Report for Project one ECE276A

Amirhosein Javadi

February 8, 2024

1 Introduction

Orientation tracking, the process of accurately determining the orientation or pose of an object in three-dimensional space, is a fundamental problem with far-reaching applications in various fields such as robotics, virtual reality, augmented reality, and human-computer interaction. The ability to precisely track the orientation of objects is crucial for tasks ranging from navigation and control of autonomous vehicles to the development of immersive virtual environments. In this project, we address the challenge of orientation tracking using measurements from an inertial measurement unit (IMU). IMUs provide data on the linear acceleration and angular velocity of a body in motion, along with precise timestamps. By leveraging this data, we implement a projected gradient descent algorithm to estimate the three-dimensional orientation of a rotating body accurately. Our project also extends beyond orientation tracking by utilizing these orientation estimates to generate panoramic images. By stitching together camera images obtained by the rotating body, we create a comprehensive visual representation of the environment. Additionally, we validate our orientation estimates against ground truth data provided by a VICON motion capture system, ensuring the accuracy and reliability of our tracking algorithm. Through this project, we aim to demonstrate the importance and practicality of orientation tracking techniques in real-world scenarios, showcasing their potential to enhance various applications across different domains.

2 Problem Formulation

2.1 Orientation Tracking

In the initial phase of the project, our goal is to determine the body's orientation throughout time by utilizing measurements of IMU angular velocity ω_t and linear acceleration. To represent the orientation of the body frame at time t, we employ a unit quaternion q_t belonging to the unit quaternion space $\mathbb{H}^*$. By leveraging the IMU angular velocity measurements ω_t alongside the differences between consecutive timestamps τ_t, we can forecast the quaternion at the subsequent time step q_{t+1} using the quaternion kinematics motion model.

$$q_{t+1} = f(q_t, \tau_t w_t) := q_t \circ \exp([0, \frac{\tau_t w_t}{2}]) \tag{1}$$

We begin with the initial quaternion $q_0 = [1, 0, 0, 0]$ and proceed to integrate the angular velocity ω_t by computing the quaternion at the next time step q_{t+1} using the function $f(q_t, \tau_t \omega_t)$. Due to the body's pure rotational motion, its acceleration in the world frame should ideally approximate [0, 0, -g], where g represents the acceleration due to gravity. Consequently, the measured acceleration in the IMU frame should align with gravity acceleration after being transformed to the IMU frame using the orientation q_t. This results in the following observation model:

$$a_t = h(q_t) := q_{t^{-1}} \circ [0, 0, 0, -g] \circ q_t \tag{2}$$

So The cost function $c(q_{1:T})$ is defined as follows:

$$c(q_{1:T}) = \frac{1}{2}\sum_{t=0}^{T-1} \|2\log(q_{t+1}^{-1} \circ f(q_t, \tau_t\omega_t))\|_2^2 + \frac{1}{2}\sum_{t=1}^{T} \|a_t - h(q_t)\|_2^2 \tag{3}$$

We require that the quaternions q_t maintain a unit norm, i.e., $q_t \in \mathbb{H}^*$, throughout the optimization process. Therefore, we are faced with a constrained optimization problem:

$$\min_{q_{1:T}} c(q_{1:T}) \quad \text{subject to} \quad \|q_t\|_2 = 1, \quad \forall t \in \{1, 2, ..., T\} \tag{4}$$

In other words, we aim to minimize the cost function $c(q_{1:T})$ while ensuring that the norm of each quaternion q_t remains equal to 1 for all time steps t within the specified range.

2.2 Panorama

In the second phase of the project, our aim is to generate a panoramic image by seamlessly stitching together RGB camera images captured over time, leveraging the learned orientation trajectory $q_{1:T}$ of the body.

3 Technical Approach

3.1 Calibrations

Initially, I determined the scale factors for the accelerometer and gyroscope by referencing the provided documentation in the project. During the initial static period where no rotations occur, I utilized this timeframe to perform calibration for the accelerometer and gyroscope biases. For instance, during this calibration process, the accelerometer readings should ideally register as $[0, 0, 1]^T$ in units of gravity, reflecting the absence of any external accelerations.

3.2 Quaternion Functions

Since for loops are known to be computationally slow in Python, I opted to implement batch versions of Quaternion functions, including exponential, logarithm, inverse, and cost function. This approach significantly improved the efficiency of my calculations, enabling me to compute gradients in less than 1 second. Compared to using conventional functions and for loops, the batch implementation resulted in a remarkable speedup of 120 times.

