# Proof of Concept

Source Code: [Click here](https://github.com/justintam5/Autonomous_Hair_Touch_Ups)

To prove our concept, we wanted to build a device that would be able to accurately track the position of the device with respect to time. We knew that this would be the most difficult part of our design as it consists of tracking a device's position with only an accelerometer, a task that has not really been done before and is very specific to the type of motion being tracked. Most motion tracking devices usually use some other form tracking(use of camera, LiDAR, GPS, etc.) to account for error present after accelerometer data has been processed into displacement data. Our final design also must incorporate tracking of the rotation of the device, but this has already been done before, and thought it more important to get an understanding of how the displacement measurements will be evaluated. To make it simpler, we only process 1 dimension of acceleration data into positional data as it is simpler to work with and can easily be translated into multiple dimensions for the final design.

## Obtaining the Accelerometer Data via Bluetooth:

The design for the proof consists of three breakout boards, a microcontroller breakout(Arduino pro mini), a slave bluetooth module(HC-06), a 9-axis inertial measurement unit (IMU)(MPU-9250), and two 18650 3.7V batteries for powering the Arduino. The microcontroller was programmed using Arduino as it presents an easily usable set of libraries specifically for programming the on board ATmega328 chip, and many other related microcontrollers. As the MPU-9250 is a common IMU, as well as, bluetooth is frequently used to interface with Arduinos, there are lots of libraries already out there to make the process simpler. The programmed microcontroller continuously reads digital values from the IMU over the I2C bus, and transmitted from the microcontroller to the HC-06. A connection is made between the HC-06 and the master bluetooth module, as well as between the master bluetooth module and the serial COM port on the computer. This way, we obtain acceleration values that are compiled into a file that can be used for analyzing and post processing.

## Processing the Accelerometer Data:

To obtain a function of position over time from acceleration over time, we double integrate, knowing the initial velocity and position(both 0 in this case). In the context of an approximate 300hz\*time array(for 10 seconds, this produces an array of size 3000) we numerically integrate from 0 to t for each element of the acceleration array. This process is repeated to obtain an array representing velocity and position. This is done in python, where we use Simpsonn’s rule via the Scientific Python API.

### Reducing Noise:

Interpreting the raw acceleration data also requires us to reduce noise, and error; see Figures 1 and 2. This is done using a low pass filter(LPF), and run to the raw acceleration signal through the LPF at a sampling rate equal to the refresh rate of the MPU-9250(~300Hz), at an experimentally cut-off frequency(see Validation below). The result is a signal where its high frequency components have been filtered out, and the lower end of the spectrum(i.e. below the cut-off frequency) that represents the signal is kept. The idea is that the high-frequency components represent the noise, as they change rapidly with respect to time. The result is shown in Figures 3 and 4. The range is much lower, and the noise is much less.

