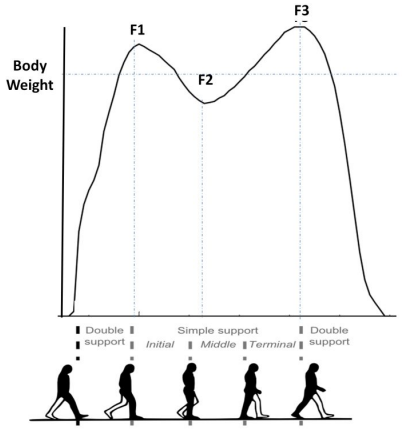
Smart Shoe for Wireless Ground Reaction Force Monitoring using Differential Pressure Sensors

Introduction

- What is Ground Reaction Force?
  GRF is the force exerted by the ground on a body in contact with it.

- Why do we need to monitor GRF ?
  - Insights into biomechanics of Human Movement
  (Gait Analysis, Balance Control, Force/Load Distribution)
  - Assessment & Identification of abnormalities & asymmetries
  (joint loadings, altered force patterns etc)
  - Injury Prevention, Performance Optimization, Rehabilitation
  - Assistive Device Design Improvement



[1]

[1] Bonnefoy-Mazure, A. & Armand, Stéphane. (2015). Normal gait.

Literature Review

1. A Wireless Human Motion Monitoring System for Smart Rehabilitation:

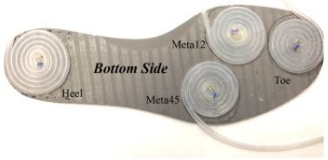
   The study introduces a wireless human motion monitoring system designed for gait analysis and visual feedback in rehabilitation training.

   The system incorporates inertial sensors and smart shoes with pressure sensors to capture lower-extremity joint rotations and measure force distributions during walking.

   The raw measurement data is used to calculate gait phases, step lengths, and center of pressure (CoP) to assess abnormal walking behaviors.

   User interfaces were developed & results demonstrated that patients could comprehend the visual feedback and achieved comparable training performance to traditional gait training with physical therapists.

   Overall, the research paper presents a promising wireless monitoring system that enables effective gait analysis and rehabilitation training.



(a)

[2]

[2] Zhang, Wenlong, Masayoshi Tomizuka, and Nancy Byl. "A wireless human motion monitoring system for smart rehabilitation." *Journal of Dynamic Systems, Measurement, and Control* 138.11 (2016).

Literature Review

2. A Wireless Human Motion Monitoring System for Smart Rehabilitation:

   The paper introduces a new analysis method that utilizes fuzzy logic to continuously and smoothly detect gait phases, enabling a comprehensive utilization of GCF sensor information.

   Additionally, the paper presents a higher-level algorithm that quantitatively monitors abnormalities in human gait, including improper GCF patterns and incorrect gait phase sequences.

   The proposed methods are implemented using sensor-embedded shoes, referred to as smart shoes, which feature four GCF sensors installed between the cushion pad and the sole.

   Overall, the research paper introduces an approach to gait monitoring using air pressure sensors and provides insights into the detection of gait phases and abnormalities, offering potential applications in advanced rehabilitation systems.

[3]

[3] Kong, Kyoungchul, and Masayoshi Tomizuka. "A gait monitoring system based on air pressure sensors embedded in a shoe." *IEEE/ASME Transactions on mechatronics* 14.3 (2009): 358-370.

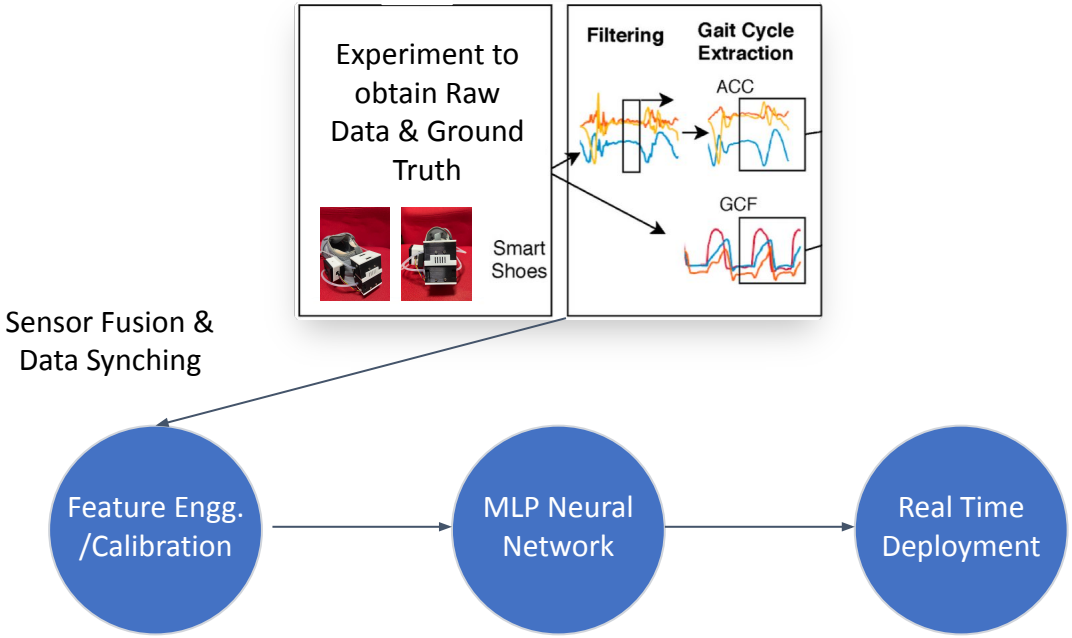
Motivation

- Smooth and Continuous Gait activity & GRF detection
- Pressure map with individual raw sensor readings:
  Heat Map for Loading and Force distribution
- Wireless & Portable Analysis:
  Convenient for End User to monitor daily life activities
- Exploring and Leveraging ML techniques in predicting real time GRF
- Inexpensive alternative for existing technologies
- Various uses in Rehabilitative and assistive research

Objective

The objective of this study is to demonstrate the development and application of a wireless and portable system to predict real-time Ground Reaction Forces through machine learning techniques while providing an inexpensive alternative for existing technologies, and exploring its potential in rehabilitative and assistive research.

