ECE 276a PR 1 - Orientation Tracking

Kenneth Vuong
PID: A15552808
February 4th, 2025

***Abstract*—This project is an exercise in learning orientation tracking, transformations between coordinate frames, and representations of motion for a rotating body.**

I. Introduction

Object orientation tracking is a method to estimate the different orientations between coordinate frames. This can be extended further to coordinate transforms, which describe both rotation and translation between two bodies. Knowing how to describe a body's orientation and translation allows for prediction and control. Use cases are seen in many fields such as robotics, computer graphics, computer vision, etc. In this report we will estimate the orientation of a moving body using Inertial Measurement Unit (IMU) measurements and further optimize the prediction using projected gradient descent. After obtaining the body's orientation, we will generate a panorama from a series of pictures taken by the body during the measurement.

A. Experiment Setup

The experiment setup is as follows. There is an IMU and camera on a wooden board which is then picked up and rotated, there at first is no motion for a few seconds. The camera axis is aligned with the IMU x axis.

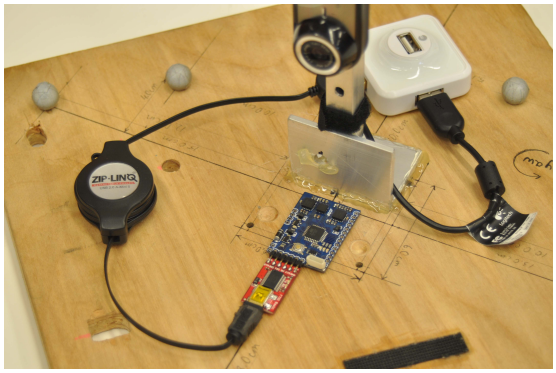


Fig. 1. Experiment Setup

The accelerometer found in the IMU is Analog Devices ADXL335. The yaw gyroscope is STMicroelectronics LY530AL while the pitch and roll gyroscopes are STMicroelectronics LPR530AL. There is also a VICON motion capture system that externally retrieves a ground-truth rotation dataset that we will compare our estimations with to access accuracy of our motion prediction.

B. Problem Formulation

The body referernce frame conventions are as follows.

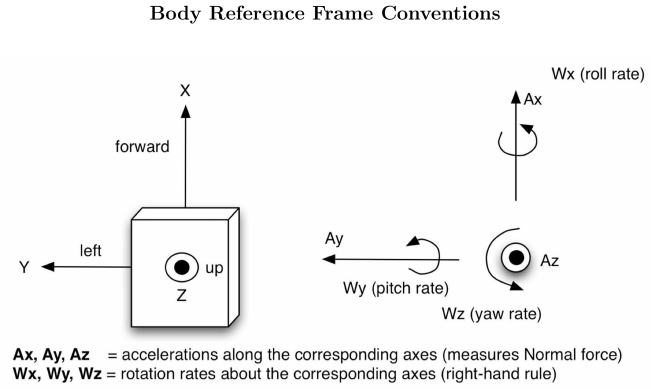


Fig. 2. Body Reference Frame Convention

1) Data Preprocessing: The VICON data reports the rotation of the body across time through rotation matrices. The IMU data comes as ADC values that are unusable before preprocessing. Using the VICON data, we will calibrate our IMU such that the values become usable.

2) Orientation Tracking: The objective of the first part of the project is to estimate the orientation of the body using IMU angular velocity $\boldsymbol{\omega}_t$ and linear acceleration $\boldsymbol{a}_t$. We assume no translation and only rotation of the body. To estimate orientation, we will predict an initial motion model f with quaternions $\boldsymbol{q}_{1:T}$ and use that to estimate an observation model h. From there, we will implement a cost function and utilize projected gradient descent to achieve our trajectory.

3) Panorama Generation: We use the images taken from the camera to stitch together a panorama of the scene. We will assume no translation and only rotation. To do this, we will utilize the IMU prediction or VICON data body motion estimates to determine where the camera was pointed over it's recording. From there, we will use coordinate transforms to project our image onto a sphere. Then to obtain our panorama we will map each angle of the sphere to a pixel in the panorama.

II. Problem Formulation

A. Data Preprocessing

The data directly from the IMU is uncalibrated and exhibits a bias that needs to be corrected for. For each axis, the raw A/D value is converted to physical units through:

$$value = (raw - bias) \times scalefactor \tag{1}$$

With the scale factor being:

$$scalefactor = Vref/1023/sensitivity \tag{2}$$

The sensitivity depends on device used for each axis.

B. Orientation Tracking

We will represent the orientation of our rotating body using unit quaternions $\boldsymbol{q}_{1:T}$. In this project, we assume that time $t = 0$ is our starting time so $\boldsymbol{q}_0 = [1, 0, 0, 0]$. To predict the orientation of our body from the starting orientation, we can use a basic angular velocity calculation where angular position is a result of speed multiplied by time.

Given τ_t, which represents the time between steps t and $t+1$, and $\boldsymbol{\omega}_t$, which represents the angular velocity at time t, we are able to formulate a quaternion representation of this same kinematic notion. This leads to the motion model representing said quaternion kinematics:

$$\boldsymbol{q}_{t+1} = f(\boldsymbol{q}_{t,t}\,\boldsymbol{\omega}_t) := \boldsymbol{q}_t exp([0,_t\,\boldsymbol{\omega}_t/2]) \tag{3}$$

Equation 3 describes the motion model to predict the next quaternion at a given time. Using this, we can determine an initial model $\boldsymbol{q}_{1:T}$ that will describe the orientation of our body. However, to optimize the same trajectory, we leverage our acceleration measurements to arrive at a observation model.

Given our assumption that the body is undergoing a pure rotation, the measured acceleration in the the IMU frame should agree with gravity acceleration after it is transformed to the IMU frame. This is done with the observation model:

$$[0, \mathbf{a}_t] = h(\mathbf{q}_t) := \mathbf{q}_t^{-1} \circ [0, 0, 0, -g] \circ \mathbf{q}_t. \tag{4}$$

The above equation transforms the world acceleration to the body (IMU) frame through the quaternion coordinate transform which maps from world to body frame. Now knowing both a motion model and an observation model, we are able to formulate our cost function.

$$c(\mathbf{q}_{1:T}) := \frac{1}{2}\sum_{t=0}^{T-1} \|2\log\left(\mathbf{q}_{t+1}^{-1} \circ f(\mathbf{q}_t, \tau_t\boldsymbol{\omega}_t)\right)\|_2^2 + \frac{1}{2}\sum_{t=1}^{T} \|[0, \mathbf{a}_t] - h(\mathbf{q}_t)\|_2^2 \tag{5}$$

The above cost function optimizes for $\boldsymbol{q}_{1:T}$. The first term measures the difference between the estimated orientation and the motion model prediction. In the first term, $\mathbf{q}_{t+1} \circ f(\mathbf{q}_t, \tau_t\boldsymbol{\omega}_t)$ represent the error between the relative rotation $\mathbf{q}_{t+1}$ and predicted orientation $f(\mathbf{q}_t, \tau_t\boldsymbol{\omega}_t)$. The second term measures the difference between measured acceleration $[0, \mathbf{a}_t]$ and the observation model prediction $h(\mathbf{q}_t)$.

We enfore the constraint that quaternions remain in unit form such that $\boldsymbol{q}_t \in \mathbb{H}_*$. Therefore, we constrain our optimization problem such that:

$$\min_{\mathbf{q}_{1:T}} c(\mathbf{q}_{1:T}) \text{s.t. } \|\mathbf{q}_t\|_2 = 1, \quad \forall t \in \{1, 2, \ldots, T\} \tag{6}$$

To perform our update step, we simply subtract the gradient of our cost function from our optimization variable. Here we leverage the library JAX to numerically compute the gradient of our cost function with respect to $\boldsymbol{q}_{1:T}$. We then update our quaternion trajectory:

$$\mathbf{q}_{1:T}^{(k+1)} = \prod_{\mathbb{H}_*}\left(\mathbf{q}_{1:T}^{(k)} - \alpha^{(k)}\nabla c\left(\mathbf{q}_{1:T}^{(k)}\right)\right) \tag{7}$$

Where the projection function maintains our unit quaternion constraint:

$$\prod_{\mathbb{H}_*}(\mathbf{q}) = \frac{\mathbf{q}}{\|\mathbf{q}\|_2} \tag{8}$$

We run gradient descent until the cost of each iteration plateaus, then we retrieve our optimized trajectory $\boldsymbol{q}_{1:T}$.

C. Panorama Generation

To generate our panorama, we need to use the rotation matrix data captured from the VICON or our optimized quaternion trajectory. Given the rotation matrix vector describing our trajectory over time, we are able to overlay images on top of eachother as the orientation moves through spherical space. We first map our image pixels to a sphere. We describe our spherical coordinates using the order (r, θ, ϕ) where r represents the distance, θ represents the inclination, and ϕ represents the elevation. Imagine that we are at the center of the sphere and project our 2D image onto the sphere. We are given that our camera FOV is 60 degrees longitudinally and 45 degrees laterally. We assume that the radius r of the sphere is 1 meter to simplify calculations. Here, we are able to describe each pixel coordinate x_{img} and y_{img} in spherical coordinates through the formula:

$$\theta(x) = \frac{x_{img}}{w_{img}}(c_{longitude}) - \frac{(c_{longitude})}{2} \tag{9}$$

$$\phi(y) = \frac{y_{img}}{h_{img}}(c_{latitude}) - \frac{(c_{latitude})}{2} \tag{10}$$

Where width $w_{img} = 320$, height $h_{img} = 240$, longitudinal field of view $c_{longitude} = 60$, and latitudinal field of view $c_{latitude} = 45$. Once we obtain our corresponding spherical coordinates, we can transform them to cartesian coordinates through:

$$x = r\sin\theta\cos\phi \tag{11}$$

$$y = r\sin\theta\sin\phi \tag{12}$$

$$z = r\cos\theta \tag{13}$$

This maps our pixel from spherical coordinates to cartesian coordinates. From there, we are able to apply the rotation matrix $R^{3\times3}$ such that:

$$\boldsymbol{y}_{world} =_{world} R_{body}\boldsymbol{y}_{body} \tag{14}$$

Doing so for every point in every image will yield a sphere of mapped 2D image to sphere pixels. From there, we are able to

convert the spherical image back into a 2D plane by projecting it onto our panorama through:

$$x_{pan} = \frac{\theta + \pi}{2\pi} \times w_{pan} \tag{15}$$

$$y_{pan} = \frac{\phi}{\pi} \times h_{pan} \tag{16}$$

Where x_{pan} and y_{pan} are pixel coordinates of the panorama.

III. Technical Approach

A. Data Preprocessing

We are given VICON, IMU, and Camera .pickle files to read and interpret data from. I construct vectors corresponding to camera, IMU, and VICON data. Following the equations laid out in section 2, I calculated the calibrated IMU values. This was done with the following sensitivities: To calculate for

Parameter	Device	Sensitivity	Units
Accelerometer	ADXL335	300	mV/g
Yaw Gyroscope	LY530AL	3.33	mV/deg/sec
Pitch/Roll Gyroscope	LPR530AL	3.33	mV/deg/sec

TABLE I
IMU Sensitivities

bias, we know that in the first few seconds of every dataset the body is at rest. For angular velocity, this means that the measured angular velocity should be zero for all axes. For linear acceleration, this means that the measured acceleration $[x, y, z]$ should be $[0, 0, 1]$ in gravity units g. I took the average value for each measured quantity over the first few seconds then subtracted it from the entire dataset to eliminate bias. Then, I multiplied by the scaling factor to find the true value following equations 1 and 2. Afterwards, I added 1 to the z acceleration vector to account for gravity. I construct a time vector $\boldsymbol{imut} \in \mathbb{R}^{n\times 1}$, acceleration vector $\boldsymbol{a} \in \mathbb{R}^{n\times 3}$, and an angular velocity vector $\boldsymbol{\omega} \in \mathbb{R}^{n\times 3}$ to keep track of the values.

B. Orientation Tracking

To store values efficiently and perform gradient descent with later, I use the JAX library. I also use Matplotlib to visualize data. In many operations reshaping had to be done to accommodate vectorized quaternion operations.

1) Quaternion Operations: There are several quaternion operations utilized in the calculation of the motion and observation models. All quaternion operations are done using vectorization and broadcasting with JAX arrays. To accommodate for a vectorized approach, each operation was implemented into a function that can accept quaternions as both individual vectors and tensors. This helps in the initial calculation of the motion model. Some special cases which arose were the quaternion log and quaternion exponential operations. Since these operations divided by magnitude, a small offset was added whenever the magnitude of a quaternion or quaternion vector component was nearing zero. This helped smooth out the graph for the motion model. Implementation of the quaternion operations can be found in the attached code.

2) Initial Motion Model Calculation: To calculate the initial quaternion vector $\boldsymbol{q}_{1:T}$, I start by initializing $\boldsymbol{q}_0 = [1, 0, 0, 0]$ and a vector $\boldsymbol{q} \in \mathbb{R}^{n\times 4}$, filling in the first entry with $\boldsymbol{q}_0 = [1, 0, 0, 0]$. I then run a for loop to sequentially calculate each step from time $t = 1, 2, \ldots, T$ which calculates an initial motion model for $\boldsymbol{q}_{1:T}$ following equation 3.

3) Vectorized f and h functions : I vectorized the f and h functions to be able to input tensors. These were of the forms of equation 3 and 4 respectively. The values passed were $\boldsymbol{\tau} \in \mathbb{R}^{n-1\times 1}$, $\boldsymbol{q} \in \mathbb{R}^{n\times 4}$, and $\boldsymbol{\omega} \in \mathbb{R}^{n-1\times 3}$. The reason that $\boldsymbol{\tau}$ and $\boldsymbol{\omega}$ are of size $n-1$ is because we are optimizing for $\boldsymbol{q_{1:T}}$, using the time difference and angular velocity between the time step before the current one.