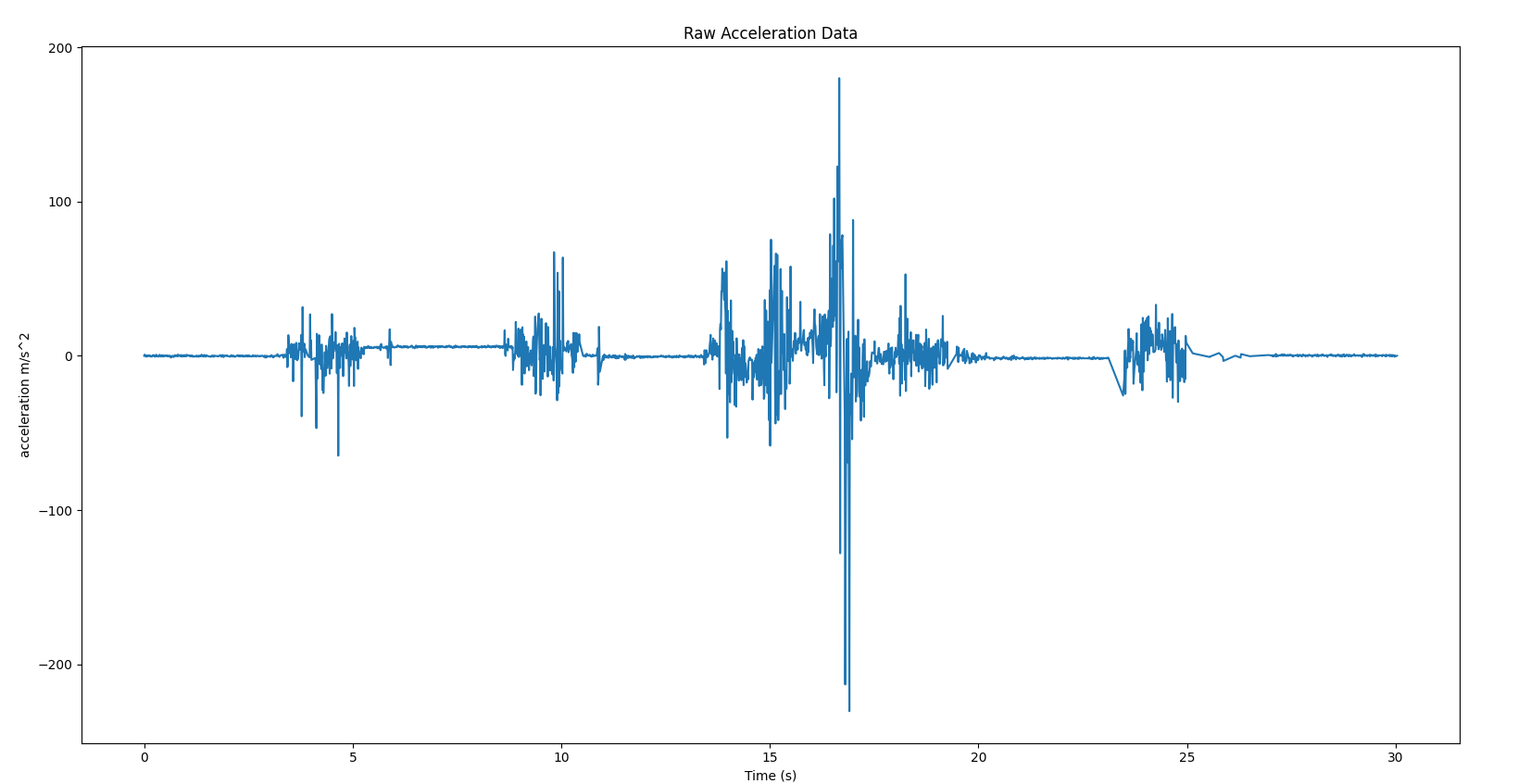


Figure 1: Raw Acceleration Data

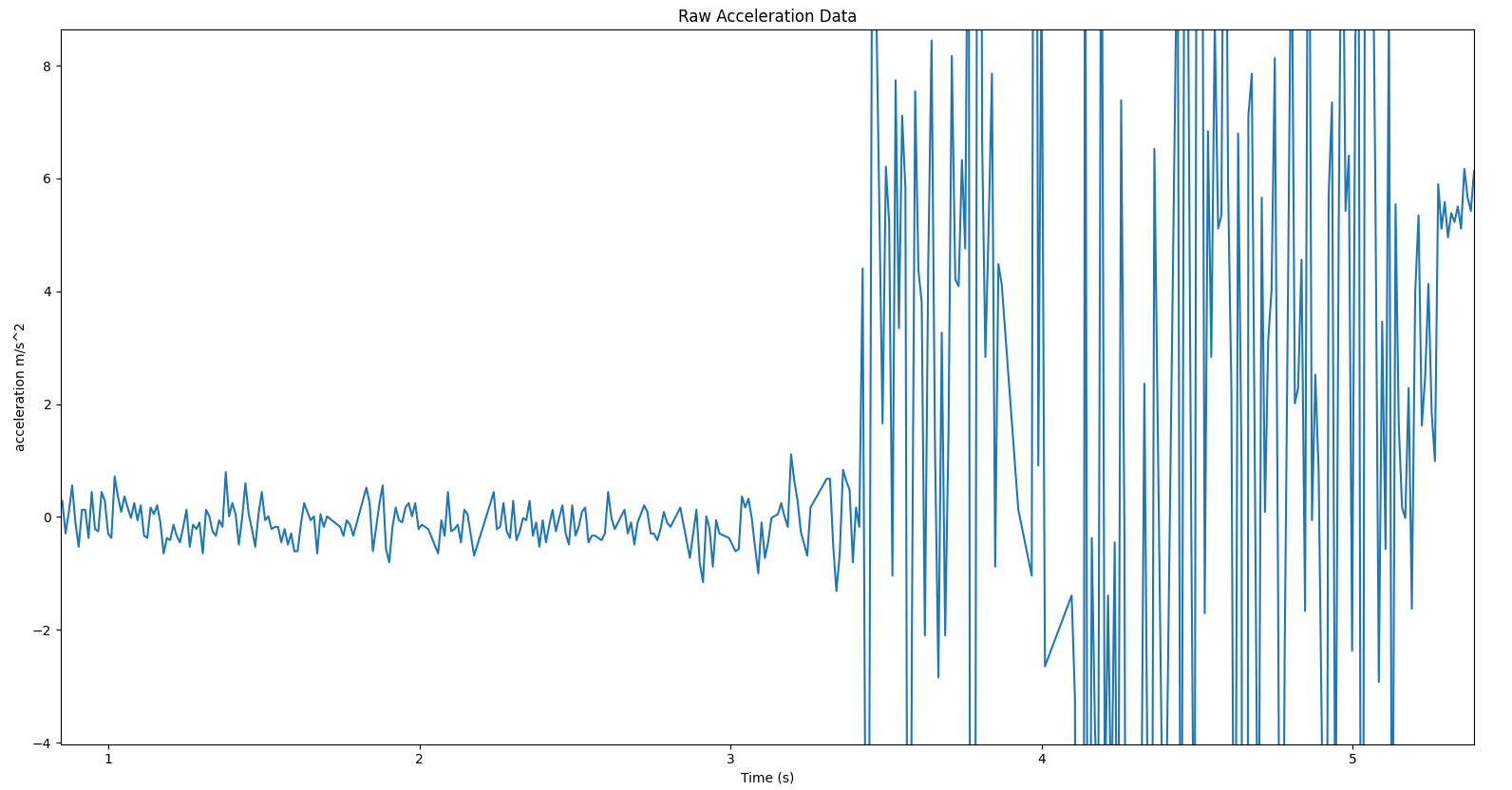


Figure 2: Raw Acceleration Data, Zoomed In

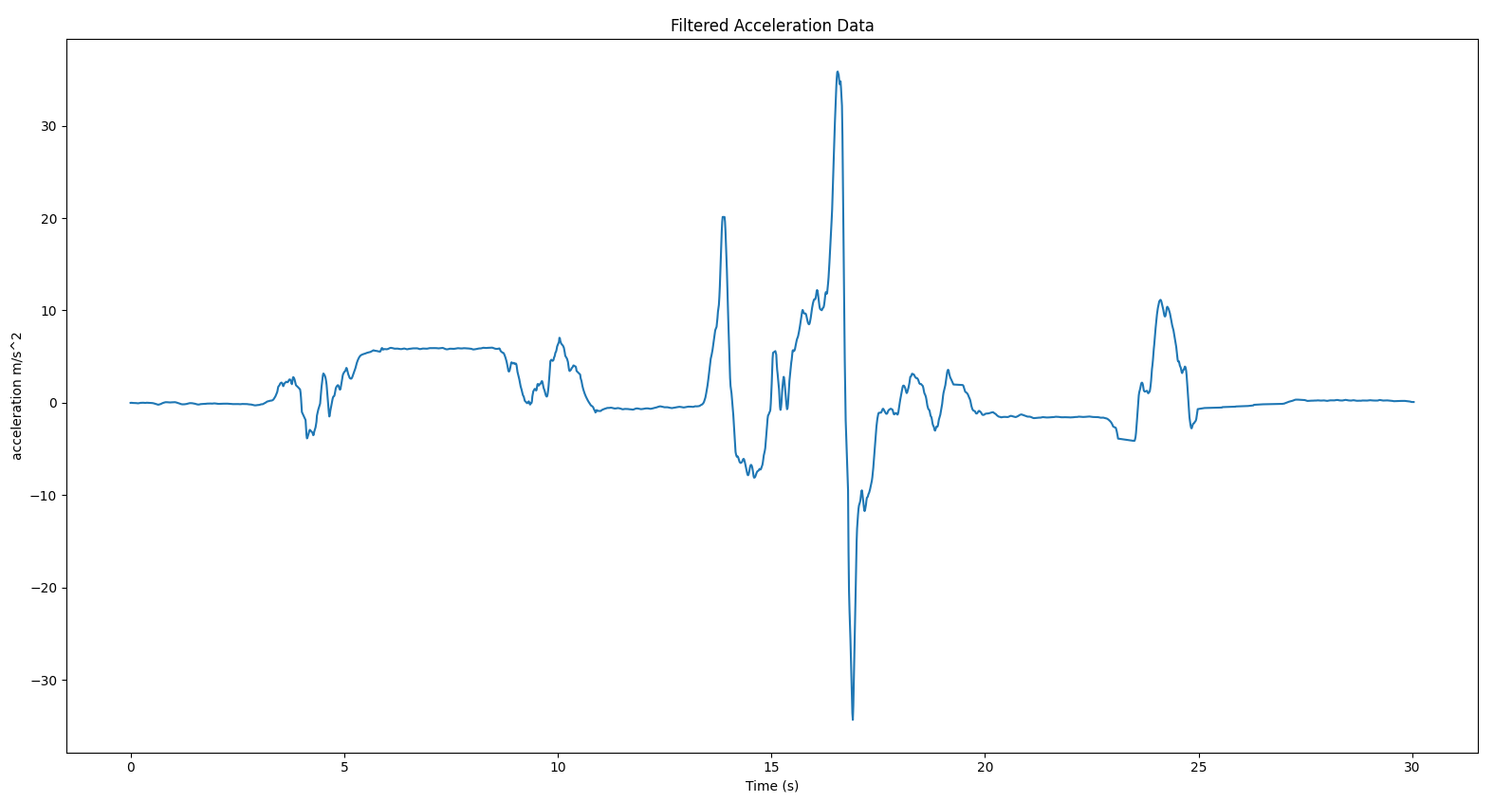


Figure 3: Filtered Acceleration Data

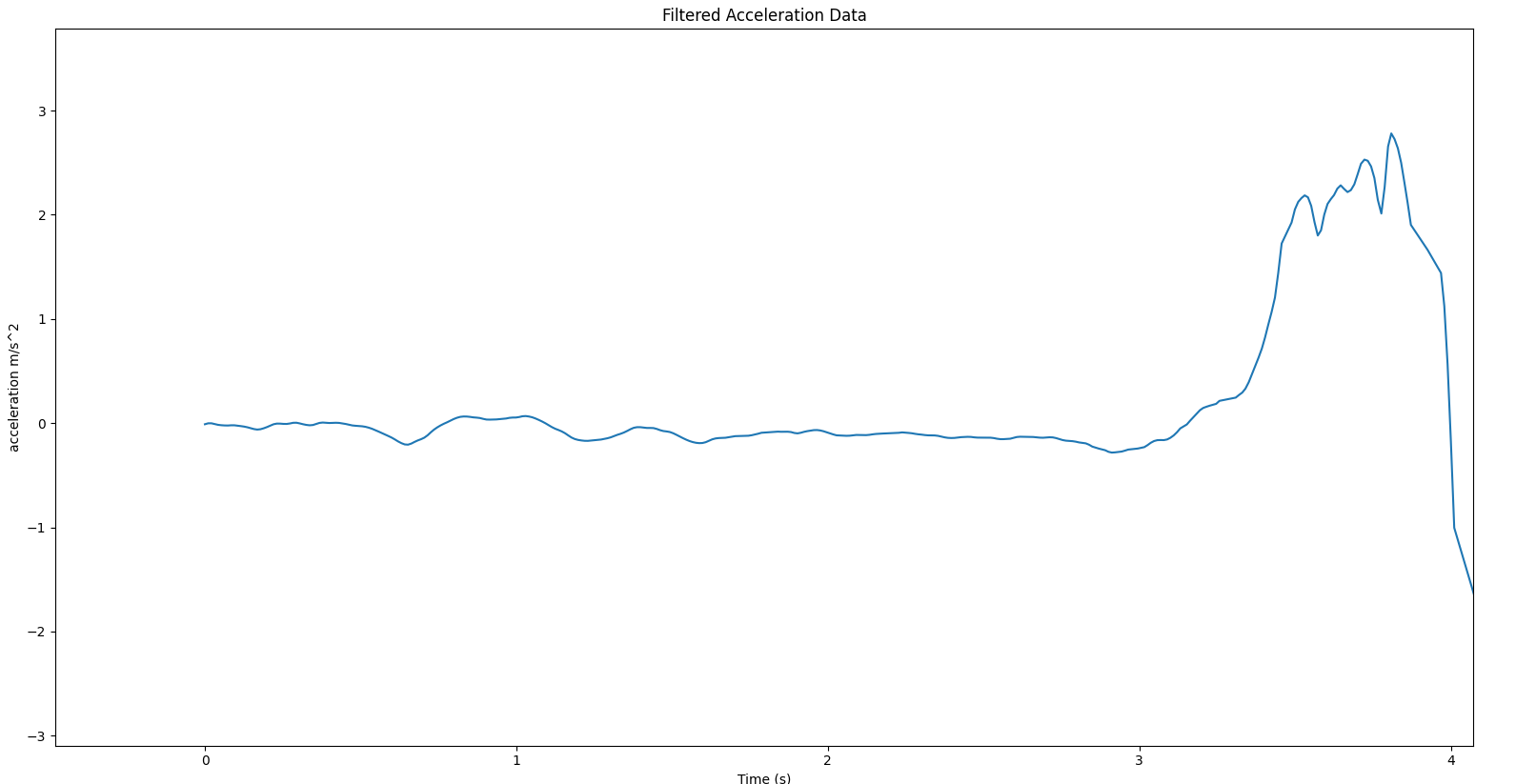


Figure 4: Filtered Acceleration Data, Zoomed In

Despite the LPF, we still obtain some non-negligible errors in the acceleration signal. Since we are hoping to obtain the position over time, this creates a massive problem since any constant error in acceleration will present itself as the same value multiplied by the given time squared. This is caused via the double integration:

=

The errors stack, and only grow with respect to time as they are integrated to obtain position. What remains is then a method to reduce this type of error, commonly referred to as “drift velocity”.

### Reducing Drift Velocity:

To achieve this we follow a similar strategy in reducing the noise. That is, apply a high pass filter(HPF). The idea is that given the 0 initial conditions, any term that evolves slowly over time is likely to be an error, and so we seek to remove these by decomposing the velocity signal into its spectral components and filtering out the low frequency contributors. This is again done using python, and the Scientific Python API. For good measure, we also pass both signals through an additional LPF with a high cut-off frequency to further reduce any noise. A comparison between the raw data and filtered data is shown below in Figures 5 and 6.