Project Phases

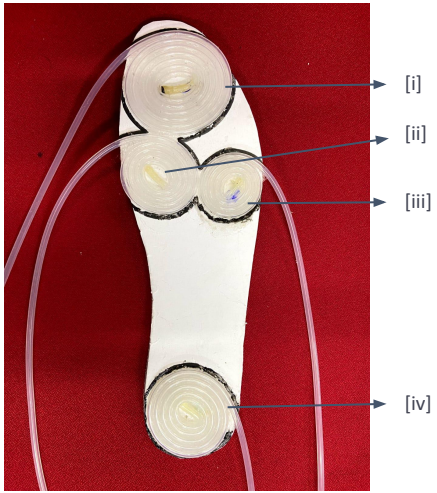


Sensors & Materials

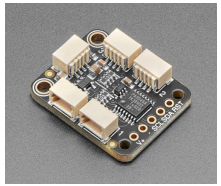
SparkFun Qwiic
MicroPressure Sensor

Portenta h7

Sparkfun ICM-20948
9-dof IMU



[4]

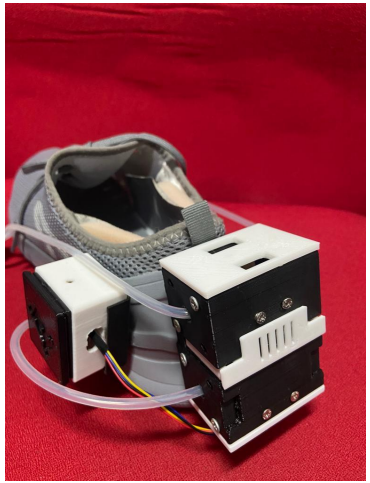


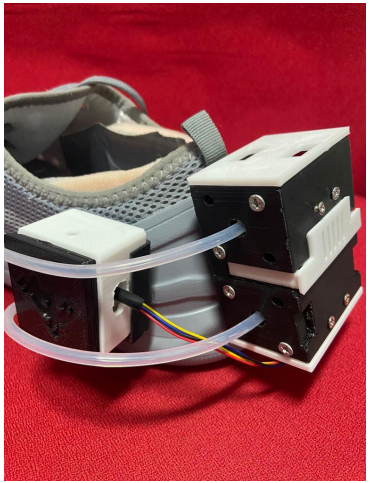
Adafruit PCA9546
4-Channel Multiplexer

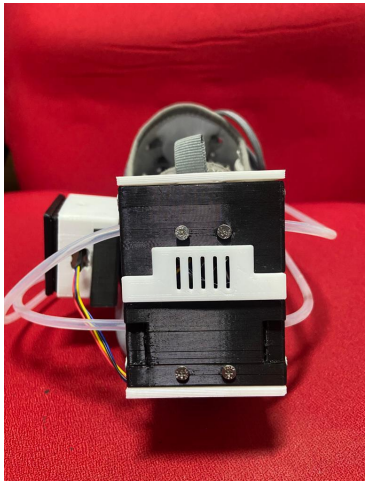
3.3V Micro SD module

[4] Shoe Insole with Silicone tube coiled pressure pads placed at: [i] Toe [ii] Metatarsophalengeal 1-2 [iii] Metatarsophalengeal 3-4 [iv] Heel

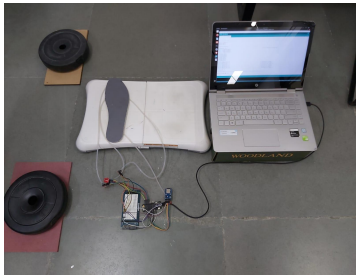
Experimental Design





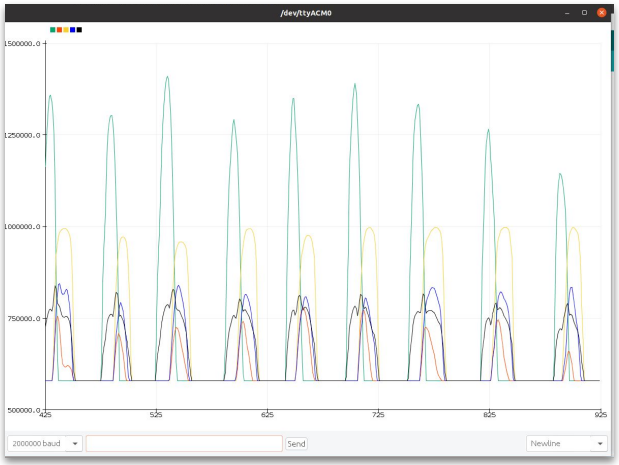


Data Collection

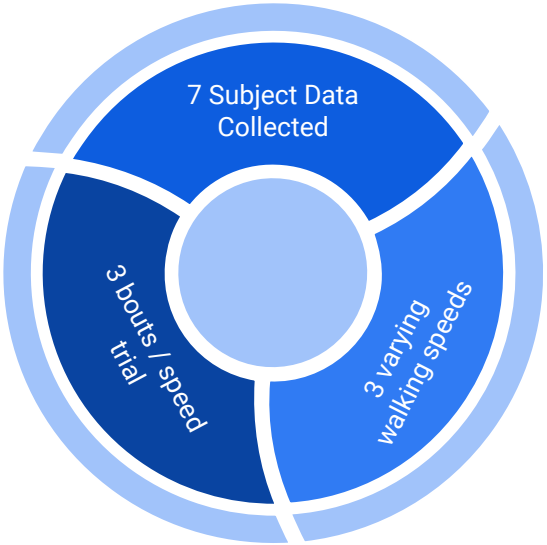


Before

Now



Data Collection



Data Visualization & Processing

Experiment Data Extraction from MATLAB

↓

Importing Libraries, Loading Feature DataFrame

↓

Normalizing Data and Creating Interaction Features

```
In [10]: df = pd.read_csv("matched_datafinal.csv")
         df.head()
```

Out[10]:

	t	w	s	ax	ay	az	gx	gy	gz
0	140.18	0.51363	455000	-511.23	1056.60	254.39	38.90	-13.78	42.26
1	140.19	0.74851	455000	-481.45	1053.70	268.07	46.71	-11.46	36.89
2	140.23	2.06570	455000	-463.87	1174.80	359.86	19.27	26.71	13.54
3	140.24	2.39500	455000	-462.89	1063.00	235.35	11.77	31.10	-0.43
4	140.25	2.72430	455000	-470.70	950.68	114.26	15.24	36.77	-18.17

```
In [11]: df.shape
Out[11]: (274, 9)
```

```
In [12]: # Create interaction features
         df['s_ax'] = df['s'] * df['ax']
         df['s_ay'] = df['s'] * df['ay']
         df['s_az'] = df['s'] * df['az']

         df['s_gx'] = df['s'] * df['gx']
         df['s_gy'] = df['s'] * df['gy']
         df['s_gz'] = df['s'] * df['gz']
```

```
In [13]: df.head()
```