4) Cost Function Implementation: The cost function was implemented according to equation 5. The cost function also took in vectorized components and did not feature any for loops. This allowed for faster processing. To compute the cost function, it accepted $\boldsymbol{\tau} \in \mathbb{R}^{n-1\times 1}$, $\boldsymbol{q} \in \mathbb{R}^{n\times 4}$, $\boldsymbol{\omega} \in \mathbb{R}^{n-1\times 3}$, and $\boldsymbol{a} \in \mathbb{R}^{n\times 3}$. It followed the quaternion operations, and utilized JAX's linalg.norm function to calculate the magnitude of the error between predicted and observed results.

5) Gradient Descent: Gradient descent was implemented according to equations 7 and 8 respecting the constraint in equation 6. JAX was leveraged to find the gradient of the cost function with respect to $\boldsymbol{q_{1:T}}$. A for loop was implemented to handle the gradient calculation as well as update the quaternion trajectory and normalize the each quaternion. The optimization was stopped arbitrarily after the cost seemed to plateau. After experimentation, it was seen that the cost usually plateaus around ten iterations.

C. Panorama

Several functions were implemented that simplified the process of generating a panorama from a series of photos. All functions were designed with vectorization in mind to make the calculations more efficient. After reading in camera data, data structures were made which paired with corresponding functions to make processing more efficient.

Firstly, a new vector *synced-rots* was implemented which looked at the closest time values between the larger *rots* vector and the camera's time vector. The VICON rotation vector and predicted rotation vectors were chosen to fit this purpose. The following functions were used to implement panorama generation:

- sph2cart() - generated a numpy array which converted spherical coordinates to cartesian coordinates.
- cart2sphvec(rotations) - a function which took the entire vectorized rotation matrix array in cartesian coordinates and output them to spherical coordinates.
- map-spherical-to-panorama(rotated-coords, cam) - mapped spherical pixels to a panorama.
- visualized-superimposed(rotated-coords,cam) - a fun function that shows the spherical image.

Each pixel in the original image for all images were mapped to a spherical coordinate. Then, the corresponding rotation matrices converted them to rotated cartesian coordinates. They

were then converted back into spherical coordinates once the sphere has been ”painted”. Then, the image of the sphere was projected back onto a panorama following equations 15 and 16.

IV. Results

The results for training set 1 and both test sets are reported here. Other training sets can be found in the appendix.

A. Dataset 1

1) Data Preprocessing: The original ADC values can be seen in Fig. 3.

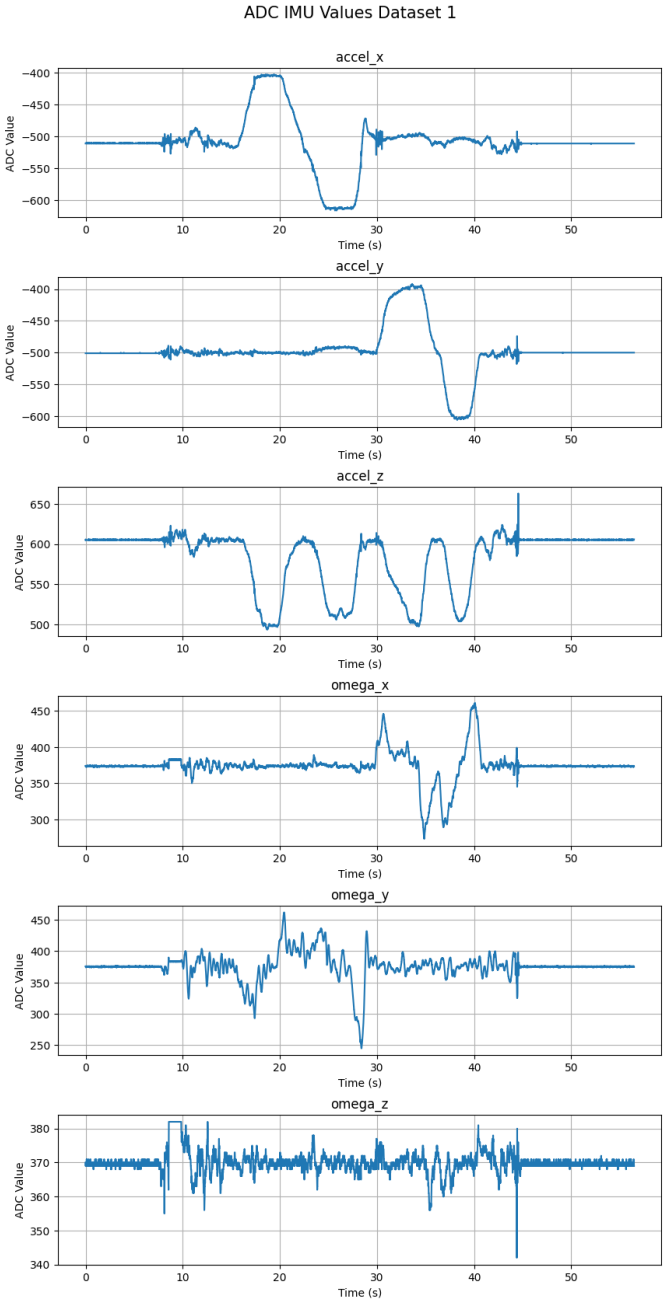


Fig. 3. Raw IMU Values Dataset 1

The calibrated IMU values can be seen in Fig. 4. The bias has been removed and the gravity measurement normalized to agree with gravity.

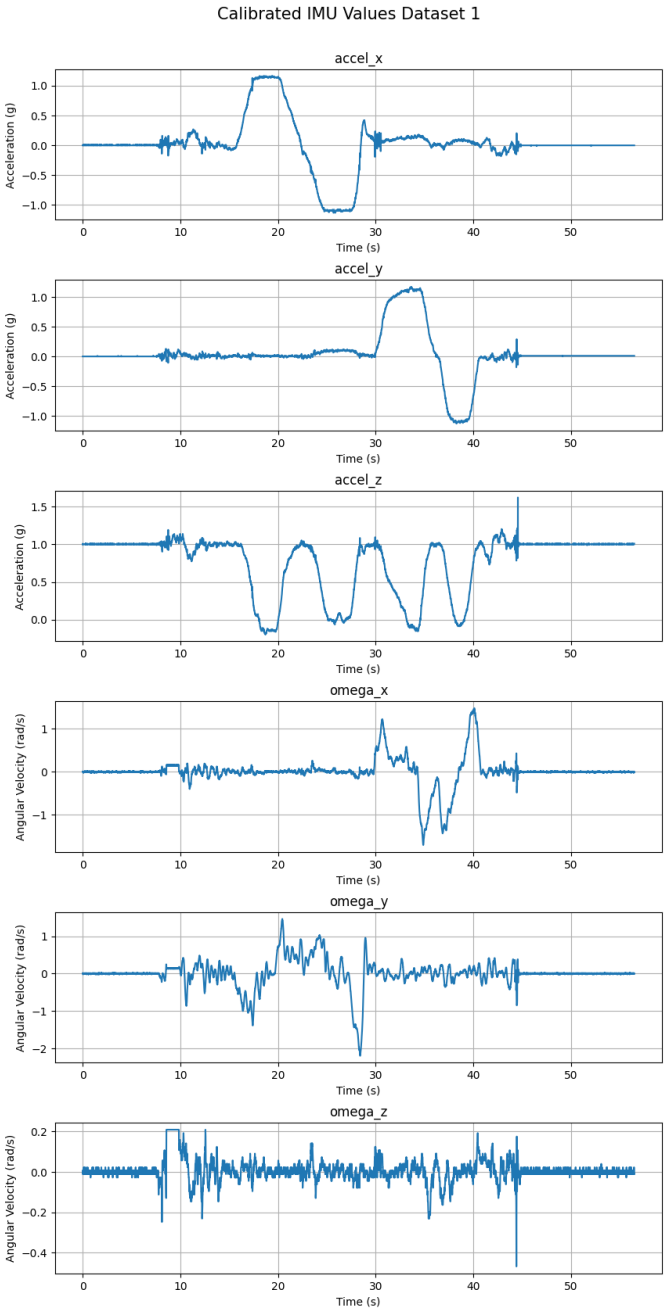


Fig. 4. Calibrated IMU Values Dataset 1

The VICON data can be seen in Fig. 5.

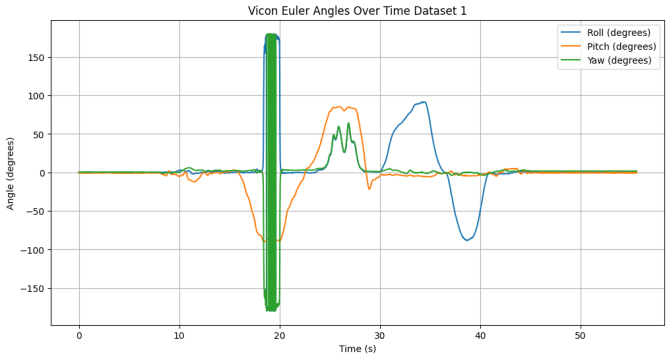


Fig. 5. VICON Values Dataset 1

2) Orientation Tracking: After running the initial motion model, we can see that the initial quaternion trajectory estimate does follow the VICON data quite well, though there is drift due to sensor inaccuracies. We run our gradient descent function with a learning rate of $\alpha = 0.001$ for $n = 10$ iterations. The cost data can be seen in Fig. 7. Here, the

```
Cost Iter 1: 14373.24162485939
Cost Iter 2: 6055.010721878291
Cost Iter 3: 2990.0810972933136
Cost Iter 4: 1867.024966487084
Cost Iter 5: 1454.721879446949
Cost Iter 6: 1302.6376849725461
Cost Iter 7: 1246.206510149814
Cost Iter 8: 1225.1295262519109
Cost Iter 9: 1217.2012125847905
Cost Iter 10: 1214.1960426111534
Cost Iter 11: 1213.04753262951
Cost Iter 12: 1212.6046574392815
Cost Iter 13: 1212.43220499179
Cost Iter 14: 1212.3643232906536
Cost Iter 15: 1212.33727306442
```

Fig. 7. Dataset 1 Gradient Descent Cost

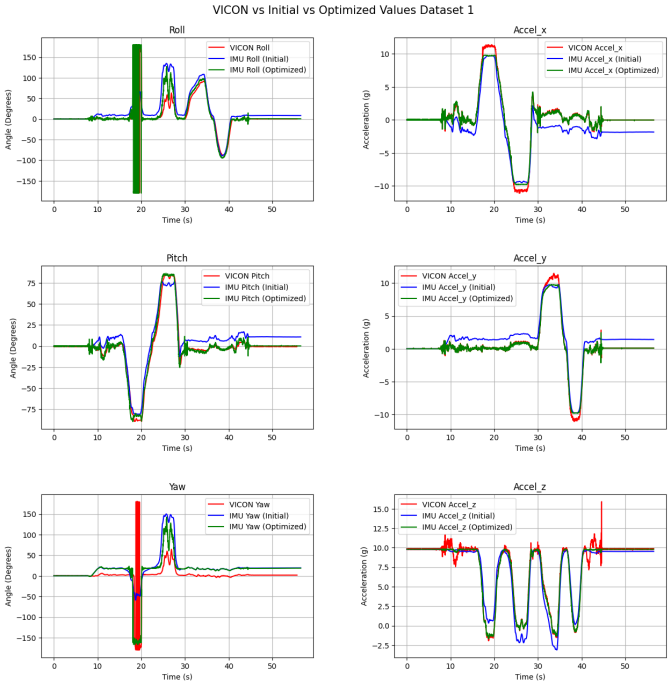


Fig. 6. VICON vs. Initial vs Optimized Values Dataset 1

intial cost is 14373.24 while the final cost is 1212.33. It is observable that the optimized estimate follows the ground truth data more closely than the initial estimate. However, there is noise in the data. This is due to the cost function accounting for noise in the original acceleration estimates. An improvement to make would be to run a post-processing algorithm to smoothen out noisy data and tune it to fit VICON data.

3) Panorama: The spherical mapping of all images superimposed upon each other can be seen in Fig. 8. It is observable that not the whole sphere has been filled in, this is simply due to the body not being rotated a full 360 degrees.

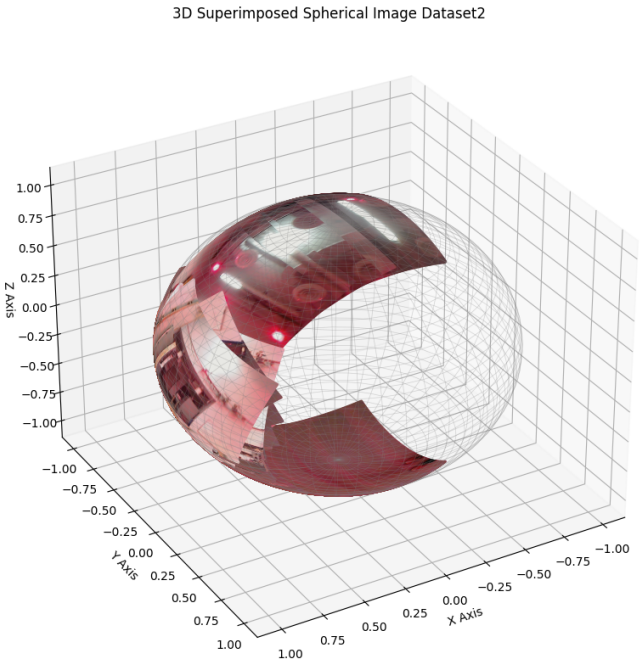


Fig. 8. Spherical Superimposed Image

The panorama can be seen in Fig. 9. The black parts of the panorama correspond to areas of the sphere not mapped to, meaning the body was not rotated to those areas. There is warping on the top and bottom of the image similar to a world map, this is expected.

