move to a set distance, stop for a moment, and then move back to the original position. By doing so we are trying to make the movement consistent with each trial by sending the same speed command to the LEGO Motor.



I performed a total of 15 trials, and all of the csv files could be found through this link:
https://github.com/0xJeremy/ME-18-Final/tree/master/data/0405_1d
Video showing me testing the performance of the setup:
https://youtu.be/Od3rj-Xdm3U
Video taken during one of the data acquisition trials:
https://youtu.be/Wtaci9q5_MU

Challenges:

I noticed that without any wait function in my code, the Pi was only able to record the data at a maximum speed of 55Hz. It might be just enough for the purpose of our experiment, but it would be nice to see if we could improve the sampling rate.

During data collection, I also noticed that the EV3 LEGO motor twitches when the motor stops moving, which resulted in a noticeable oscillation. This could be picked up and accidentally "magnified" by the IMU sensor. No conclusion could be drawn before we could visualize the data.

Goal of Next Week:

Visualize the pilot data using python, and start processing it with various data-smoothing and data-filtering techniques.

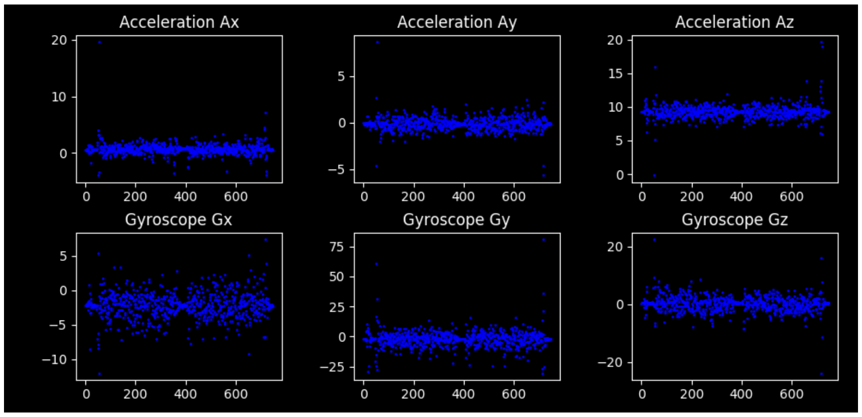
Communicate with my project partner, and decide whether we want to change the data acquisition method, sensor setup, code infrastructure, etc.

Project Checkpoint - Week 03 Date: 04/06/20 - 04/13/20
Accomplished:

1. Pilot Data Visualization
   With all the data I collected by the end of last week, My partner Jeremy started visualizing the data on his end and shared the results. Here are some example plots which contain the raw data recorded by the IMU sensor during one of the trial. During the specific trial, the Car moves 110

cm forward, pauses for half a second, and drives back to its original position. The acceleration and deceleration of the car was made consistent by setting the appropriate speed command in the LEGO program.

A total of Three Trials of the same movement was performed, and the raw results from one are shown below. I was moving the IMU sensor in the positive x direction.



From the raw data, it seems there is a lot of noise within the sensor reading. That said, we could see that there were some drastic readings coming out of the X acceleration reading at the start and end of the data, which is when the car speeds up and slows down - this is consistent with the direction to which I directed the car towards during the trials. We could also see that the Y axis remains at around zero, and the Z acceleration was constant at just below 10 (which is consistent with g = 9.81m/s^2).

2. Sampling Rate Improvement
   After spending some time playing with the sensor and the Raspberry Pi, I found that by only recording the X and Y acceleration data and getting rid of the print function through SSH, I was able to increase the sampling rate to about 125 Hz. This should be sufficient for the purpose of our project.

   Visualization of data, low-level code for data acquisition, high-level code for data post-processing can all be found in the github repository: https://github.com/0xJeremy/ME-18-Final
   My contributions to the Repo are mainly the "data_logging" folder, which contains the code I wrote to record data using the Raspberry Pi, and the "data" folder which contains all the csv files generated (and organized in sub-folders).

Challenges:

Without running the raw data through some data smoothing algorithms, We did not seem to get a lot of useful data from the sensor during the initial trials.

Goal of Next Week:

We plan to try to improve the results out of the sensor by splitting the tasks in the following way:

--I will continue to play with the physical setup and see if different translational movements can generate better data for post-processing, and I will record more data to share it with my partner.
--Jeremy will continue looking into post-processing methods, as well as improving the code he uses to process the data.

We will also be frequently communicating and switching tasks during the rest of the project, so that each of us will have some experience with both data acquisition and post processing.