3.3 Gradient Projection

By utilizing the Jax grad function, I am able to compute the gradient of my cost function with respect to $q_{1:T}$. Since the problem imposes a constraint on Quaternions to have a unit norm, I adopted a method outlined in a Stack Exchange link provided within the project description. First, I computed the gradient of the cost function with respect to Quaternion variables. Each row of this gradient matrix is the gradient of cost function with respect to its corrsponding Quaternion vector, for example q_i. Subsequently, I projected each row of this gradient onto its corresponding tangent plane and then normalized it. This process resulted in a matrix termed proj_grad in my code. The updated Quaternion is then determined using the formula:

$$q_k' = \cos(\alpha)q_k + \sin(\alpha)p_k^g$$

where p_k^g is the kth row of normalized projected gradient. This update procedure guarantees that the norm remains equal to one at each iteration. The subsequent stage involves conducting a one-dimensional search to determine the optimal α. Since we have normalized the projected gradient, all

the information about the current state of the Quaternion that could be derived from the norm of gradient is eliminated. Applying the same α to all Quaternion vectors would result in equal movements, regardless of their current state. Given that individually performing a one-dimensional search for each Quaternion vector is time-intensive, I leveraged Jax to compute the deviation of the cost function concerning α array. Assuming that I have determined the optimal α array, with each Quaternion vector represented by a row, the Quaternion update rule becomes:

$$q'_k = cos(\alpha_k)q_k + p^g_k sin(\alpha_k) \tag{5}$$

To determine the optimal α array, I initially generate an α_0 array, initialized as the zero vector. Subsequently, I compute the gradient of the cost function relative to $\alpha = \alpha_0$. Utilizing Jax, the resulting deviation is calculated as follows:

$$\frac{\partial C}{\partial \alpha} = \frac{\partial C}{\partial q'}\frac{\partial q'}{\partial \alpha} \tag{6}$$

The resulting deviation provides insight into the impact of each element in the α array on the cost function. Subsequently, I compute the loss function for $\alpha_0 - lr * \frac{\partial C}{\partial \alpha}$, employing a learning rate (lr). In each iteration, I begin with a lr value of 10^{-2}, halving it iteratively until it falls below 10^{-5}. Next, I identify the optimal learning rate associated with the smallest cost function and proceed to the next iteration. This optimization process continues until no further improvement can be achieved.

3.4 Panorama

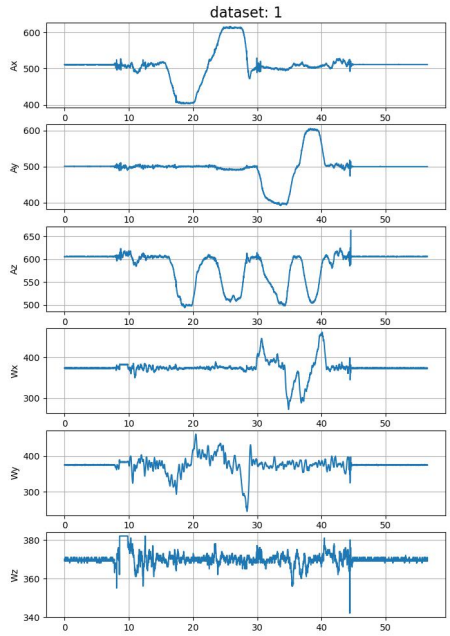
Upon computing the Quaternion vectors, I employed them to construct a panorama using the camera dataset. Initially, I computed the horizontal and vertical field of view (FOV), as well as the height and width of my camera. Utilizing these parameters, I generated two grids representing the longitude and latitude of the spherical coordinates for each pixel in the camera images. Subsequently, I converted each set of spherical coordinates to its corresponding Cartesian coordinates. For each image, I associated a timestamp and identified the Quaternion closest in time to the image's timestamp. From this Quaternion, I constructed a rotation matrix. Using the rotation matrix, I rotated the Cartesian coordinates and subsequently converted them back to spherical coordinates. Following the logic outlined on slide 5, where point $(\lambda, \phi, 1)$ on the sphere corresponds to height ϕ on the cylinder and longitude λ along the cylinder circumference, I determined the position of each image on the cylinder. Once all images were transferred to the cylinder, I unwrapped the cylinder surface to produce a rectangular image with a width of 2π radians and a height of π radians.