Out[13]:

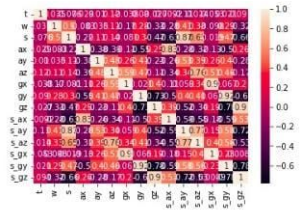
	t	w	s	ax	ay	az	gx	gy	gz	s_ax	s_ay	s_az	s_gx	s_gy	s_gz
0	140.18	0.51363	455000	-511.23	1056.60	254.39	38.90	-13.78	42.26	-232609650.0	480753000.0	115747450.0	17699500.0	-6269900.0	19228300.0
1	140.19	0.74851	455000	-481.45	1053.70	268.07	46.71	-11.46	36.89	-219059750.0	479433500.0	121971850.0	21253050.0	-5214300.0	16784950.0
2	140.23	2.06570	455000	-463.87	1174.80	359.86	19.27	26.71	13.54	-211060850.0	534534000.0	163736300.0	8767850.0	12153050.0	6160700.0
3	140.24	2.39500	455000	-462.89	1063.00	235.35	11.77	31.10	-0.43	-210614950.0	483665000.0	107084250.0	5355350.0	14150500.0	-195650.0
4	140.25	2.72430	455000	-470.70	950.68	114.26	15.24	36.77	-18.17	-214168500.0	432559400.0	51988300.0	6934200.0	16730350.0	-8267350.0

Visualizing Data, Outliers, Distribution, Correlation Matrix

Splitting Data for Training and Testing

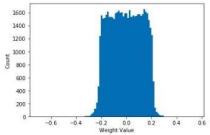
Model Training

```
In [32]: import seaborn as sns
         sns.heatmap(corr_matrix, annot=True)
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd3897f2a00>
```



```
In [33]: from tensorflow.keras.callbacks import ReduceLROnPlateau

         reduce_lr = ReduceLROnPlateau(monitor='loss', factor=0.1, patience=10, min_lr=0.001)
```

```
In [41]: weights = []
         for layer in best_model.layers:
             if hasattr(layer, 'kernel'):
                 weights.append(layer.kernel)

         # Concatenate all the weights into a single 1D array
         weights = np.concatenate([w.numpy().flatten() for w in weights])

         # Plot the histogram
         plt.hist(weights, bins=100)
         plt.xlabel('Weight Value')
         plt.ylabel('Count')
         plt.show()
```




```
In [325]: import numpy as np
          import matplotlib.pyplot as plt

          weights = []
          for layer in model.layers:
              if hasattr(layer, 'kernel'):
                  weights.append(layer.kernel)

          # Concatenate all the weights into a single 1D array
          weights = np.concatenate([w.numpy().flatten() for w in weights])

          # Plot the histogram
          plt.hist(weights, bins=100)
          plt.xlabel('Weight Value')
          plt.ylabel('Count')
          plt.show()
```

Model Training

- Why Deep Learning?

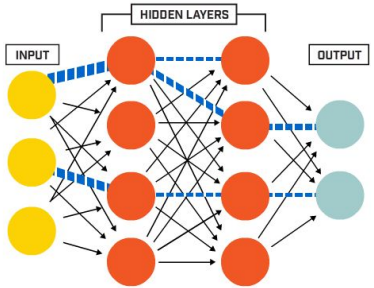
  Multiple Layers/hidden layers that distills some of the important patterns from the inputs and passes it onto the next layer

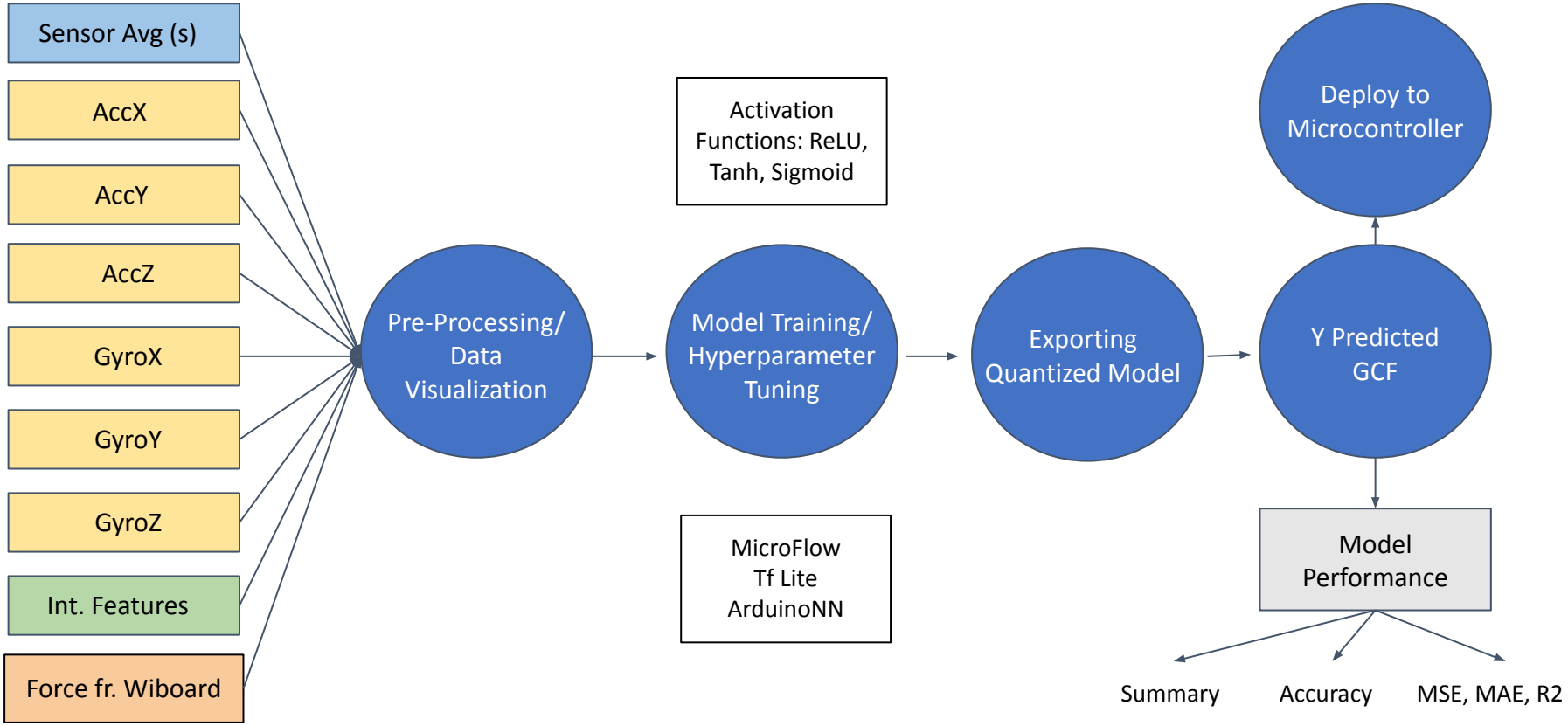
  The activation function serves to capture non-linear relationship between the inputs, therefore it has the ability to learn and model non-linear and complex relationships