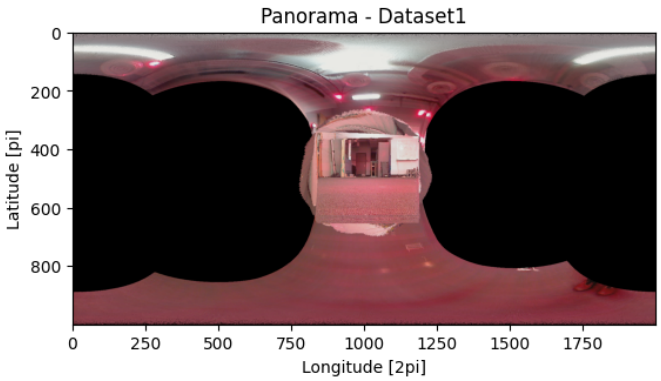


Fig. 9. Panorama Dataset 1

B. *Test Set 1 - Dataset 10*

The raw IMU values for dataset 10 is seen in Fig. 10.

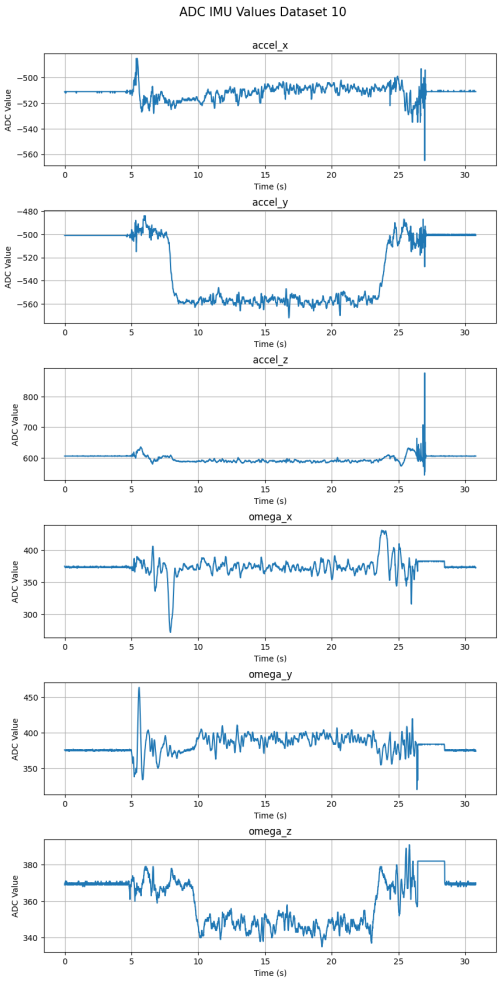


Fig. 10. Raw IMU Values Dataset 10

The calibrated values can be seen in Fig. 11.

The bias term is removed, corresponding to an effective calibration. There seems to be more noise in this dataset, particularly in the angular velocity measurements, there is also a large peak in acceleration in the z-direction around second 28.

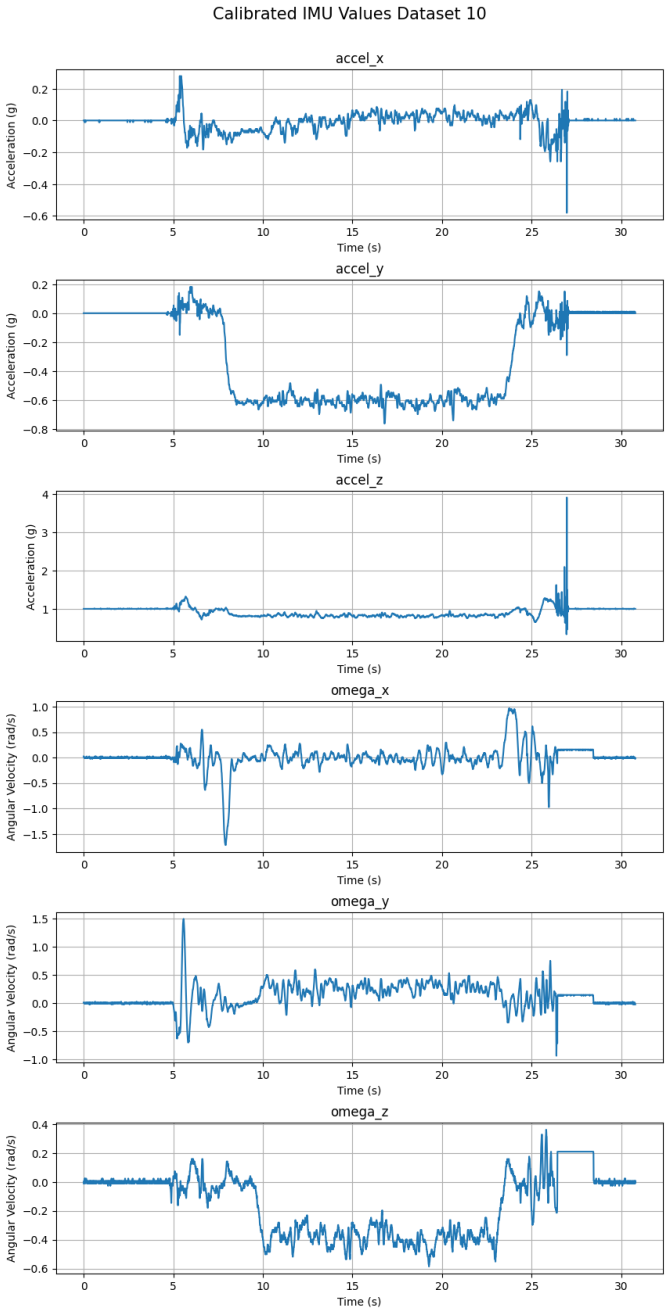


Fig. 11. Calibrated IMU Values Dataset 10

The iniital motion model estimate for the euler angle orientation of the body is seen in Fig 12. There is a flip in signage around the 17 second mark. This can be due to a quaternion operation returning the correct amplitude but wrong direction. This bug can be seen whenever there is lots of noise in the dataset. To correct this, my quaternion operations should handle edge cases better and should check for large jumps in a single axis. Post processing can optimize this orientation trajectory as well.

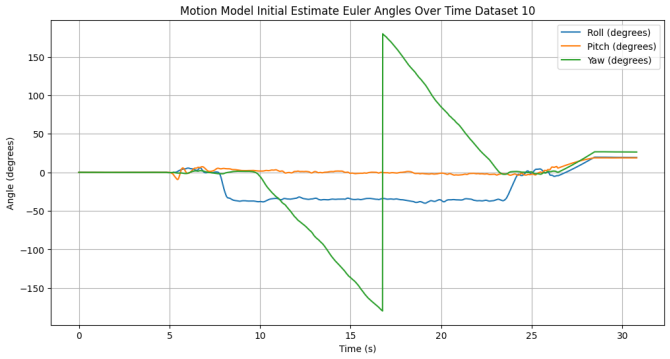


Fig. 12. Initial Motion Model Estimate Dataset 10

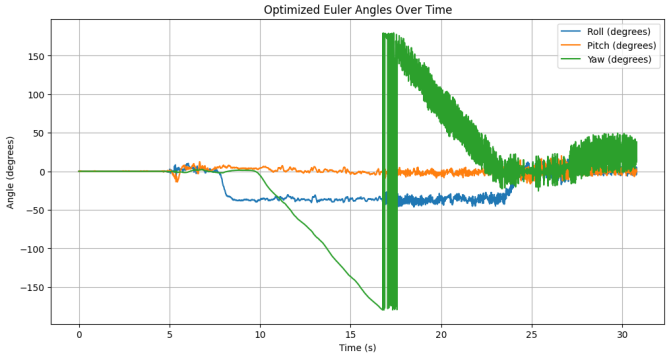


Fig. 13. Optimized Motion Model Estimate Dataset 10

A comparison between initial and predicted trajectories can

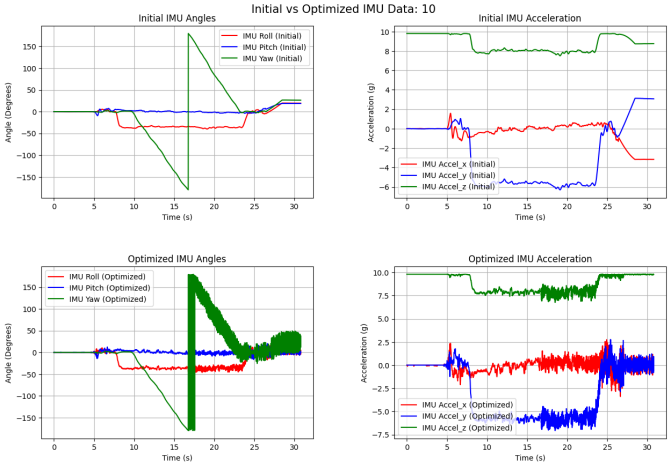


Fig. 14. Initial vs. Optimized Values Dataset 11

be seen in Fig. 14. The optimized quaternion trajectory added a lot of noise right around the 17 second mark where the unexpected sign change in the yaw direction occurred. This is likely due to a missed edge case handling. The noise contributed likely is due to this missed edge case, and does not correct itself after many iterations. This can be fixed by applying a more robust motion model with quaternion operations which handle more edge cases. This was run with a learning rate $\alpha = 0.001$ and $n = 15$ iterations. The cost is shown to increase due to the noise in Fig. 15. The generated

```
Cost Iter 1: 4912.665239211908
Cost Iter 2: 30217.24123988682
Cost Iter 3: 28871.694310210776
Cost Iter 4: 28178.593959476548
Cost Iter 5: 27770.92677279954
Cost Iter 6: 27436.61036340341
Cost Iter 7: 27079.56257902424
Cost Iter 8: 26621.438191156354
Cost Iter 9: 26194.789779544477
Cost Iter 10: 25667.694395656286
Cost Iter 11: 25148.318721342188
Cost Iter 12: 24506.9093736177
Cost Iter 13: 23920.988284588933
Cost Iter 14: 23346.5616947995
Cost Iter 15: 22749.837879493632
```

Fig. 15. Dataset 10 Gradient Descent Cost

panorama for dataset 10 is seen in Fig. 16. We can see that the panorama captures only a small segment of the panorama is reconstructed. It is estimated from the IMU data that the pitch did not vary much, however I suspect that this small range is due to an improper rotation matrix calculation. This likely is due to a bug in the representation, i.e. my rotation data is not represented in the same way as the VICON data.

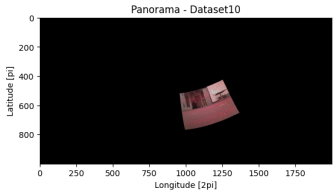


Fig. 16. Panorama Dataset 10

C. Test Set 2 - Dataset 11

Raw and calibrated IMU values can be seen in Fig. 17 and Fig. 18. It appears that calibration is effective and that the dataset is noisy.

For the same reason as dataset 10, there are points in the initial motion model where one axis will spike or flip signage. The optimized values also introduce noise into the motion estimate. This noise contributes to a higher cost at the next iteration of gradient descent. This can be fixed by handling edge cases of the quaternion operations more robustly. Running gradient descent with learning rate $\alpha = 0.001$ and $n = 15$, we get the following cost. Similar to dataset 10, the rotation matrix passed to the panorama algorithm is improperly calculated, leading to a squished image.

V. Conclusion

The algorithm implemented to estimate the orientation of a rotating body from IMU measurements works however outputs noisy data unexpectedly when the input data is noisy. The calibrated IMU values are accurate and remove bias from the raw ADC values. In the datasets, the observation model works as expected. The motion model works with most data, however

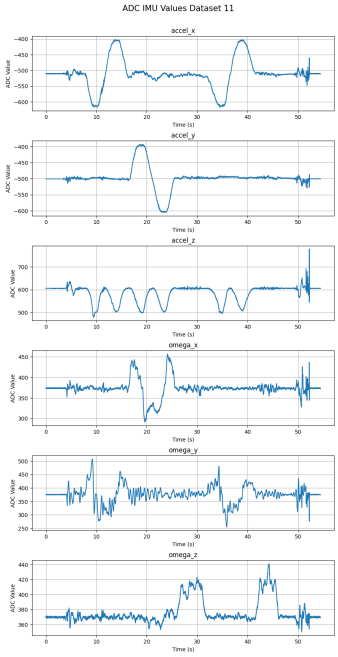


Fig. 17. Raw IMU Values Dataset 11

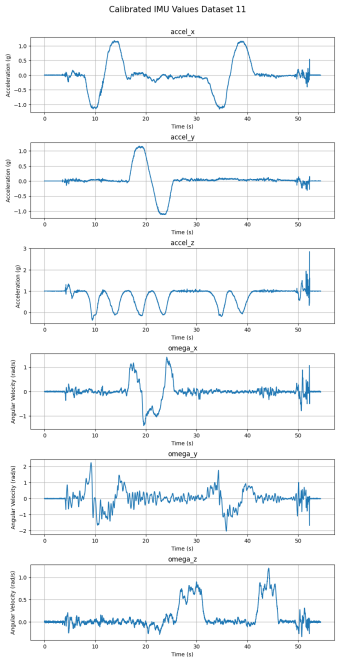


Fig. 18. Calibrated IMU Values Dataset 11

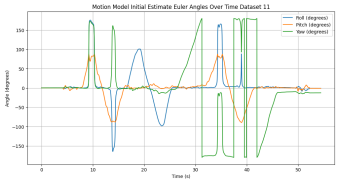


Fig. 19. Initial Motion Model Estimate Dataset 11

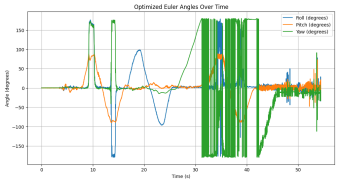


Fig. 20. Optimized Motion Model Estimate Dataset 11

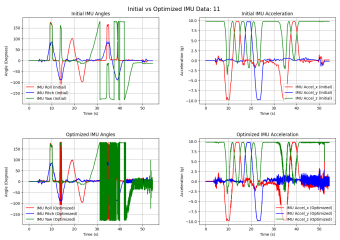


Fig. 21. Initial vs Optimized Values Dataset 11

```
Cost Iter 1: 2509.36385090154
Cost Iter 2: 42568.70206835303
Cost Iter 3: 43475.152452587485
Cost Iter 4: 40752.83751704784
Cost Iter 5: 38885.33729323606
Cost Iter 6: 37583.7767242748
Cost Iter 7: 36546.153627657666
Cost Iter 8: 35368.00029565184
Cost Iter 9: 34408.79575802392
Cost Iter 10: 33677.98465578538
Cost Iter 11: 32730.1206016325
Cost Iter 12: 31956.20789422492
Cost Iter 13: 31180.820223306135
Cost Iter 14: 30273.133338769163
Cost Iter 15: 29460.457336709373
```