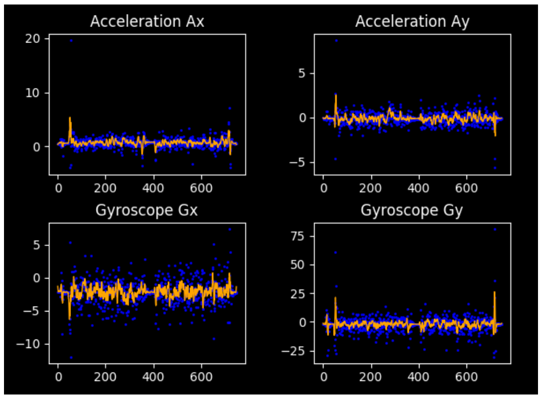
Project Checkpoint - Week 04 Date: 04/14/20 - 04/20/20
Accomplished:

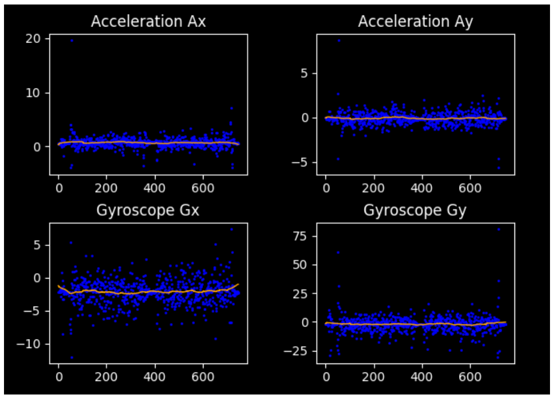
1. Understanding the Processing Methods
https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37
I came across this link as I tried to understand what's going on behind the data smoothing and filtering techniques. 1D Convolution Smoothing was one of the methods that appears to be quite effective.

Here is the smooth_convolution function that we incorporate as one of our post-processing methods. This part of the code is written by my project partner, Jeremy Kanovsky.

```
def smooth_convolution(data, smoothing_coefficient=1):
        box = np.ones(smoothing_coefficient)/smoothing_coefficient
        data_copy = copy.deepcopy(data)
        for key in data:
                if key == 'time' or key == 'delta':
                        continue
                data_copy[key] = np.convolve(data[key], box, mode='same')
        return data_copy
```

By changing the "smoothing_coefficient" that gets passed into the function along with the 1D data, we can adjust how much smoothing the function performs on the given data. Some examples:





SC = 5 SC = 100

As shown from the plots above, I noticed that simply increasing the smoothing coefficient does not necessarily lead to a more promising result. Too much smoothing might accidentally wipe out trends in the data.

2. More Data Acquisition

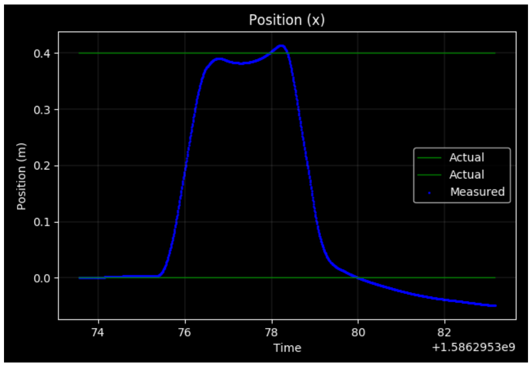
   In order to bring the post-processing study to the next stage, we need to collect some new, good data. This week, using the set up I previously built, I spent some more time performing trials on the IMU sensor.

   For the second set of data, I was moving the test fixture by sliding it back-and-forth smoothly across a fixed distance. The motion is limited to 1D by placing the fixture on a 4-wheel. unpowered LEGO car.

   For the third set of data, I used the same physical setup, except that instead of moving to a position and return to the origin, I moved the car to several stop points (e.g. 0cm - 40cm - 20cm - 60cm).

3. Updated Sample Data

   For this week's deliverable, we were able to perform some post-processing on our second set of data. We first load the .csv file to the program, retrieve the Acceleration data from the x direction; we then run the data through one of our pre-processing tools to smoothen the data, such as convolution smoothing, double digital filter, etc; After filtering the data, we performed double-integration on the dataset, assuming a starting position of 0m and an initial speed of 0m/sec.