  Does not impose any restrictions on the input variables (like how they should be distributed), feed-forward

  Iteratively different values of W are used and prediction errors assessed to get the optimal Ws (Back Propagation)

  Quicker Performance, Similar Accuracy with less data




Sensor Avg (s)
AccX
AccY
AccZ
GyroX
GyroY
GyroZ
Int. Features
Force fr. Wiboard
Pre-Processing/ Data Visualization
Activation Functions: ReLU, Tanh, Sigmoid
Model Training/ Hyperparameter Tuning
MicroFlow
Tf Lite
ArduinoNN
Exporting Quantized Model
Deploy to Microcontroller
Y Predicted GCF
Model Performance
Summary
Accuracy
MSE, MAE, R2

Model Validation

MAE:

refers to the magnitude of difference between the prediction of an observation and the true value of that observation.

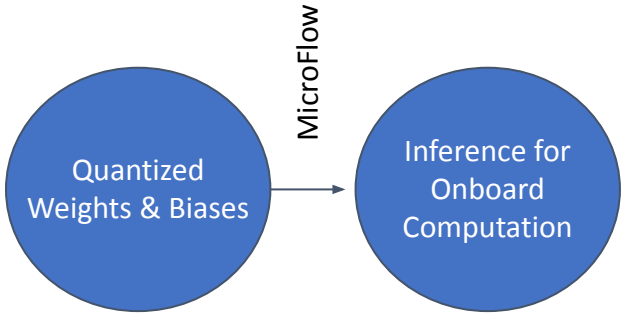
MSE:

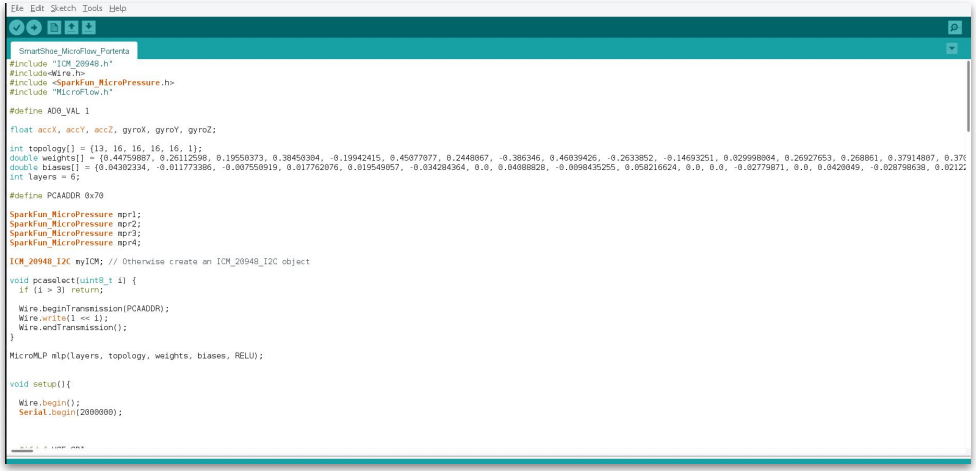
measures how close a regression line is to a set of data points. It is a risk function corresponding to the expected value of the squared error loss

R2 Value:

acts as an evaluation metric to evaluate the scatter of the data points around the fitted regression line

Model Deployment





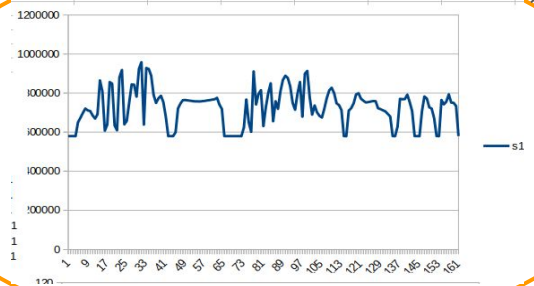
Preliminary Results with Wi-Board

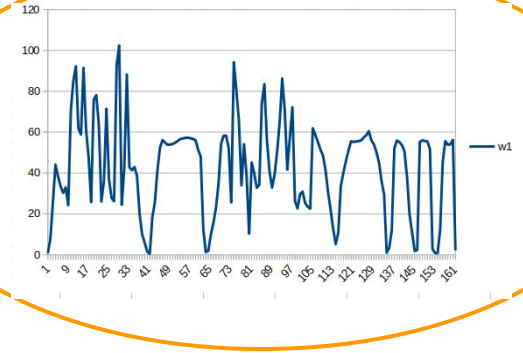
Input Features

Output Features

IMU 6 Axis

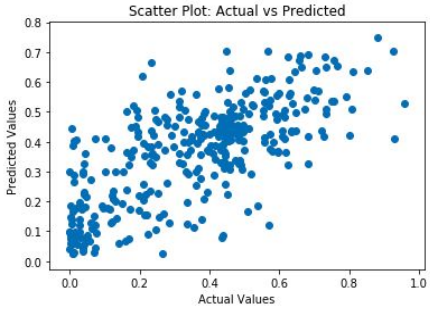
Pressure Av (s)

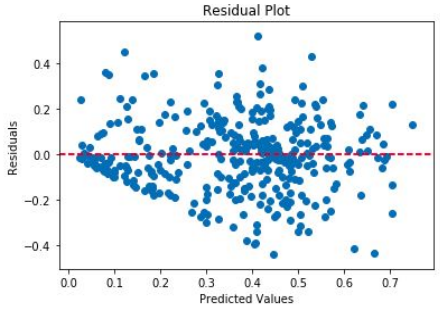




W

```
# Residual Plot
residuals = y_test - predictions
plt.scatter(predictions, residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='r', linestyle='--')  # Add a horizontal line at y=0
plt.title('Residual Plot')
plt.show()
```