Fig. 22. Dataset 11 Gradient Descent Cost

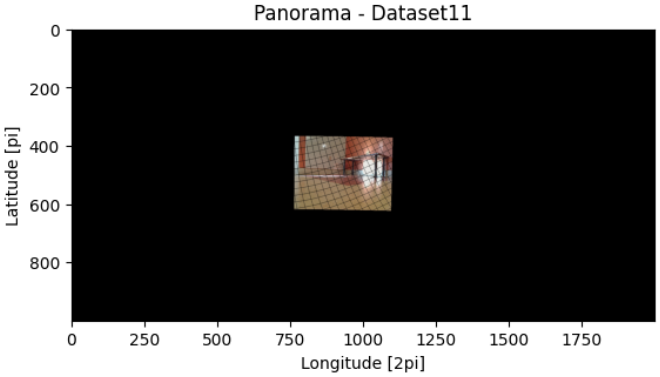


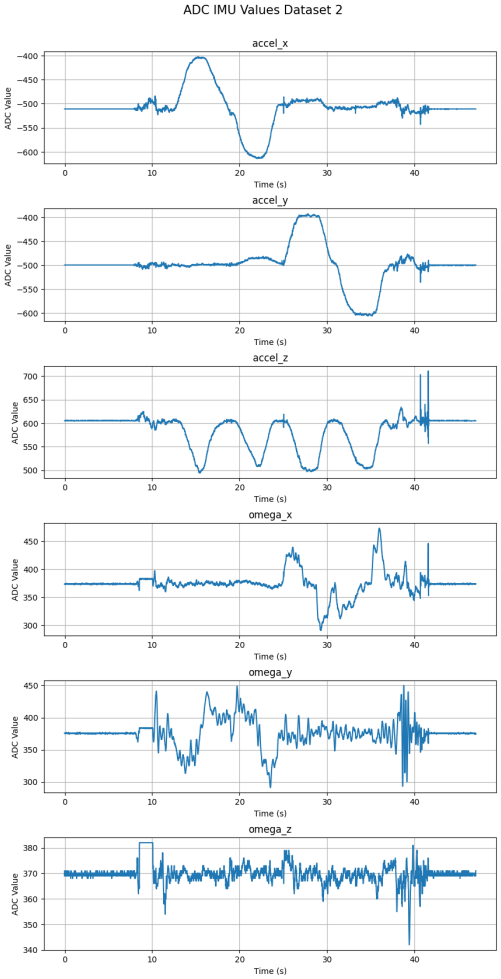
Fig. 23. Enter Caption

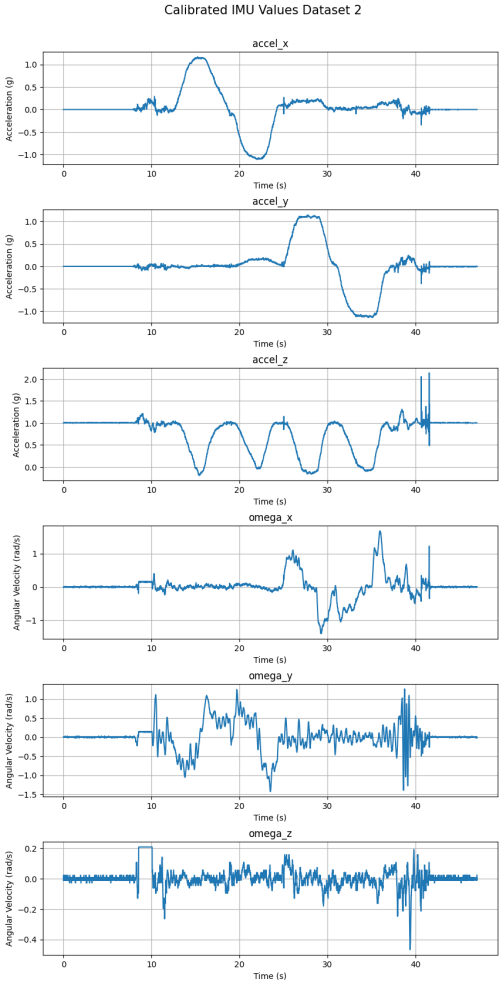
the vectorized quaternion operations struggle to account for edge cases and exhibit unexpected results. The main one being a euler angle with the same magnitude but rotated by π. This contributes to a noisy optimized trajectory as seen in the testsets. To fix this, perhaps revising the implementations regarding trigonometric functions will allow for a smooth initialization of the motion model leading to a smooth optimized

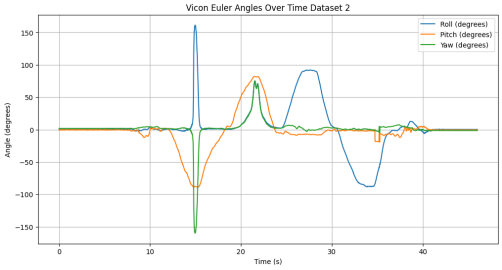
trajectory as a result. This is likely the case as in some datasets in the appendix, the rotation matrix to euler angle calculation from the VICON data also exhibited some sign changes. The panorama implementation works perfectly with the VICON data. It squishes images using the IMU data. This likely has to do with the method that the IMU rotation matrices are stored versus the VICON rotation matrix dataset. Fixing the IMU initial motion model and rotation matrix representation will lead to cleaner results.

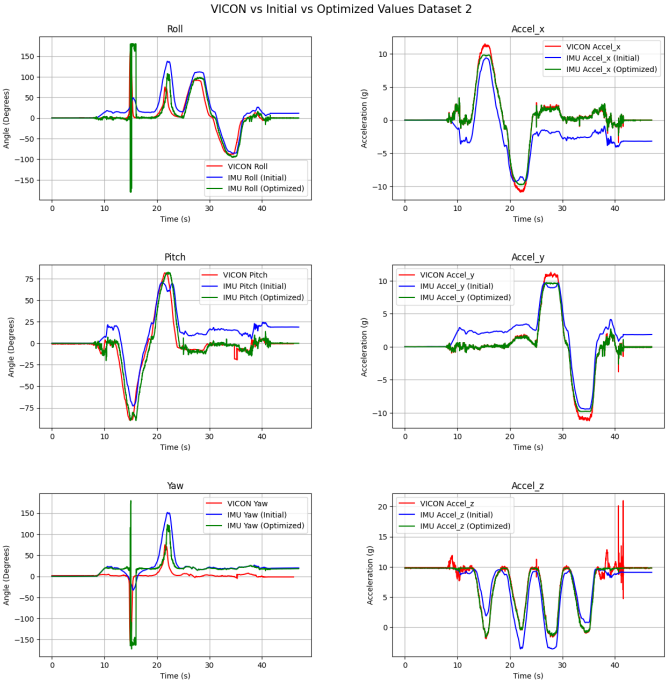
VI. Appendix - Other Datasets

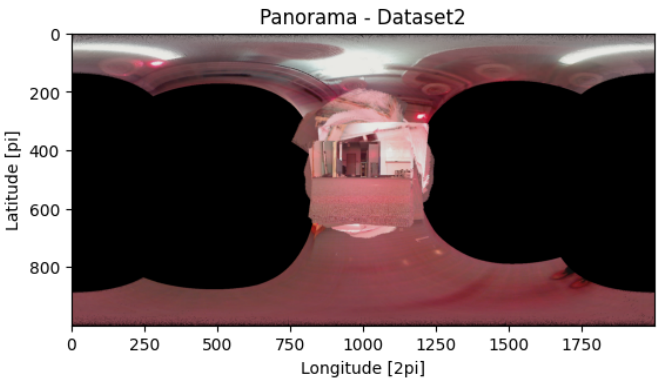
This includes data from other datasets, please see the panorama images for these datasets. The learning rate for the gradient descent algorithm of each dataset is $\alpha = 0.001$ with $n = 15$ iterations.