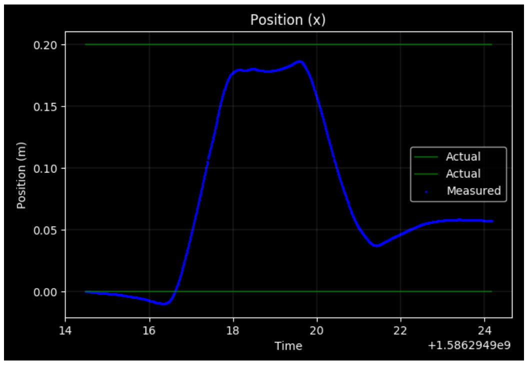


Figure 1 Figure 2

For the figures above, the car was manually moved by me to 20cm and 40cm. I paused for approximately 2 seconds, and moved the car back to its original position. The blue line shows the calculated position with respect to time; as a comparison, we also drew two green lines representing the actual position that I moved the sensor to. We generated a total of 33 graphs from the .csv files, and all of them can be accessed through here: https://github.com/0xJeremy/ME-18-Final/tree/master/images/double_integration_position_plots

From figure 1 and figure 2, we could see that by a simple double-integration, we were able to interpolate the position from acceleration data to a reasonably accurate degree. However, we do see that the value tends to drift after a sudden change in acceleration. This is mainly present in the part of the data after I moved the fixture back to the original position - the value starts to drift even when the car is not moving at all. This could be mainly caused by the small drifts from the raw sensor readings, which somehow got magnified during the data-smoothing of the double integration process.

Challenges:

The results from all 33 plots from data set #2 are fairly inconsistent. At about half of the times, the plots generated were either way too different from the actual distance or were giving some weird predicted values. I personally found this a little surprising, as I had believed that I was doing a pretty good job in maintaining the consistency of my maneuvers on the IMU LEGO car.

After talking with Professor Huang and the TAs, there are two potential improvements we could try implementing: One is to add in another sensor, with maybe a different axis inline with the movement of the car. Hopefully the two sensors can offset the drifts that each of them would have produced individually. The other is find a way to move the car in a more

Another challenge might come up if we eventually wanted to get the position data from real time. As we are using a Raspberry Pi zero, doing such computations might significantly slow down the sampling rate and thus might negatively affect the accuracy.

Goal of Next Week:

Finish processing the third set of data, which contains some more complicated movements of the sensor fixture. If time allows, do some more trials with improved stability during movements or with the second sensor added.

Warp up the experiment and start making the presentation video for final submission.

Project Checkpoint - Week 05 Date: 04/21/20 - 04/27/20

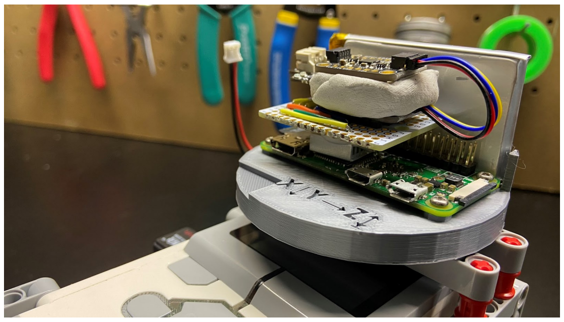
Accomplished:

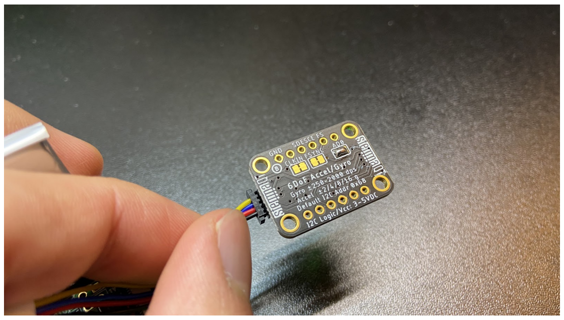
1. More Data Acquisition

   I got feedback from Jeremy, who had been performing most of the data post-processing to try to extract useful positional information from the sensor data we collected. It seemed that we might still want to introduce a second sensor into the experimentation setup. This should be fairly easy and straightforward, as the MPU6050 is designed to communicate through I2C.

   So I take a closer look at the sensor and the code:
   https://learn.adafruit.com/mpu6050-6-dof-accelerometer-and-gyro/pinouts

   After some research I found out that all of the MPU6050 from Adafruit have the default 0x68 address assigned to it. However, there’s a jumper at the back of the board and soldering the plates together would close the connection and make the address 0x69 - perfect for our purposes! I also modified the code by declaring a second sensor object of the same class. Below are images of the final setup:





I then did 3 sets of data acquisition, all of them have the same movement sequence (0cm - 20cm - 60cm), but have different value output to the csv files:

1 sensor, all 6 axis of data
2 sensors, 3 axis of acceleration only
2 sensors, all 6 axis of data

During data collection, the sampling rate of the Pi reduced by at least half the rate - mostly because I increased the amount of data being loaded to the csv files.