In [27]:
```
y_test
```

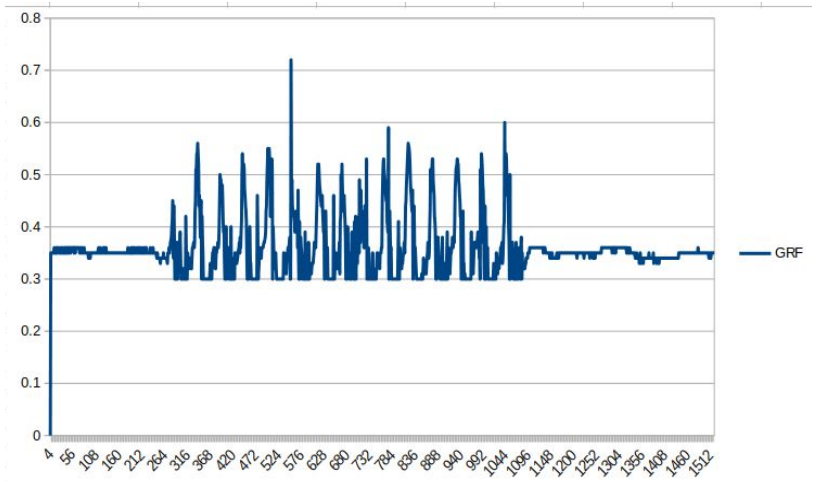
Out[27]:
```
19     45.8912
45     66.9984
139    51.4386
30     29.4586
67     49.9008
16     62.3438
119    56.8726
173    53.1834
109    41.8012
140    61.7978
24     54.6470
161    60.3082
41     39.7892
118    64.3158
15     59.1404
111    33.9602
113    81.1850
82     46.7706
9      40.4454
114    92.2216
18     75.7650
66     24.2132
60     20.1632
168    50.7504
170    26.2616
150    26.0556
117    29.9960
65     47.3428
90     31.7786
55     29.2924
29     25.5966
127    49.9730
144    42.4600
31     49.0030
12     33.9790
42     18.1128
Name: MeanW, dtype: float64
```

In [28]:
```
y_pred
```

Out[28]:
```
array([43.493314, 56.242584, 53.604412, 52.679238, 50.4157  , 58.721954,
       35.25482 , 51.347632, 40.970404, 61.34575 , 57.247208, 42.171472,
       53.39482 , 47.699224, 61.03527 , 48.329322, 54.587196, 57.745946,
       30.678644, 50.826134, 59.84769 , 38.73512 , 28.551538, 51.151138,
       37.808564, 32.676798, 40.313928, 56.34218 , 43.561526, 42.443382,
       36.65938 , 51.264168, 32.608192, 55.404824, 24.057218, 35.903554])
```

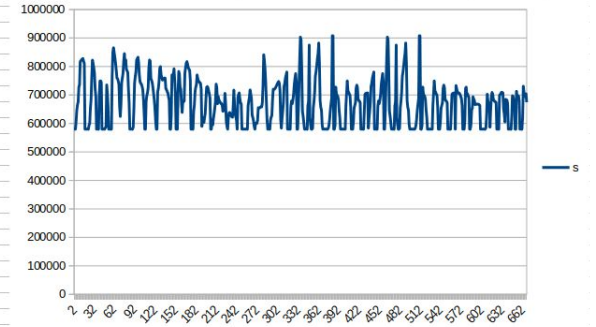
```
In [402]: y_test

Out[402]: 1600    0.319614
          1473    0.587513
          1361    0.452262
          1218    0.831645
          380     0.488315
                    ...
          1587    0.039528
          1759    0.193982
          427     0.012418
          643     0.411664
          1640    0.459575
          Name: w, Length: 354, dtype: float64

In [403]: rf_predictions

Out[403]: array([0.34218258, 0.32442479, 0.3957521 , 0.64929377, 0.43624487,
                 0.0674088 , 0.53569165, 0.4261029 , 0.3735896 , 0.40187954,
                 0.46065797, 0.52671459, 0.51408897, 0.33151276, 0.18068251,
                 0.40452575, 0.65697943, 0.37111781, 0.35730669, 0.37835664,
                 0.40365779, 0.45730076, 0.64442957, 0.22903602, 0.50624471,
                 0.56711399, 0.46544362, 0.27230973, 0.44197671, 0.42987549,
                 0.43517087, 0.45817843, 0.45347794, 0.08925589, 0.2019502 ,
                 0.40363654, 0.65754616, 0.37629708, 0.39867286, 0.29322529,
                 0.30412109, 0.49670718, 0.32090963, 0.61081575, 0.39250259,
                 0.37807665, 0.44614951, 0.4087949 , 0.49120329, 0.24990415,
                 0.26400942, 0.15567699, 0.6146165 , 0.21672168, 0.5270318 ,
                 0.3279495 , 0.08394634, 0.42554853, 0.23605844, 0.17364132,
                 0.44743449, 0.48952682, 0.24849663, 0.49214882, 0.39644399,
                 0.16116722, 0.06118814, 0.39970223, 0.20840012, 0.1376901 ,
                 0.29555973, 0.45381639, 0.55021987, 0.12748868, 0.49305944,
                 0.45345192, 0.47599634, 0.42774478, 0.41222936, 0.40037601,
                 0.43011611, 0.40141262, 0.44874384, 0.44626974, 0.16469868,
                 0.40930333, 0.14471301, 0.35651827, 0.58053494, 0.41658827,
                 0.55492976, 0.54728939, 0.46118039, 0.44265527, 0.23985276,
                 0.5710012 , 0.35094109, 0.60467447, 0.36685228, 0.1837436 ,
                 0.37966204, 0.40201426, 0.14753463, 0.32320421, 0.06607392,
                 0.43550702, 0.51157551, 0.58970041, 0.17284731, 0.49874747,
                 0.41255398, 0.45568671, 0.40047499, 0.45253504, 0.03796614,
                 0.5824629 , 0.27531466, 0.33158094, 0.50343001, 0.32221799,
                 0.32042426, 0.10671278, 0.39377094, 0.36890607, 0.40153515,
                 0.46836141, 0.10376929, 0.38899863, 0.52263765, 0.37373339,
                 0.35625574, 0.08619645, 0.40394585, 0.20827264, 0.53067428,
                 0.37825288, 0.50456775, 0.05684042, 0.34532915, 0.38093022
```