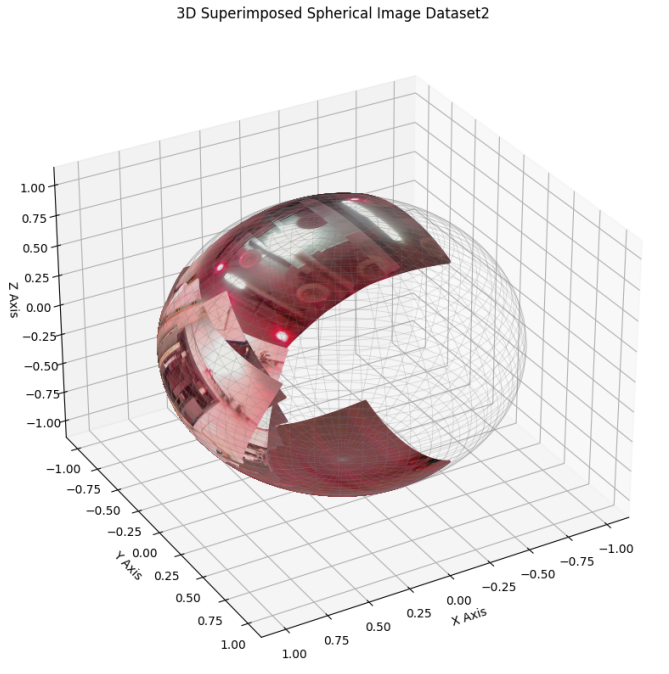


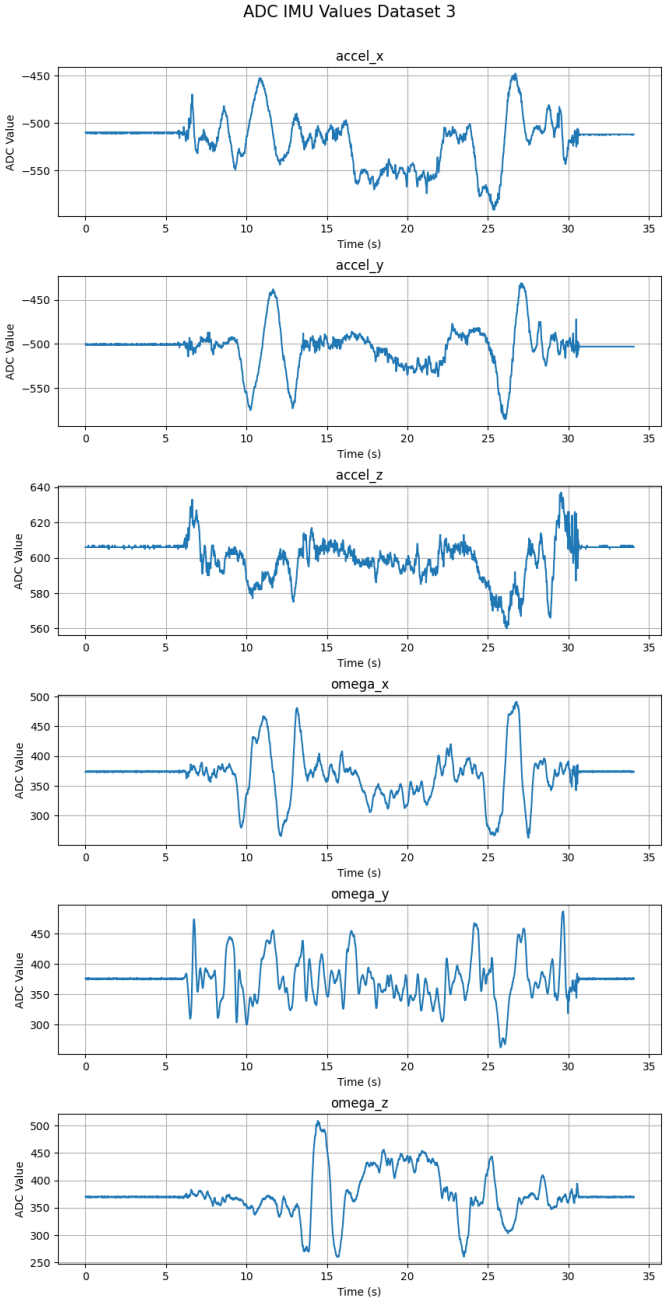


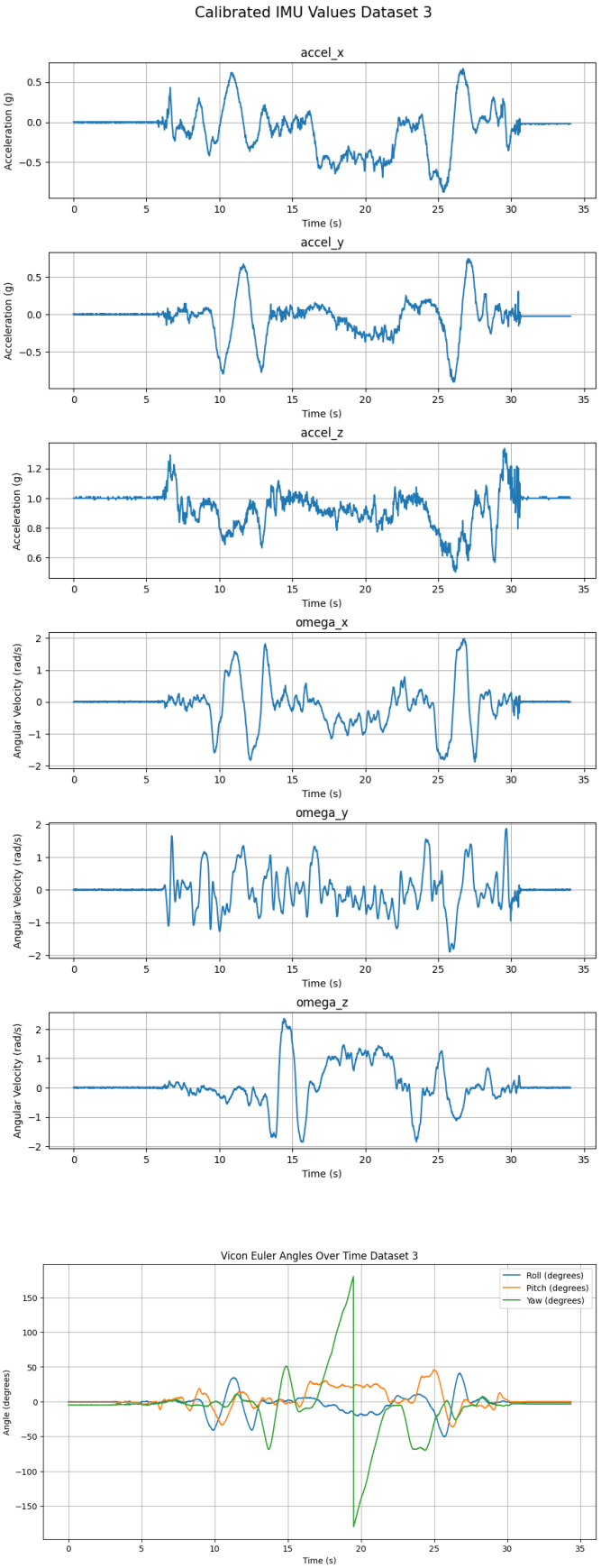


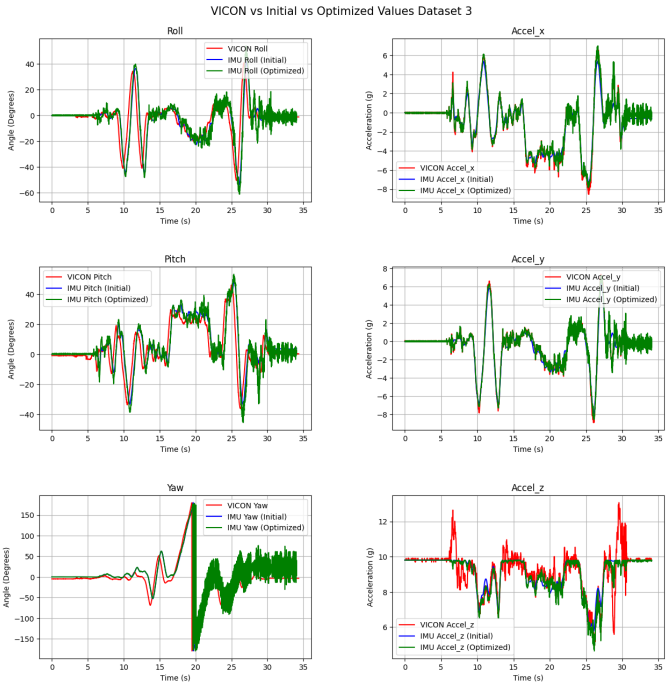

VICON vs Initial vs Optimized Values Dataset 2
Roll
Angle (Degrees)
Time (s)
VICON Roll
IMU Roll (Initial)
IMU Roll (Optimized)
Accel_x
Acceleration (g)
VICON Accel_x
IMU Accel_x (Initial)
IMU Accel_x (Optimized)
Pitch
VICON Pitch
IMU Pitch (Initial)
IMU Pitch (Optimized)
Accel_y
VICON Accel_y
IMU Accel_y (Initial)
IMU Accel_y (Optimized)
Yaw
VICON Yaw
IMU Yaw (Initial)
IMU Yaw (Optimized)
Accel_z
VICON Accel_z
IMU Accel_z (Initial)
IMU Accel_z (Optimized)


Panorama - Dataset2
Latitude [pi]
Longitude [2pi]

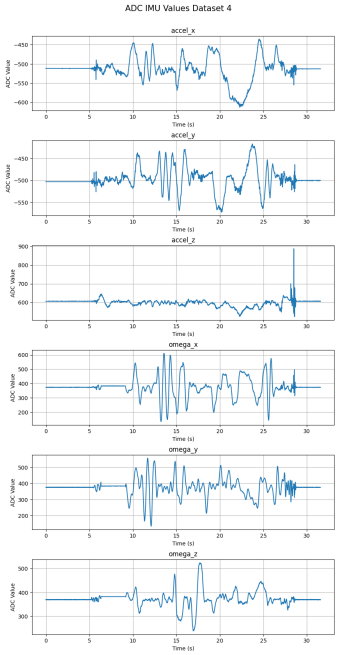

3D Superimposed Spherical Image Dataset2
Z Axis
Y Axis
X Axis

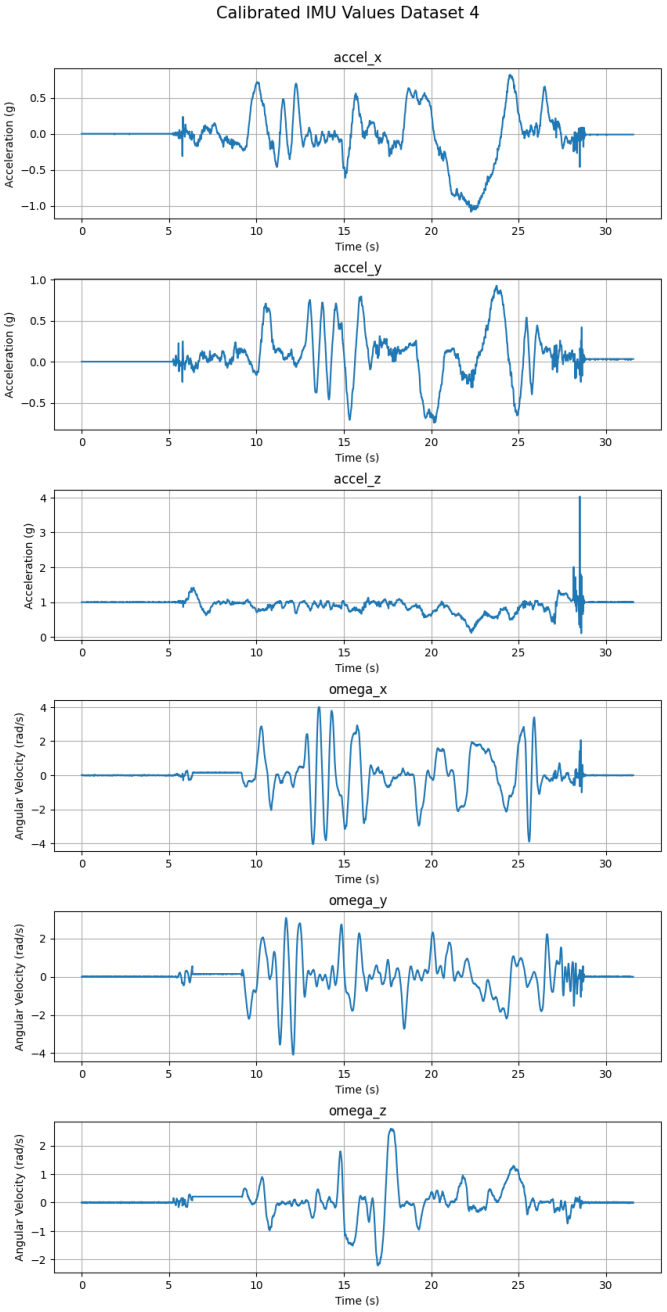

ADC IMU Values Dataset 3
accel_x
accel_y
accel_z
omega_x
omega_y
omega_z
ADC Value
Time (s)

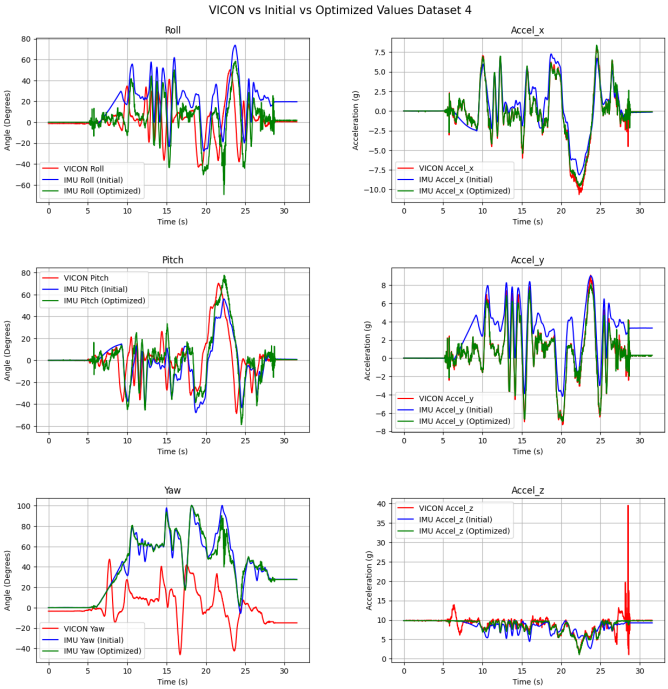

Calibrated IMU Values Dataset 3
accel_x
Acceleration (g)
Time (s)
accel_y
Acceleration (g)
Time (s)
accel_z
Acceleration (g)
Time (s)
omega_x
Angular Velocity (rad/s)
Time (s)
omega_y
Angular Velocity (rad/s)
Time (s)
omega_z
Angular Velocity (rad/s)
Time (s)

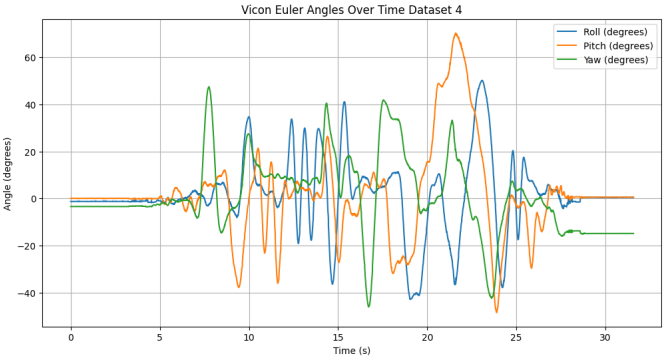

VICON vs Initial vs Optimized Values Dataset 3
Roll
VICON Roll
IMU Roll (Initial)
IMU Roll (Optimized)
Angle (Degrees)
Time (s)
Accel_x
VICON Accel_x
IMU Accel_x (Initial)
IMU Accel_x (Optimized)
Acceleration (g)
Time (s)
Pitch
VICON Pitch
IMU Pitch (Initial)
IMU Pitch (Optimized)
Angle (Degrees)
Time (s)
Accel_y
VICON Accel_y
IMU Accel_y (Initial)
IMU Accel_y (Optimized)
Acceleration (g)
Time (s)
Yaw
VICON Yaw
IMU Yaw (Initial)
IMU Yaw (Optimized)
Angle (Degrees)
Time (s)
Accel_z
VICON Accel_z
IMU Accel_z (Initial)
IMU Accel_z (Optimized)
Acceleration (g)
Time (s)

Vicon Euler Angles Over Time Dataset 3
Roll (degrees)
Pitch (degrees)
Yaw (degrees)
Angle (degrees)
Time (s)

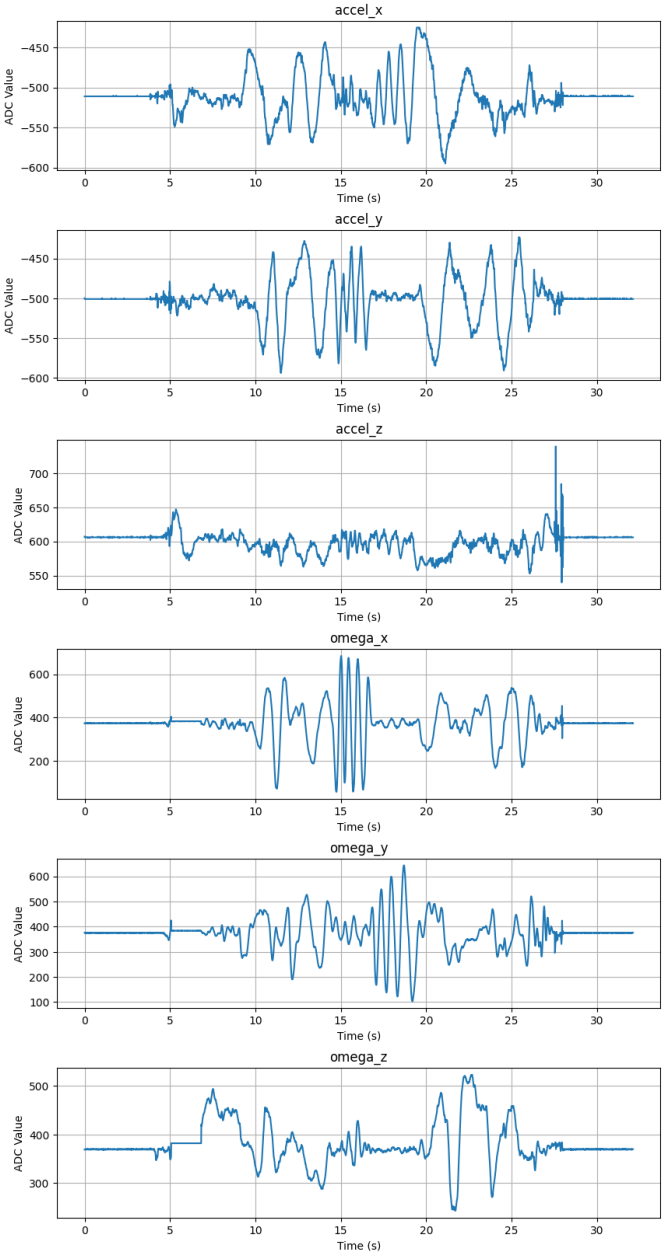

ADC IMU Values Dataset 4
accel_x
accel_y
accel_z
omega_x
omega_y
omega_z
ADC Value
Time (s)