4 Results

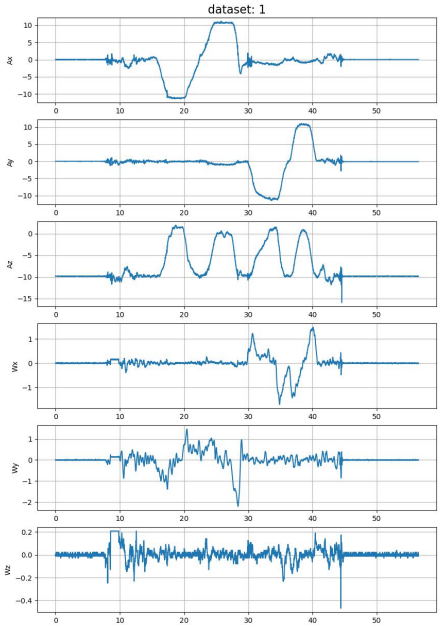
The result provided for each dataset are:

1. Raw IMU data
2. Calibrated IMU data after applying bias and scale adjustments
3. Euler angles calculated from initializations compared with the ground truth Euler angles from VICON data
4. Rotated acceleration calculated by rotating gravity with initialized Quaternion and calibration IMU acceleration data
5. Euler angles and rotated acceleration calculated from optimized Quaternion vectors
6. Loss function of the optimization process
7. Panorama created from the VICON rotation matrix and optimized Quaternion vectors

4.1 dataset 1

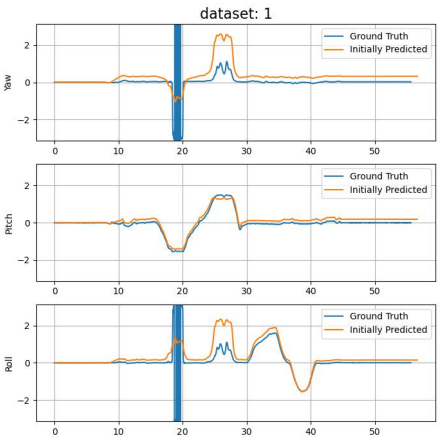


(a) Raw

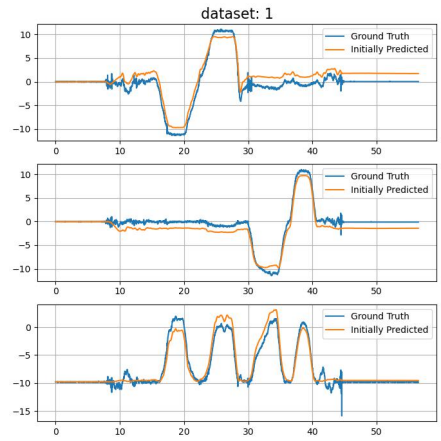


(b) Calibrated

Figure 1: IMU-data

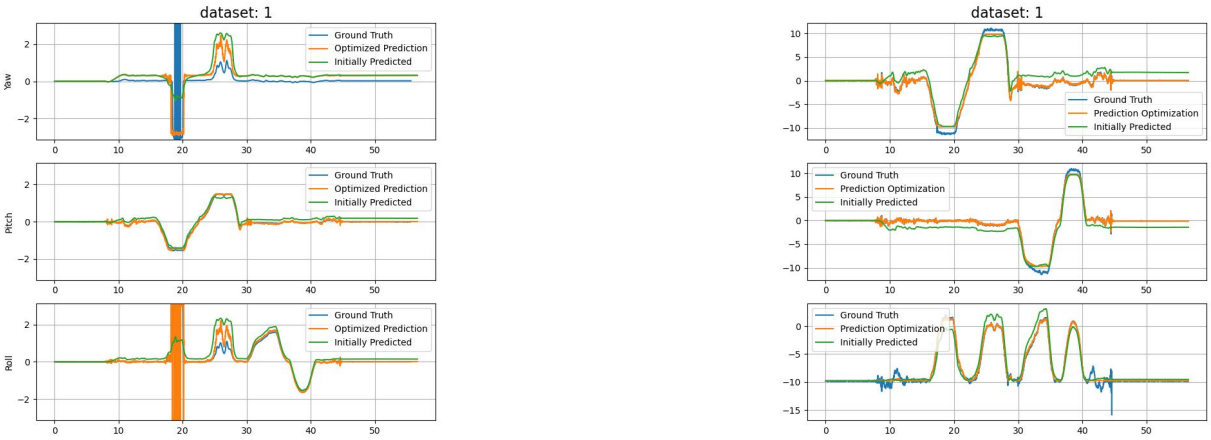


(a) Euler



(b) Acceleration

Figure 2: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 3: Euler and rotated acceleration after optimization