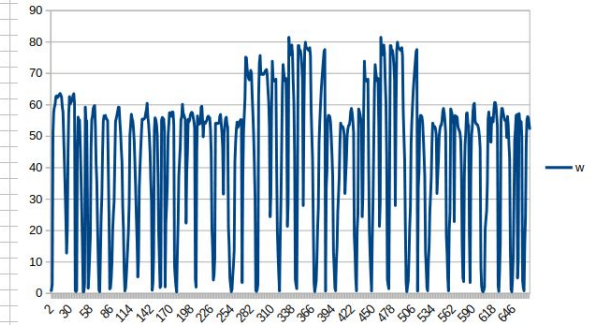
```
int topology[] = {13, 16, 16, 16, 16, 16, 16, 1};
double weights[] = {-0.46993333, 0.8886685, 0.39887175, -0.33392772, 0.6136528, -
double biases[] = {-0.14078854, 0.04950403, -0.06156708, -0.052017972, -0.0298949
int layers = 8;
```



Current Results

matching_values	ax	ay	az	gx	gy	gz	s	w
59.12	415.04	1533.69	-279.3	-54.51	16.04	-18.6	580000	0.602
59.15	96.68	575.68	399.41	-10.06	-9.51	5.12	580000	1.112
59.18	15.14	981.45	185.06	-7.07	-4.88	1.34	580000	2.906
59.46	30.27	962.89	188.96	0.43	1.16	3.05	607998.5	50.824
59.59	23.44	950.2	191.41	5.18	1.71	1.52	646791	57.66
59.64	27.83	957.03	179.69	6.34	1.1	0.79	664911.5	59.224
59.67	32.71	972.66	163.57	6.46	2.99	1.65	676844.75	60.12
59.82	20.02	963.87	154.3	3.11	1.59	4.21	726883.5	62.526
59.85	48.34	965.33	170.9	0.06	2.32	2.38	736141.25	62.848
60.08	97.17	975.59	168.95	14.82	-6.16	35.06	816928.75	62.314
60.11	113.77	1003.91	175.78	9.88	-9.15	38.78	820344.5	62.518
60.14	127.44	989.26	170.9	0.49	-5.67	43.23	823528.75	62.882
60.16	184.57	1045.9	162.6	-4.02	-7.38	39.88	826633.75	63.372
60.19	158.2	992.68	193.85	-13.72	-7.07	44.94	827751.5	63.602
60.24	213.87	929.69	214.84	-9.21	-8.6	52.93	818227.25	63.112
60.27	253.91	977.05	154.3	-12.74	-3.54	53.35	810183	62.52
60.47	376.46	999.02	-125.49	11.65	29.39	134.7	580000	59.372
60.5	76.17	379.88	68.85	27.87	53.9	118.66	580000	57.796
60.55	-522.46	923.83	160.64	-41.71	-12.38	-58.96	580000	49.334
60.58	-98.14	506.35	-66.41	-35.12	-6.77	-65.3	580000	41.242
60.6	-103.52	590.82	-61.52	-9.02	35.49	-71.59	580000	32.708
60.63	-164.06	552.73	80.08	-9.82	77.8	-96.52	580000	23.236
78.33	23.93	969.73	176.76	2.38	-1.83	2.44	593923.5	12.858
78.36	14.16	977.05	178.71	1.71	1.28	2.8	600378.75	20.33
78.46	17.09	962.4	173.34	1.89	2.38	2.13	652787	45.058
78.51	30.27	966.31	191.41	-3.78	0.91	4.63	683792.25	57.506
78.62	114.75	960.45	177.73	-4.76	4.09	16.34	761653.75	62.608
78.82	160.64	985.35	229.49	-15.85	-7.26	57.74	822087.25	60.306
78.85	183.11	1026.37	265.14	-18.11	-12.99	65	810550.25	60.71
78.87	201.17	1101.07	210.45	-20.55	-11.28	71.28	795586.75	61.218
78.9	260.25	1050.29	213.38	-24.76	-11.34	79.82	775936.75	61.8
78.95	312.01	1239.26	201.17	-30.43	-5.06	119.21	719263.25	63.08
78.98	353.52	1438.96	65.92	-19.27	6.22	158.66	695402.75	63.53
79.11	-692.38	-244.14	-171.87	75	107.38	154.39	580000	60.49
25.5	481.93	1013.67	326.17	-19.51	77.8	54.57	580000	0.96
33.67	2068.85	3999.88	-729.49	-271.77	-71.59	131.16	580000	0.58
33.7	-460.45	546.39	1308.59	-118.78	-59.09	82.87	580000	1.004
33.88	7.81	993.65	177.73	6.16	1.83	11.46	693844	47.17
34.08	103.03	949.71	190.43	-31.83	0.18	50	748643.5	56.086
34.11	90.33	994.14	289.06	-26.65	-6.59	62.13	749360.5	55.006
34.14	53.71	1079.1	376.46	-23.41	-12.2	71.95	744540.5	55.124
34.26	-691.41	1993.65	-9.77	-62.56	-6.16	267.2	580000	47.27
34.29	-952.15	-1063.48	-383.3	-71.83	3.23	253.11	580000	37.87
34.31	-1109.86	-318.36	-709.47	-87.68	64.76	-58.05	580000	26.42
34.34	-1103.52	383.79	-271.48	-4.88	29.94	-26.1	580000	15.226
34.44	-399.9	791.5	-188.96	60.73	14.39	-263.72	580000	0.524





In [312]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

In [313]:
```
# Train and evaluate Random Forest Regression
rf_model = RandomForestRegressor(n_estimators=500, random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)
print("Random Forest Regression:")
print("Mean Squared Error:", rf_mse)
print("R-squared Score:", rf_r2)
```

```
Random Forest Regression:
Mean Squared Error: 0.010496562071082351
R-squared Score: 0.8415530717727713
```

In [314]:
```
y_test
```

Out[314]:
```
403    0.662806
277    0.847370
47     0.033687
323    0.892615
231    0.662114
         ...
96     0.670832
321    0.570165
60     0.728451
477    0.891776
236    0.696789
Name: w, Length: 117, dtype: float64
```