Calibrated IMU Values Dataset 4
accel_x
Acceleration (g)
Time (s)
accel_y
Acceleration (g)
Time (s)
accel_z
Acceleration (g)
Time (s)
omega_x
Angular Velocity (rad/s)
Time (s)
omega_y
Angular Velocity (rad/s)
Time (s)
omega_z
Angular Velocity (rad/s)
Time (s)

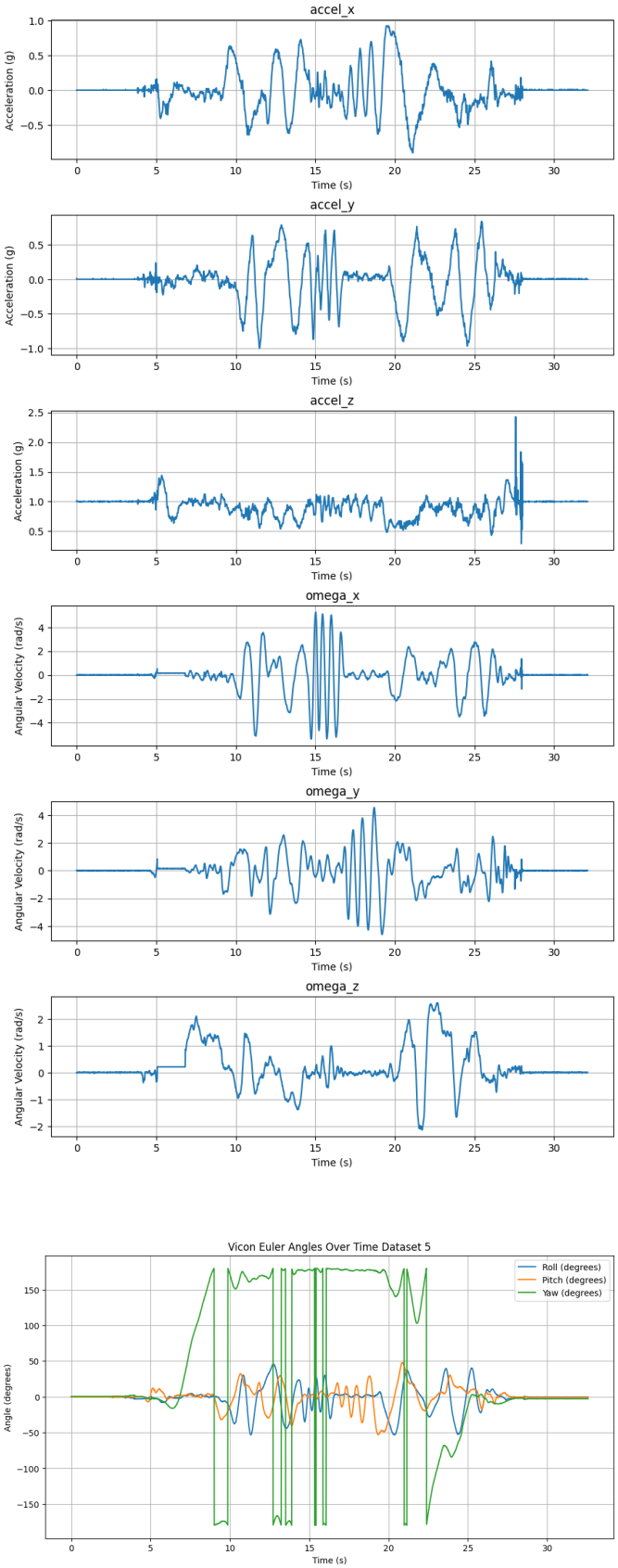

VICON vs Initial vs Optimized Values Dataset 4
Roll
Angle (Degrees)
VICON Roll
IMU Roll (Initial)
IMU Roll (Optimized)
Time (s)
Accel_x
Acceleration (g)
VICON Accel_x
IMU Accel_x (Initial)
IMU Accel_x (Optimized)
Time (s)
Pitch
Angle (Degrees)
VICON Pitch
IMU Pitch (Initial)
IMU Pitch (Optimized)
Time (s)
Accel_y
Acceleration (g)
VICON Accel_y
IMU Accel_y (Initial)
IMU Accel_y (Optimized)
Time (s)
Yaw
Angle (Degrees)
VICON Yaw
IMU Yaw (Initial)
IMU Yaw (Optimized)
Time (s)
Accel_z
Acceleration (g)
VICON Accel_z
IMU Accel_z (Initial)
IMU Accel_z (Optimized)
Time (s)


Vicon Euler Angles Over Time Dataset 4
Roll (degrees)
Pitch (degrees)
Yaw (degrees)
Angle (degrees)
Time (s)

ADC IMU Values Dataset 5



Calibrated IMU Values Dataset 5



Vicon Euler Angles Over Time Dataset 5

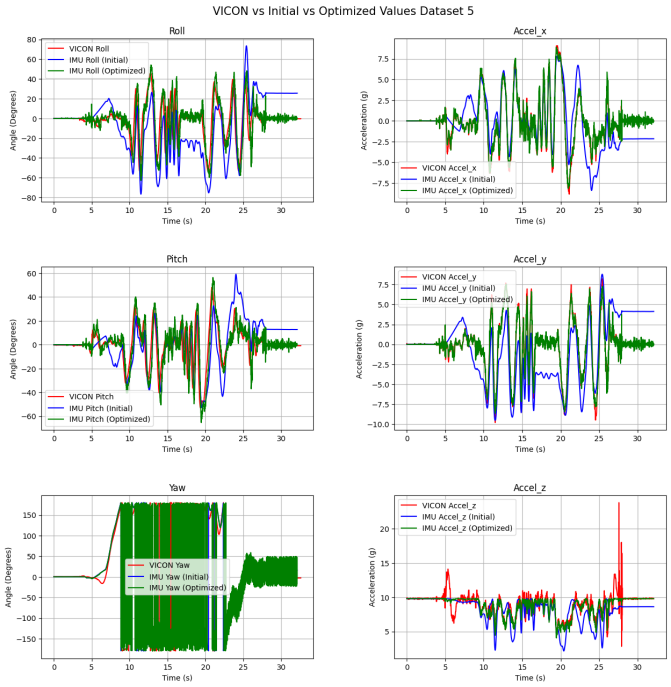
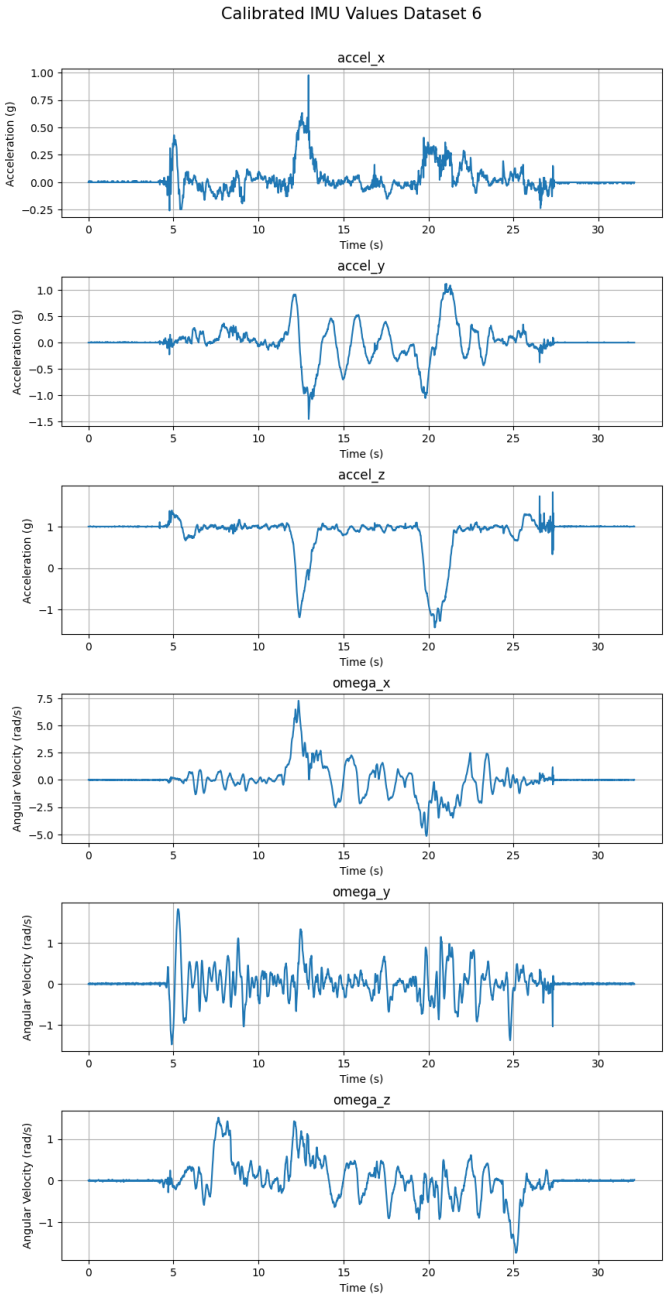
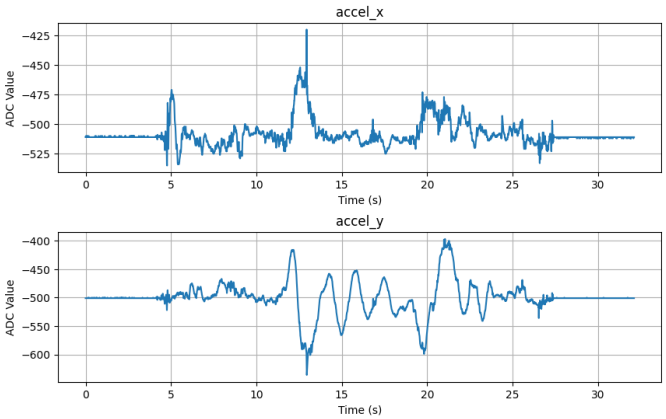
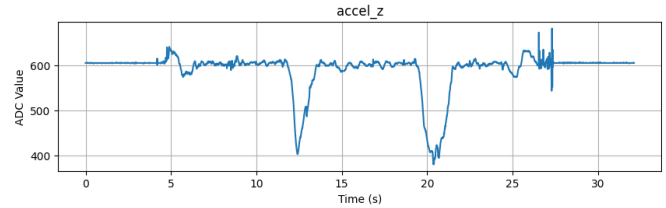