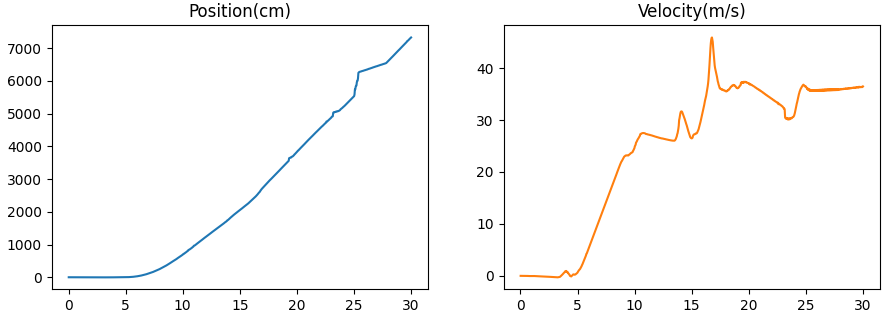


Figure 5: Raw Position and Velocity Data

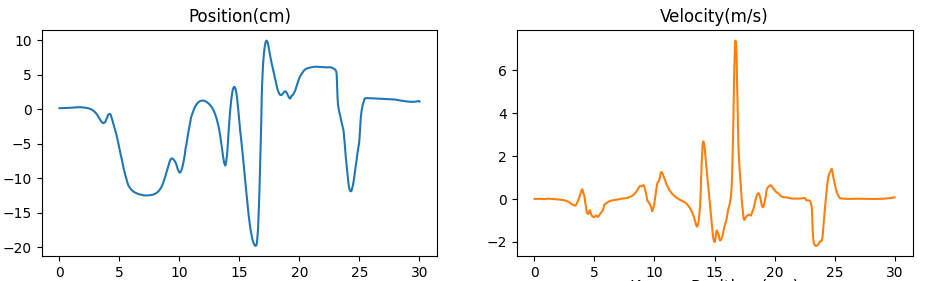


Figure 6: Filtered Position and Velocity Data

## Validation:

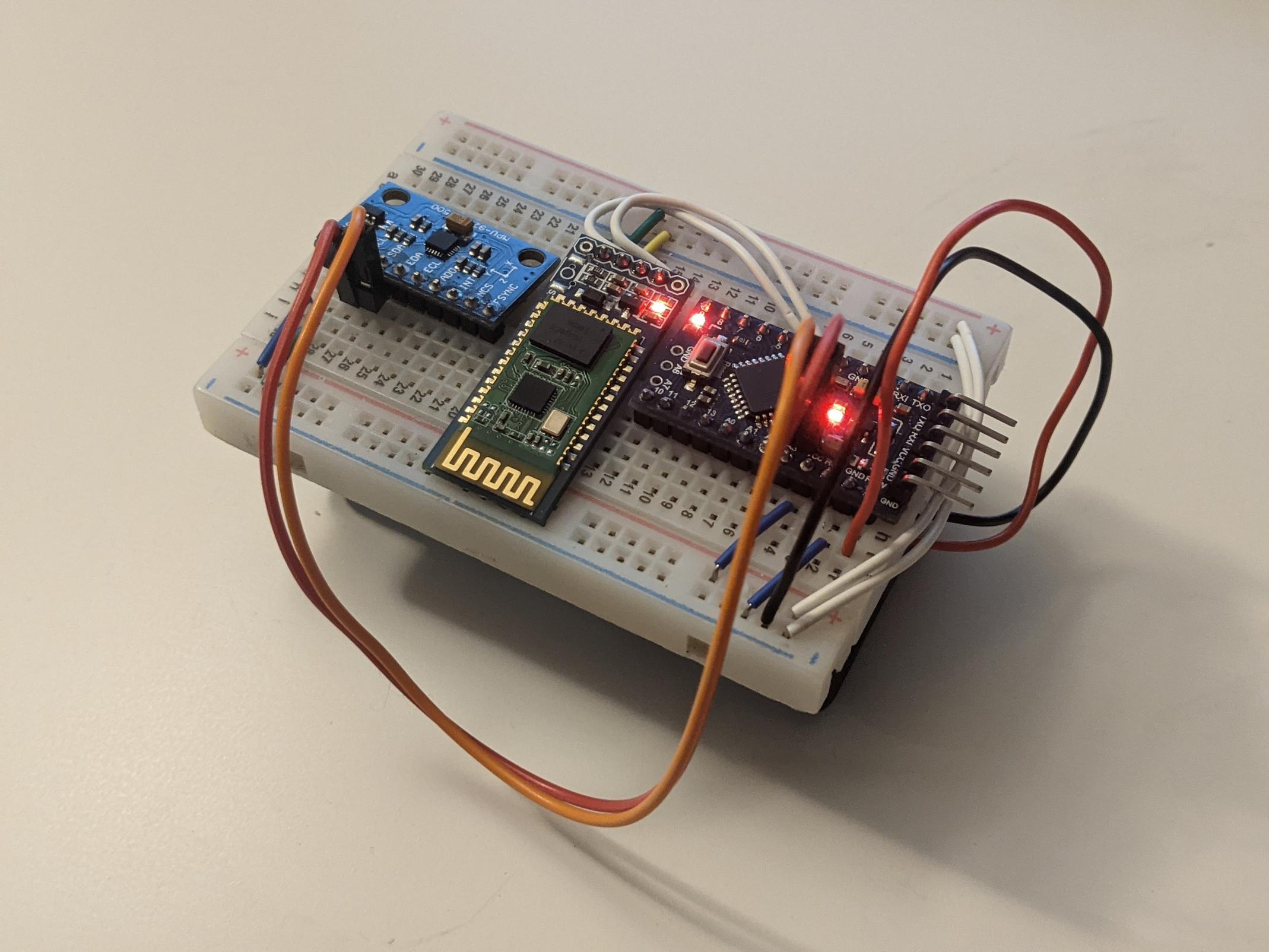


Figure 7: Proof of Concept Hardware Device

We needed a way of validating our test cases, so a camera was used to track the real position of the device while simultaneously recording the device's accelerometer values. Using OpenCV’s built in tracking class, we were able to easily follow the motion of the device and get corresponding pixel locations of where the device was during each frame. A regression line was fit to eight distance measurements that corresponded to a pixel location of the device measured by the tracking class. Knowing the camera records at 30 fps, each frame was associated with a time so that each displacement measurement had a corresponding time. Although the MPU-9250 