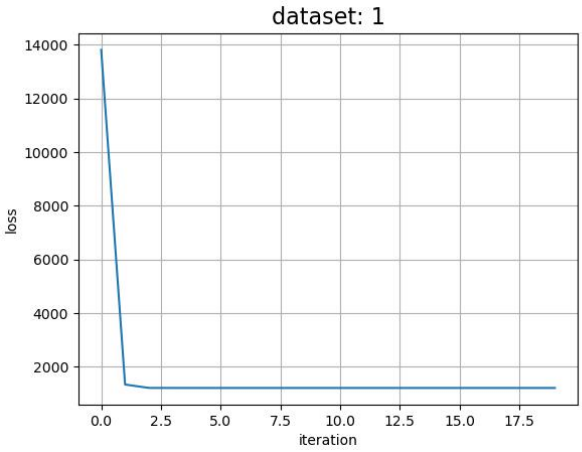
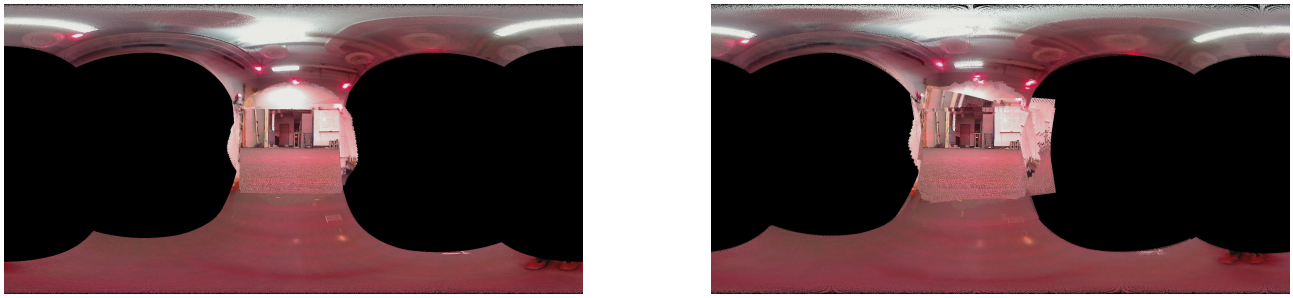


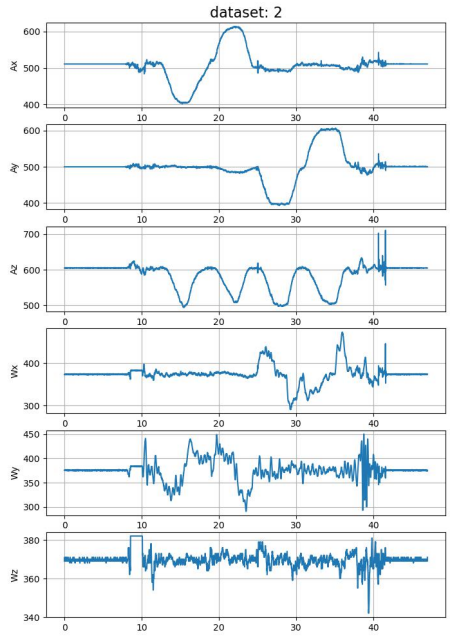
Figure 4: Loss function curve during optimization