Updated csv files as well as the code have been updated and pushed to the github repository.

Challenges:

None-my part of the project is almost done. Perhaps the only challenge left for me is to try and understand all the code that Jeremy has been writing:) .

Goal of Next Week:

At this point, we should be all set in terms of data collection. The only steps left for this project would be to finish the final report and the final presentation video.

Appendix C: Jeremy Kanovsky Logbook Entries

Project Checkpoint - Week 2 Date: 03/30/20 - 04/05/20

Accomplished:

1. Data Acquisition

   This week we began collecting data for our experiment. Eric worked on CADing and fabricating a test fixture that can be attached to LEGO pieces. He also built a LEGO car to drive our accelerometer around a table at a constant rate. This is extremely useful because it let's us eliminate noise in the dataset without any processing. It is possible this reduction of noise will make the model unrealistic, but we can always remove the test fixture later and move it by hand. The data collected from this week was uploaded to GitHub for processing. All files were in .csv format, making them easy to parse in Python (which, we decided, would be our language of choice for processing).
2. Data Processing

After Eric uploaded the data from this set of trials, I began working on data processing. I started by finding a way to ingest the data into Python. Because we recorded a .csv, I wanted to find a method to parse the individual columns into a dictionary. I also wanted to be able to ingest multiple files at once so that when we started working on bigger datasets it would be easier to get into a usable format. Initially our data didn't contain a timestamp which was unfortunate because it makes double integrations difficult. Some rather robust data tools were developed this week, which can be found here:
https://github.com/0xJeremy/ME-18-Final/blob/master/pre_processing/load_data.py

Challenges:

As mentioned above, we don't have timestamps on the dataset. This is not problematic at the moment, but will be in the future because most data processing techniques for double integration rely on a time-element. Without it we won't be able to account for the (albeit small) variations in clock speeds between trials, and between file writes. Definitely something we want to add in the future. It would also allow us to calculate a "delta" column after the fact for use in other processing techniques.

Goal of Next Week:

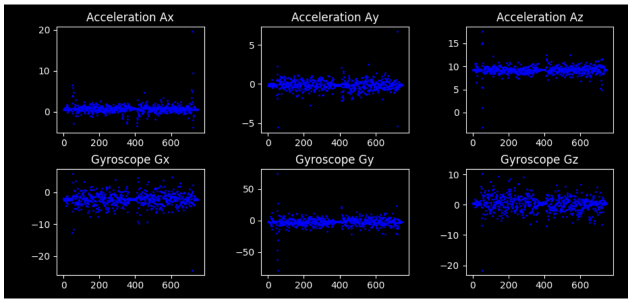
Next week we want to begin visualizing the data, and begin adding some smoothing techniques. We should also begin researching smoothing techniques before the fact to avoid wasting time on some that might not work.

Project Checkpoint - Week 03 Date: 04/06/20 - 04/13/20

Accomplished:

1. Data Visualization

   This week we began visualizing the data. Below is a chart of all six axis of data we recorded (3 accelerometer and 3 gyroscope). The data ingestion techniques made last week made this process surprisingly easy. In the future however we will have to standardize a naming convention for the files. I think it would be possible to store metadata about each trial in the filename and load that dynamically. Something to think about and talk about with Eric. One interesting thing to note about the data is the effect of gravity is quite pronounced. It may prove interesting to eliminate this "bias", and use similar techniques to eliminate inherent bias from the other columns. It might potentially be worth trying to eliminate drift this way.



2. Smoothing Techniques

This week we also began working on smoothing techniques. We very quickly came up with four different techniques: a moving convolution, the Savitzky-Golay filter, a rolling window, and a double digital filter (forward-backward). Initial tests with these filters show promise. It's unlikely we will need to use more than one, but it's nice to have options. Something interesting is that they all filter the data in slightly different ways, but we can get approximately the same effect between them all. It will be good to investigate if one is actually better than the rest, or if they are all "good enough".

Challenges:

There weren't any major challenges this week. The data pipeline between trials and uploading to GitHub and processing in Python is fairly robust and serviceable. The largest difficulty is how to deal with different file names between trials. I'm working on a system for metadata processing using a filename.