sensor data is processed into positional data with the same frequency as the acceleration data, some of the validation needed to be done with the camera measurement and sensor measurement at the same frequency, so the frequency of the sensor measurements were reduced to match that of the cameras measurements(See Test\_Plan.pdf for details).

Comparing the measured data and the known data for a tracked data set, we obtain the final result and validation of our proof, shown in Figure

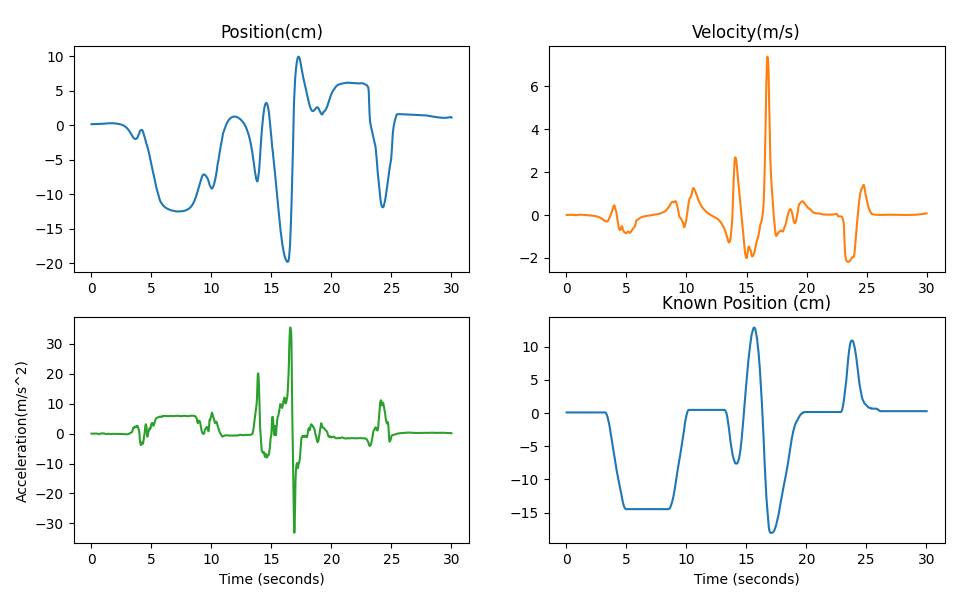


Figure 8: Final Comparison of Known vs Measured Data

## Discussion

As seen in Figure 8, the position data holds the same shape but has significant errors. A large part of this is due to the tracker used, a mere $10 CAD vs many standard trackers on the range of $500 CAD. All things considered, we are content with the data obtained and recognize the need, and possibility of improvement.

## 