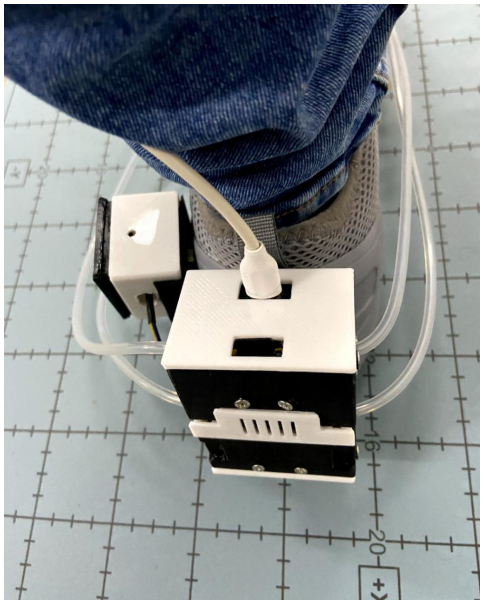
In [315]:
```
rf_predictions
```

Out[315]:
```
array([0.65197782, 0.72563902, 0.23800242, 0.82300929, 0.69151509,
       0.3722453 , 0.54223379, 0.79990353, 0.04741457, 0.19062055,
       0.26979852, 0.02085987, 0.73740178, 0.56131158, 0.58952611,
       0.49212038, 0.23738982, 0.64143547, 0.80719165, 0.54143655,
       0.69342741, 0.67032364, 0.66156528, 0.68629716, 0.55363601,
       0.05536216, 0.66254231, 0.80880089, 0.65747078, 0.20522233,
       0.82435001, 0.07406115, 0.80528333, 0.71133766, 0.62805335,
```

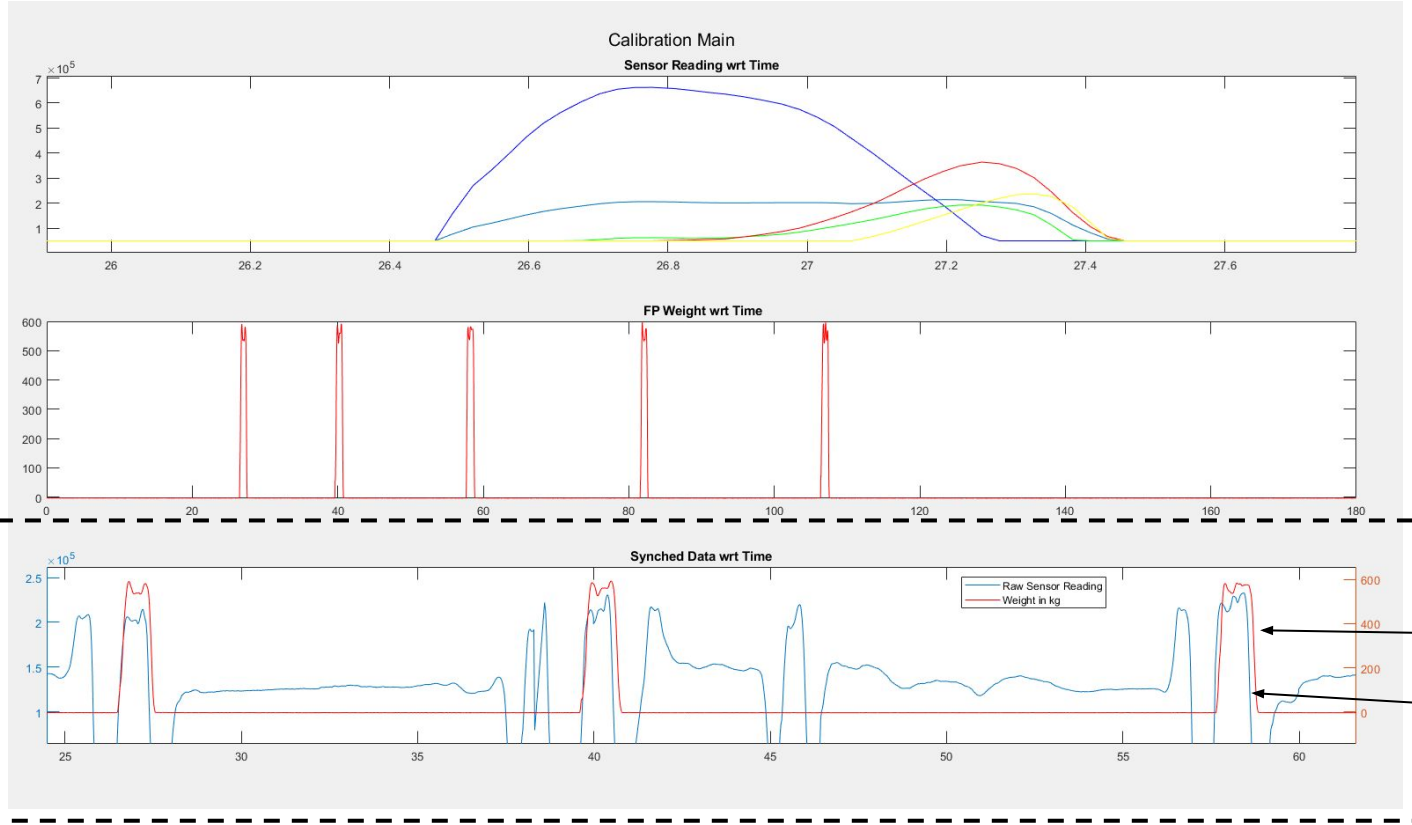
```
print("Y Test:")
print(y_test)
print("Y Predictions:")
print(y_pred)
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
```

```
4/4 [==============================] - 0s 2ms/step
Y Test:
12     0.776340
214    0.661793
497    0.043443
347    0.948506
285    0.005927
         ...
400    0.375426
443    0.238405
528    0.375426
362    0.713707
258    0.639244
Name: w, Length: 117, dtype: float64
Y Predictions:
[ 0.7423792   0.7076608   0.41565233  0.9611623   0.02230274  0.04866318
  0.6265865   0.7469257   0.6119231   0.61594284  0.11967886  0.68643093
  0.69094676  0.27540553  0.15812892  0.25540107  0.7438188   0.7291359
  0.90294373  0.83091     0.70867646  0.46038857  0.9204968   0.73718035
  1.1232926   0.7502847   0.70960355  0.62139326  0.61396736  0.06573115
  0.32029995  0.02612858  0.71888757  0.67705834  0.680987    0.34453997
  0.6339353   0.68479437  0.33306953  0.7912621   0.6856028   0.6172696
  0.702922   -0.01108614  0.7659545   0.02692869  0.67251825  0.30137625
  0.7201616   0.3908216   0.54321986  0.2771597   0.3203187   0.08731677
  0.6538501   0.6209514   0.7713186   0.69717515  0.62924075  0.7415739
  0.7282168   0.40948823  0.35213017  0.48460254  0.46932232  0.00771803
  0.11993184  0.76335675  0.67136526  0.03917685  0.55177796  0.64043593
  0.6115025   0.56438637  0.53564334  0.74387956  0.6706244   0.6281302
  0.19790816  0.63375294  0.10577557  0.41565233  0.06681111  0.15891358
  0.63578594  0.5372459   0.02612856  0.59095776  0.7228956   0.63689154
  0.11993184  0.3484548   0.71163535  0.6602269   0.6375067   0.6334096
  0.65801334  0.6817095   0.5583559   0.6383089   0.7343513   0.33306953
  0.6884522   0.21336523  0.74261534  0.60786015  0.55807525  0.5189326
  0.65772724  0.54202664  0.39455482  0.63697445  0.53299314  0.18017152
  0.5329931   0.7502847   0.66889536]
Mean Squared Error: 0.024757749287191412
R-squared Score: 0.6734318687919288
```