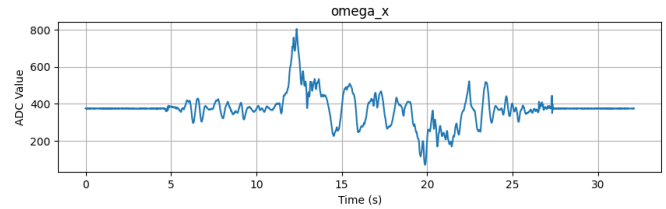
Fig. 24. Enter Caption

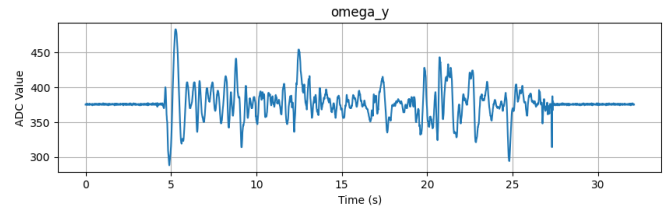


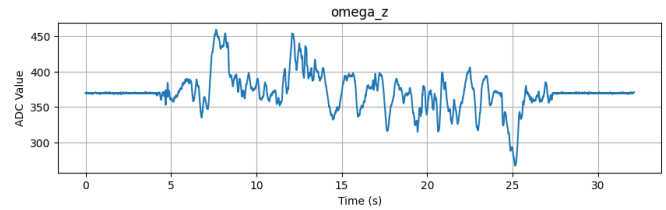
ADC IMU Values Dataset 6

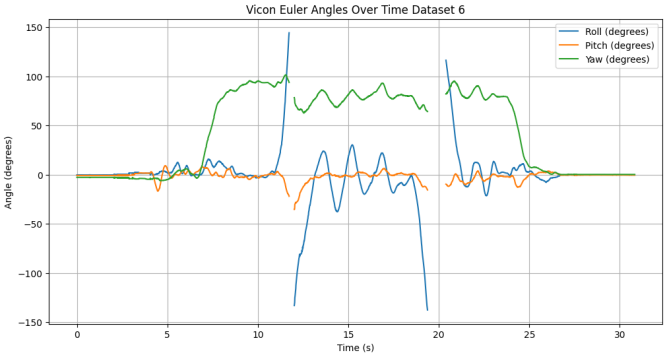


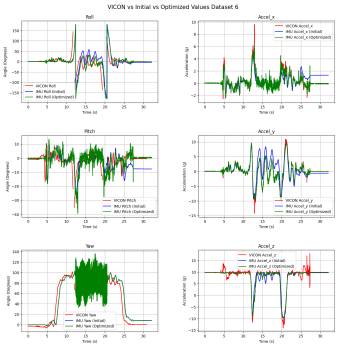




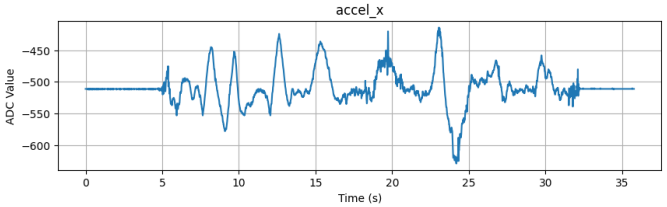


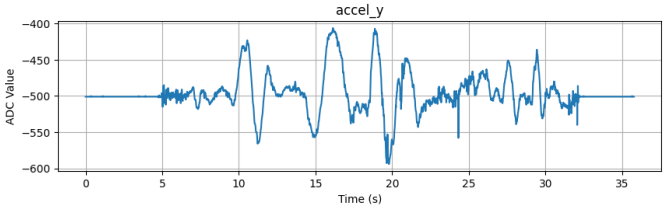


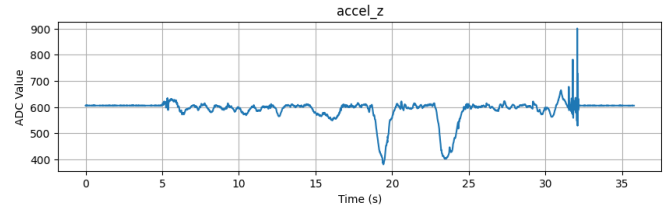


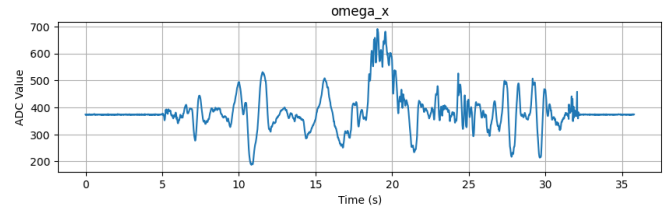


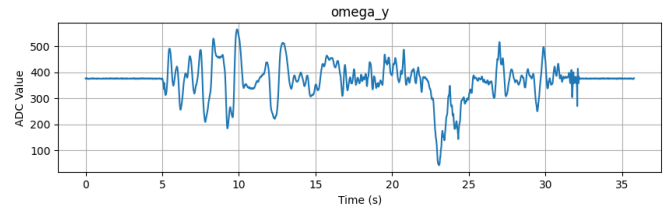
ADC IMU Values Dataset 7

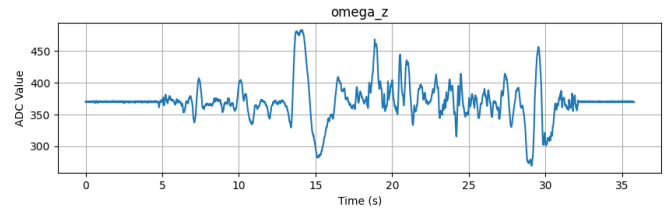


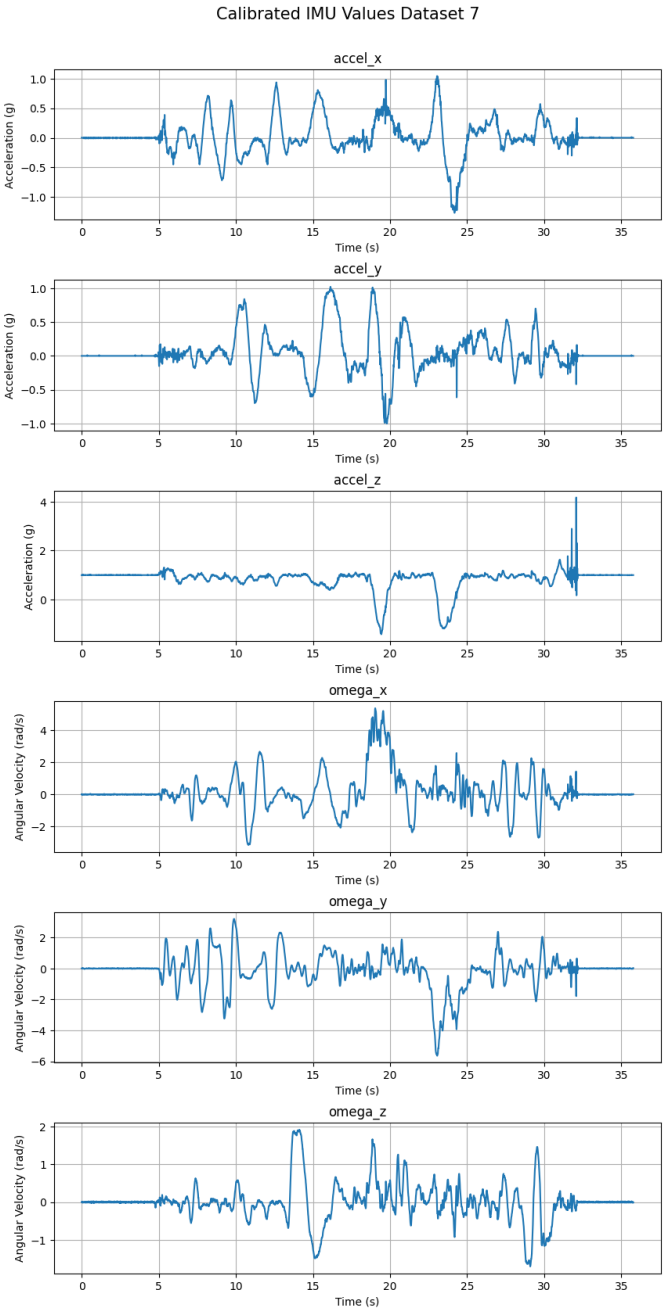


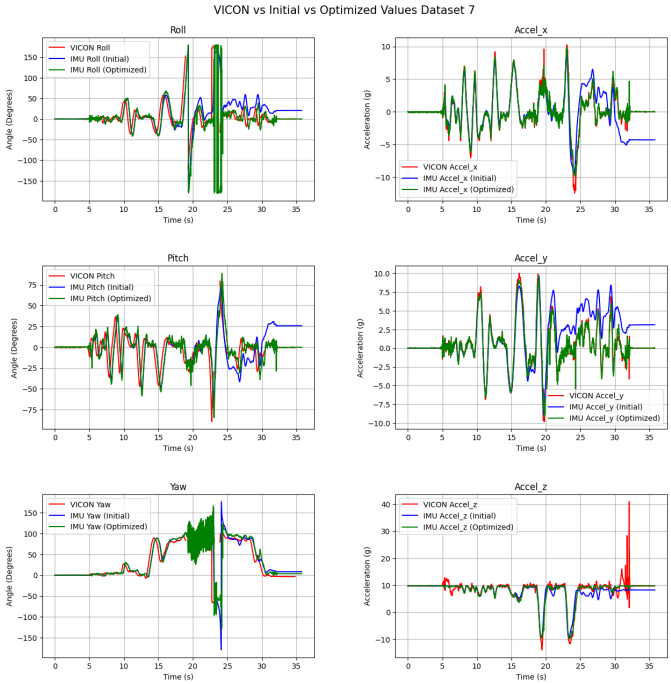


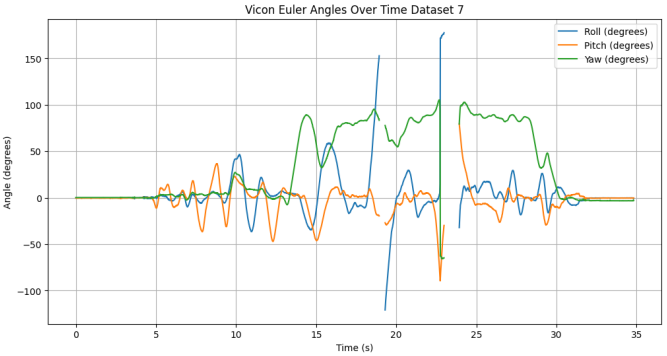




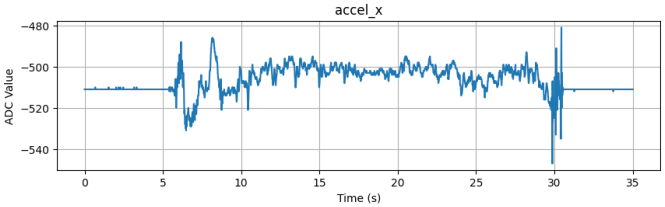


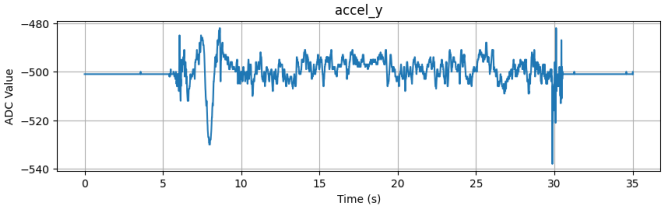

Calibrated IMU Values Dataset 7
accel_x
Acceleration (g)
Time (s)
accel_y
Acceleration (g)
Time (s)
accel_z
Acceleration (g)
Time (s)
omega_x
Angular Velocity (rad/s)
Time (s)
omega_y
Angular Velocity (rad/s)
Time (s)
omega_z
Angular Velocity (rad/s)
Time (s)

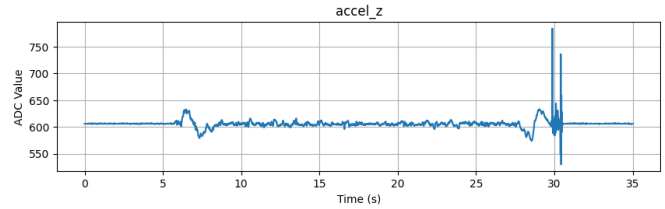

VICON vs Initial vs Optimized Values Dataset 7
Roll
VICON Roll
IMU Roll (Initial)
IMU Roll (Optimized)
Angle (Degrees)
Time (s)
Accel_x
VICON Accel_x
IMU Accel_x (Initial)
IMU Accel_x (Optimized)
Acceleration (g)
Time (s)
Pitch
VICON Pitch
IMU Pitch (Initial)
IMU Pitch (Optimized)
Angle (Degrees)
Time (s)
Accel_y
VICON Accel_y
IMU Accel_y (Initial)
IMU Accel_y (Optimized)
Acceleration (g)
Time (s)
Yaw
VICON Yaw
IMU Yaw (Initial)
IMU Yaw (Optimized)
Angle (Degrees)
Time (s)
Accel_z
VICON Accel_z
IMU Accel_z (Initial)
IMU Accel_z (Optimized)
Acceleration (g)
Time (s)

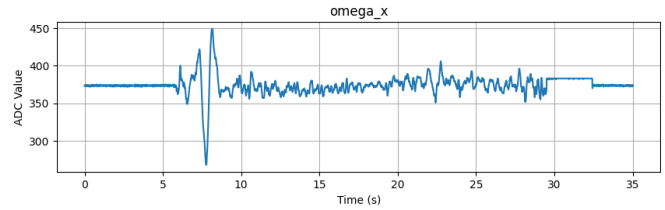

Vicon Euler Angles Over Time Dataset 7
Roll (degrees)
Pitch (degrees)
Yaw (degrees)
Angle (degrees)
Time (s)