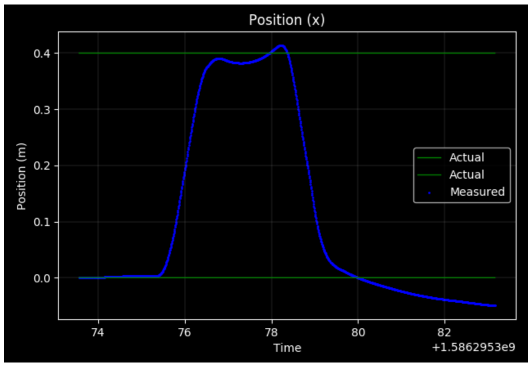
Goal of Next Week:

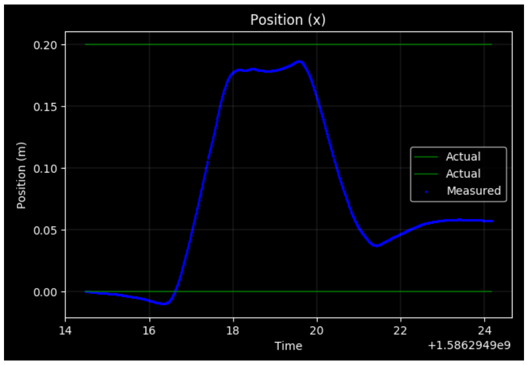
Next week we will need to invest some quantity of time in data tooling. While it's manageable at a small scale, it seems we need to make small adjustments with every dataset. It would be nice to have comfort knowing we can load any dataset without modifying the main processing structure. On Eric's end, we will probably continue collecting data and different translational movements.

Project Checkpoint - Week 04 Date: 04/14/20 - 04/20/20

Accomplished:

1. Data Visualization
   This week I put most of my time inventing methods of visualizing all aspects of the data in as flexible a manner as possible. We developed ways of plotting the acceleration, gyroscope, and individual columns of data, as well as the entire dataset. This is useful for auto generating plots of the data all at once for side by side comparison. I also began looking into 3D plots. When we have translational algorithms, we can probably verify them by observing the motion in three dimensions and comparing it to the actual movement. All visualization techniques can be found here: https://github.com/0xJeremy/ME-18-Final/blob/master/visualization/plot.py.
2. Beginning Post Processing
   This week I also began experimenting with post processing techniques. Because of the new visualization framework we can experiment with different algorithms and compare them to the actual movement. Initial work with double integration looks promising. Below are two plots generated from doing a double time integrations on the x-axis accelerometer data.





While these two plots make it appear that we are fully successful, other plots (not included) are less optimistic. These two plots are most likely outliers in the dataset, and only appear correct by coincidence. Drift is starting to appear to be a problem. In the second graph the tail end of the data is likely caused by drift in the sensor. We're not quite sure how to correct this, but will continue investigating.

Challenges:

While the plots shown above are very consistent with the ground truth of the data, I am a little concerned that as we move forward this will not always be the case. We are going to have to investigate more methods for translational pose estimation beyond double integration (if it were that easy it would have been done long ago). We're a little unsure what next steps might be on this front, but got some good ideas from Professor Huang and the course TAs. Eric has also been working on the Raspberry Pi to investigate how we can improve data collection rate.

Goal of Next Week:

We would like to investigate other methods of performing post-processing on the data. One early idea was to use machine learning, but that seems unlikely that we can collect a large enough sample size to make it viable. It also seems problematic to solve this problem non-deterministically because that may lead to edge cases never accounted for in the data.

Project Checkpoint - Week 05 Date: 04/21/20 - 04/27/20

Accomplished:

1. Data Investigation
   This week I began to work on further investigations into the data. One common thing I began to notice was drift in the calculated velocity over time of the sensor. While I only computed this in a single axis, it likely occurs in all of them. There are, however, techniques to remove this drift. I am having a little trouble implementing them, but think it's only a matter of time invested. A very interesting technique I found was the one outlined here: https://dsp.stackexchange.com/questions/34463/removing-drift-from-integration-of-accelerometer-data. I intend to work through the methods proposed in the post, and see if they work on our dataset.
2. Data Collection

Eric has (very kindly) agreed to collect several other datasets for us to work with. I don't yet know exactly what kind of data we need to be the most successful, but the more data to work with will probably prove useful. Some techniques for reducing noise seem to imply sensor fusion of multiple IMUs can be used to reduce overall noise and drift. Hopefully we can align the data and use it to cancel out artefacts.

Challenges:

A challenge this week, that will likely persist into the future, is accumulated noise in the velocity and position estimates.

Goal of Next Week:

I would like to focus the next week on solving the issues that arise in data processing, and working on more advanced methods of data interpolation. We will also begin working on our final report and video presentation.