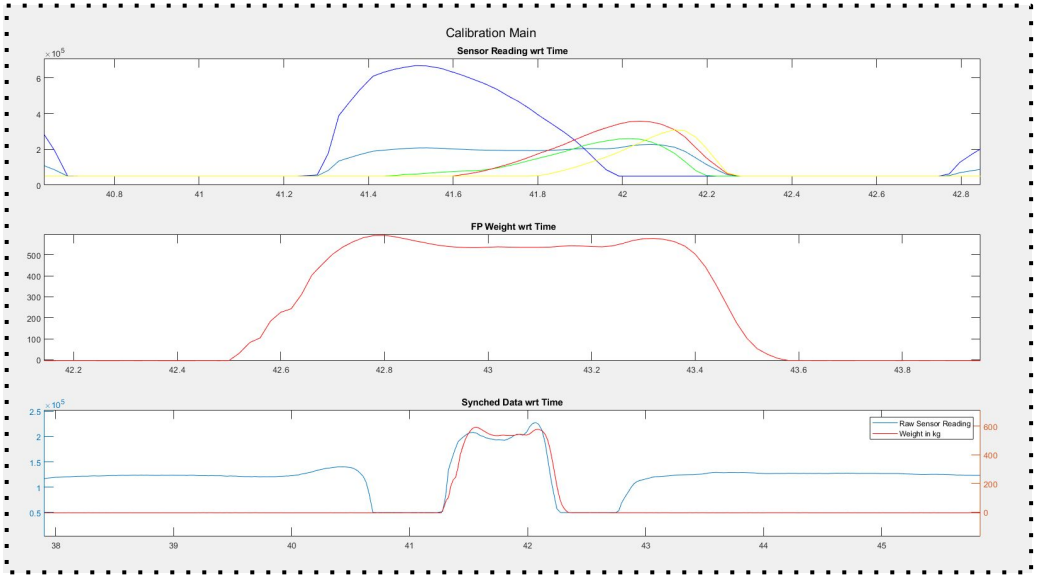
Data Collection with Force Plates

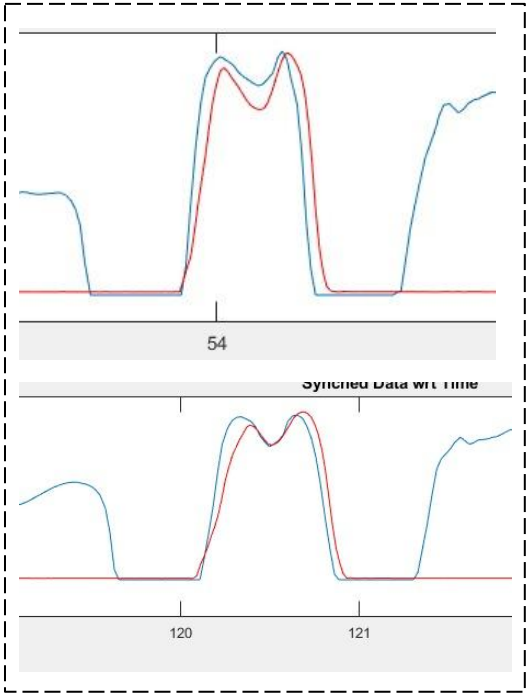


Data Synching & Calibration



Similar Force Profiles obtained from Developed Smart Shoe and Force Plates





Force Profiles

- Force Plates
- Smart Shoe

Matching m curves profiles for Smart Shoe and Force Plates Data

Validated Data

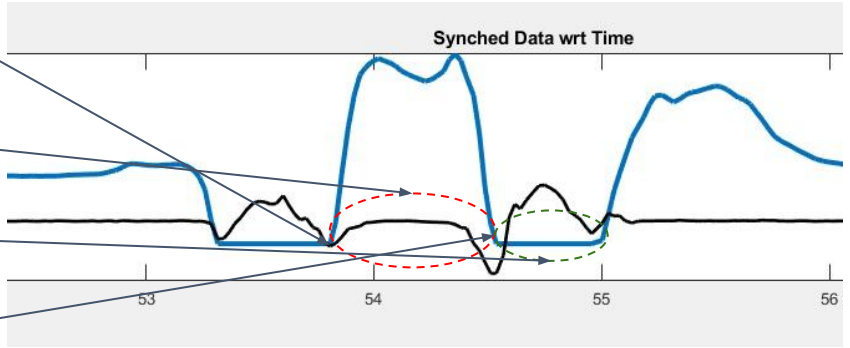
- IMU GyroZ
- Smart Shoe

Heel Strike

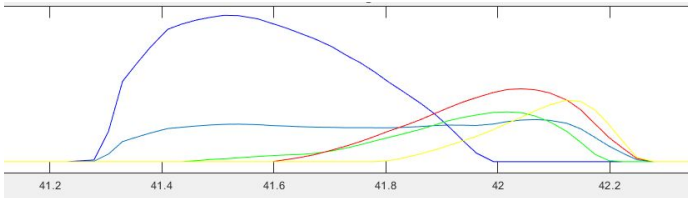
Stance Phase

Swing Phase

Toe Off



More Information from Individual Sensor Readings about Comparative Force distribution, Timing of GE, profiles etc



Sensor Readings

- Heel
- Meta45
- Meta12
- Toe
- Average m Curve