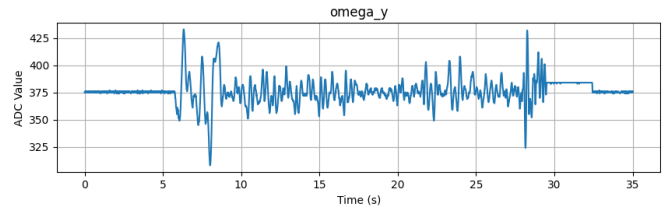
ADC IMU Values Dataset 8

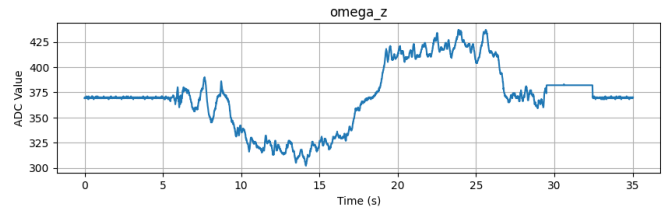




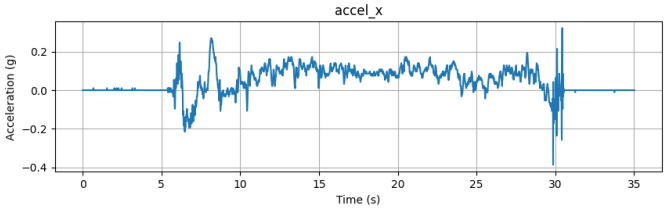


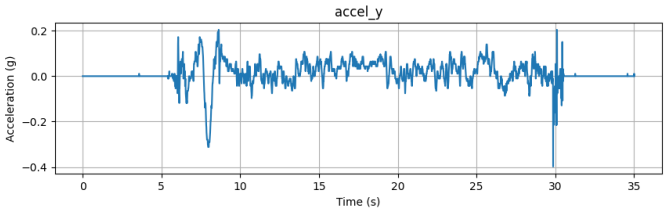


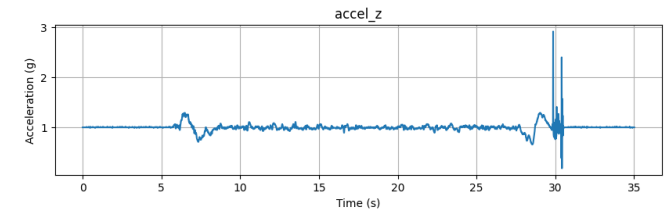


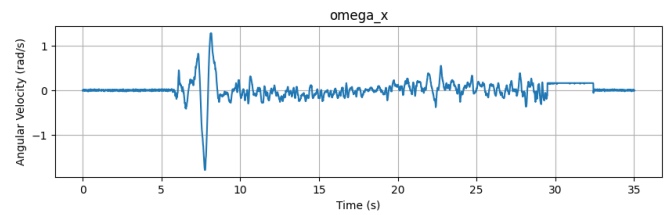


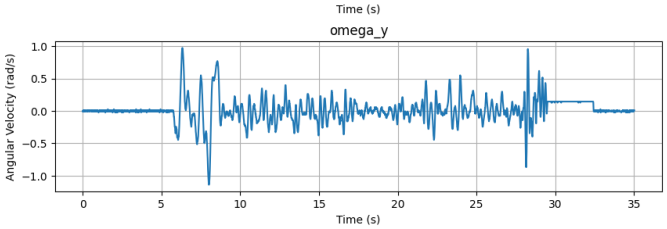
Calibrated IMU Values Dataset 8

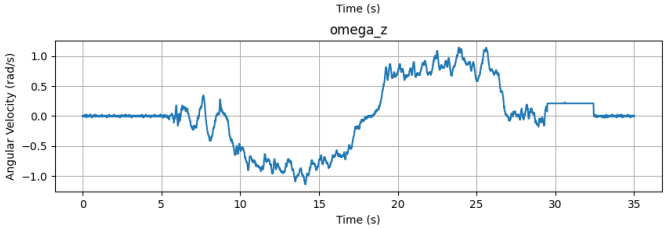


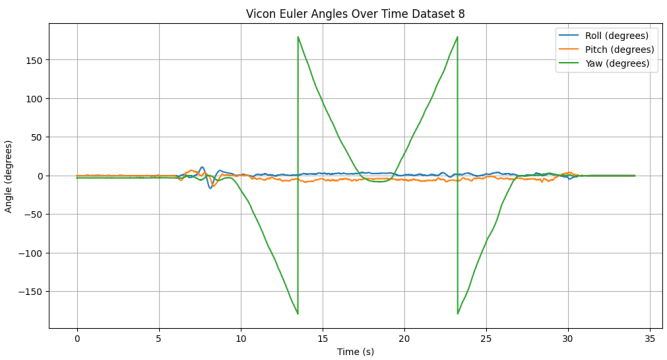


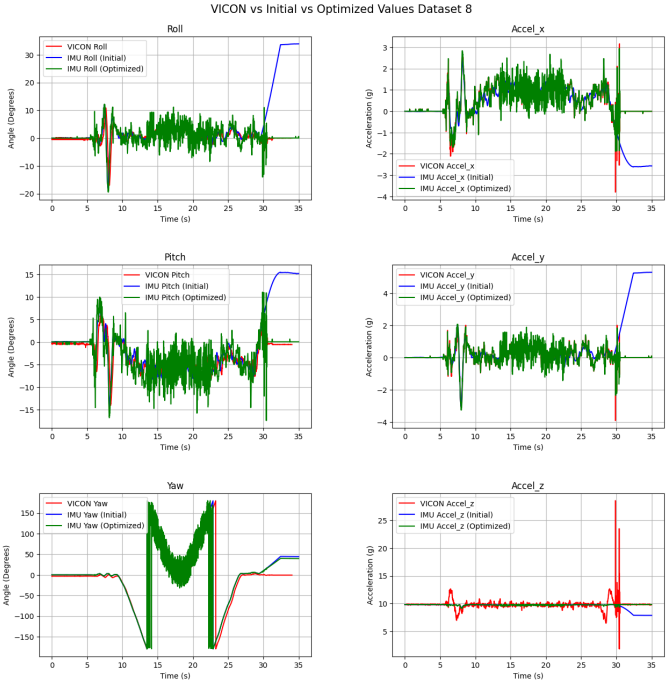


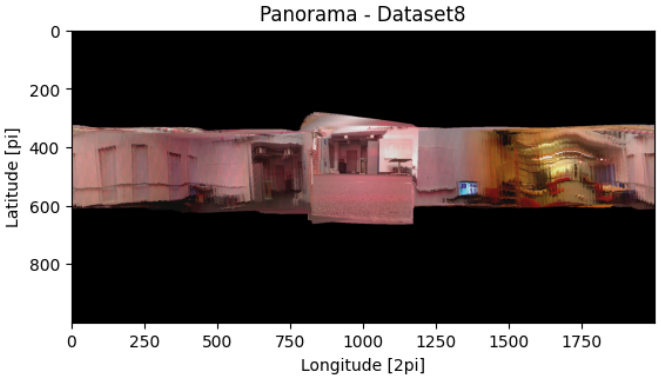


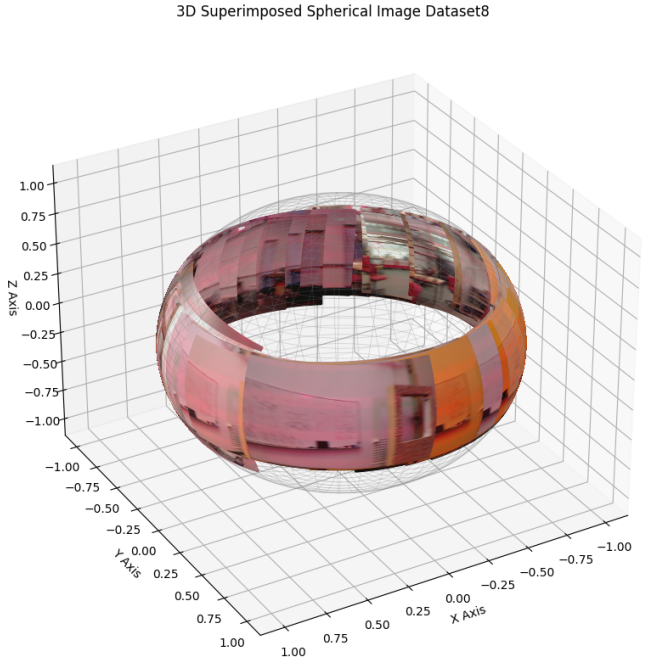


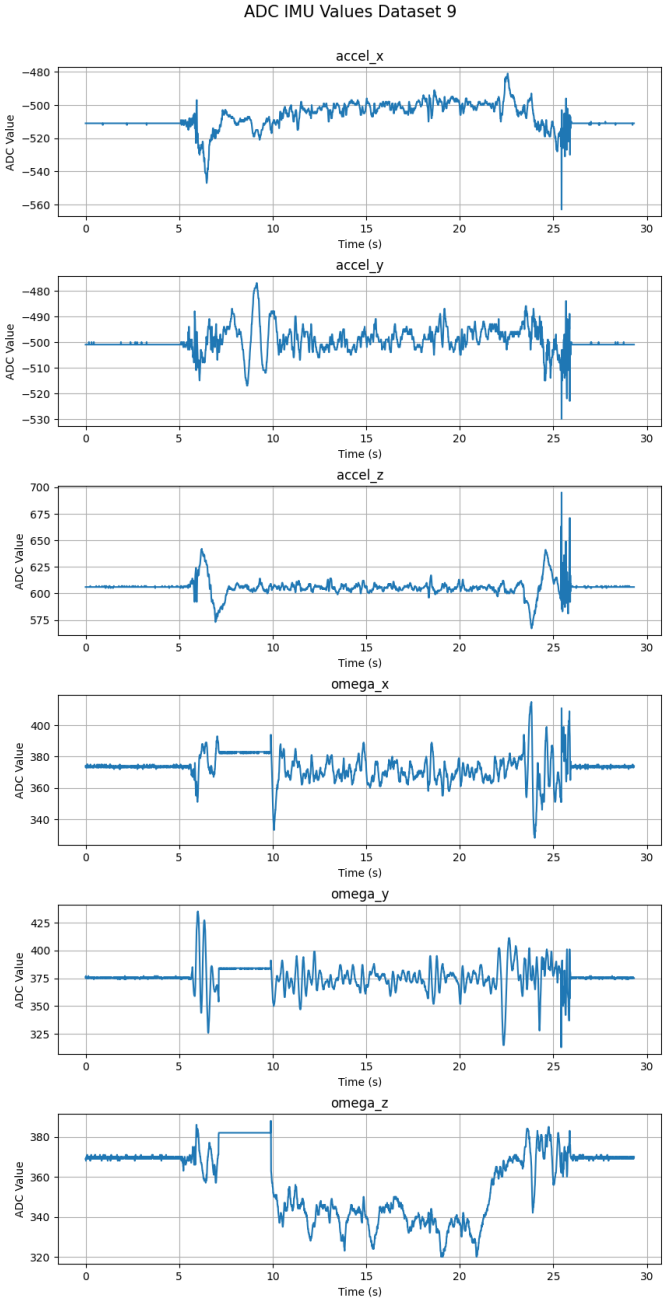




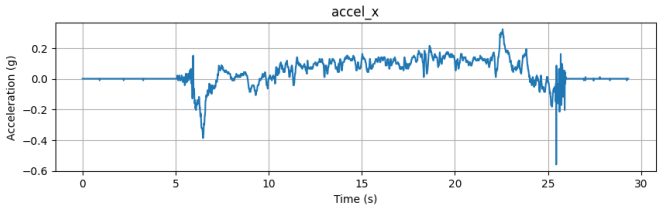

VICON vs Initial vs Optimized Values Dataset 8
Roll
VICON Roll
IMU Roll (Initial)
IMU Roll (Optimized)
Angle (Degrees)
Time (s)
Accel_x
VICON Accel_x
IMU Accel_x (Initial)
IMU Accel_x (Optimized)
Acceleration (g)
Time (s)
Pitch
VICON Pitch
IMU Pitch (Initial)
IMU Pitch (Optimized)
Angle (Degrees)
Time (s)
Accel_y
VICON Accel_y
IMU Accel_y (Initial)
IMU Accel_y (Optimized)
Acceleration (g)
Time (s)
Yaw
VICON Yaw
IMU Yaw (Initial)
IMU Yaw (Optimized)
Angle (Degrees)
Time (s)
Accel_z
VICON Accel_z
IMU Accel_z (Initial)
IMU Accel_z (Optimized)
Acceleration (g)
Time (s)

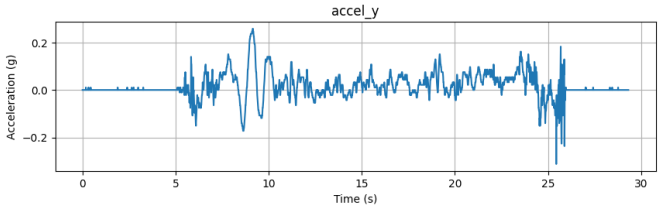

Panorama - Dataset8
Latitude [pi]
Longitude [2pi]

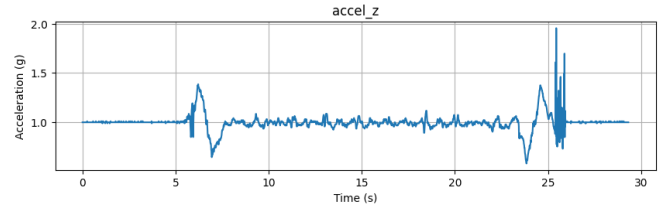

3D Superimposed Spherical Image Dataset8
Z Axis
Y Axis
X Axis

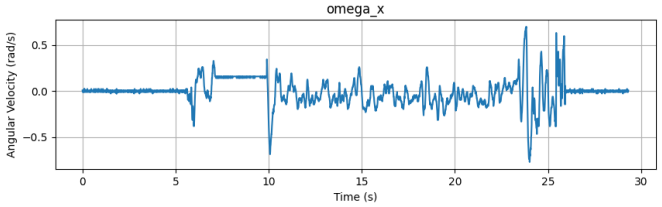

ADC IMU Values Dataset 9
accel_x
accel_y
accel_z
omega_x
omega_y
omega_z
ADC Value
Time (s)

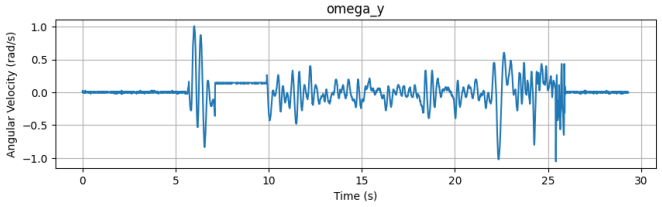
Calibrated IMU Values Dataset 9

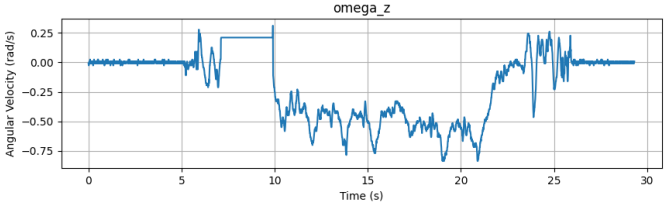


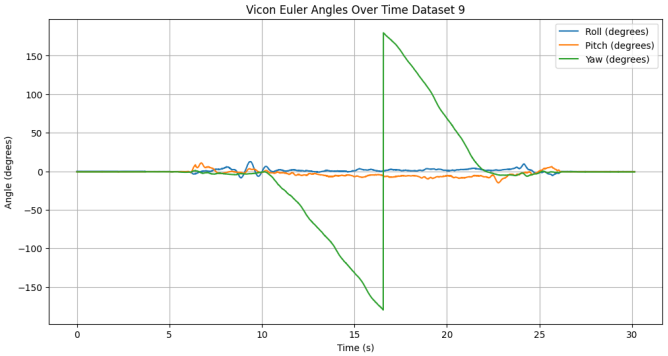


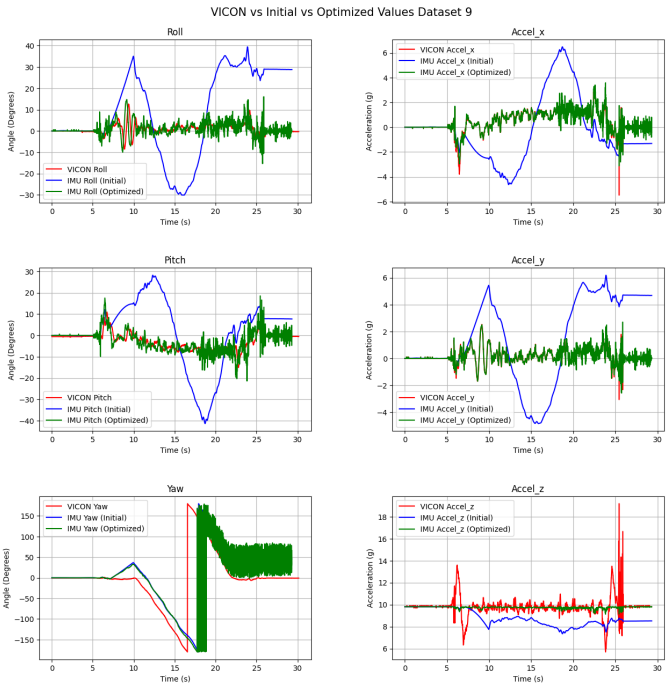












3D Superimposed Spherical Image Dataset9