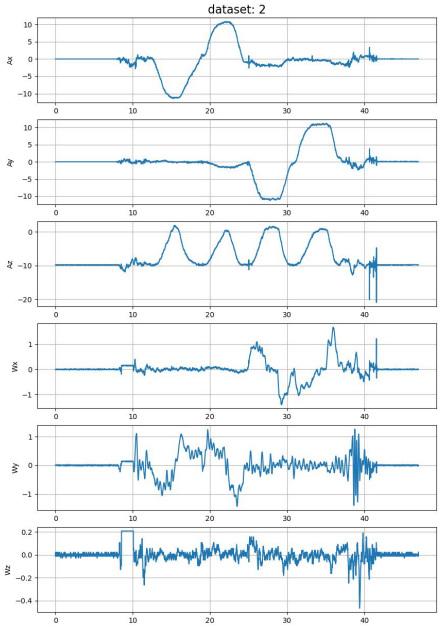
(a) VICON (b) Optimizations

Figure 5: Panorama Image with Rotations from VICON and optimizations

4.2 dataset 2

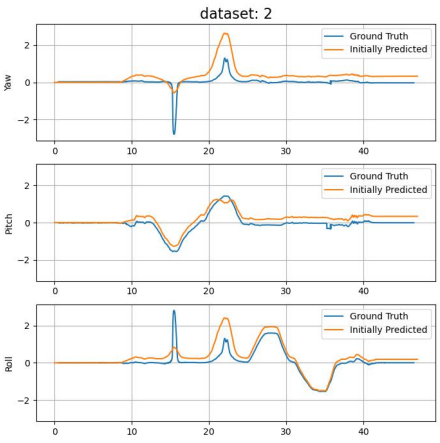


(a) Raw

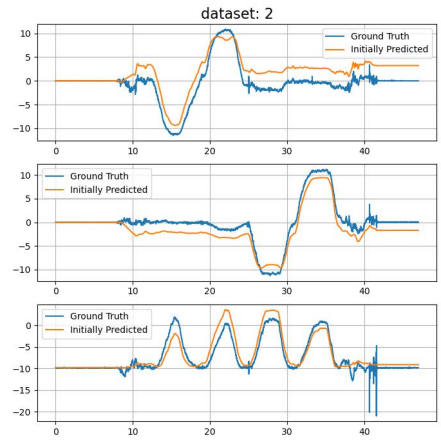


(b) Calibrated

Figure 6: IMU-data

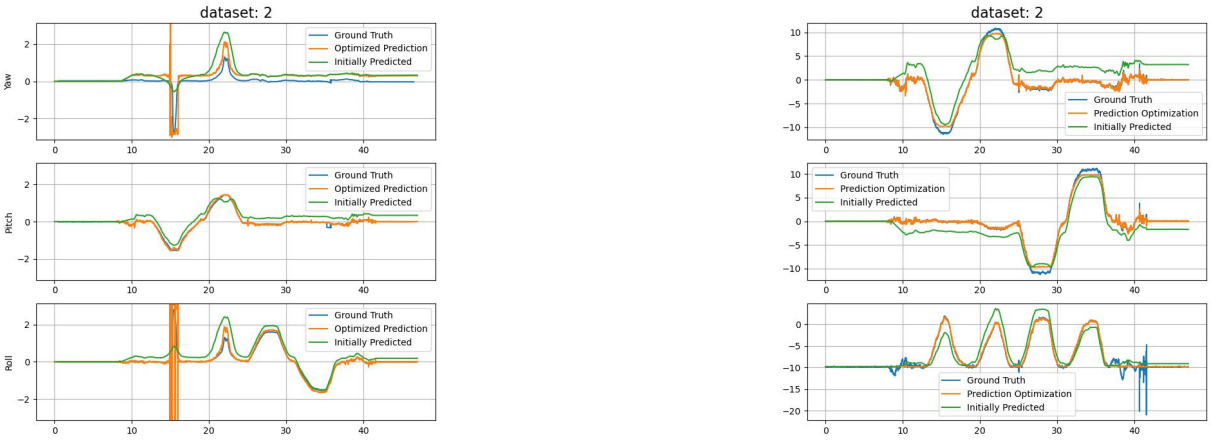


(a) Euler



(b) Acceleration

Figure 7: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 8: Euler and rotated acceleration after optimization

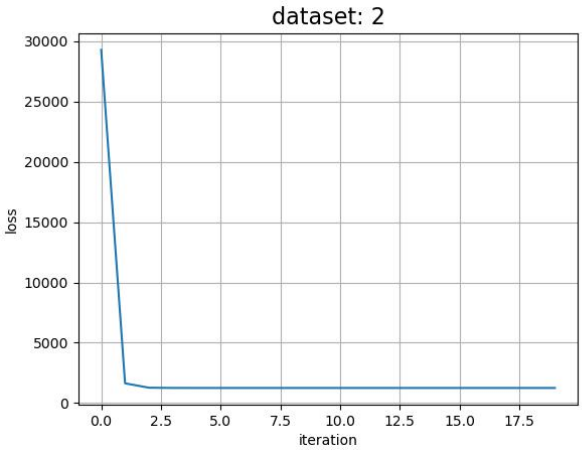
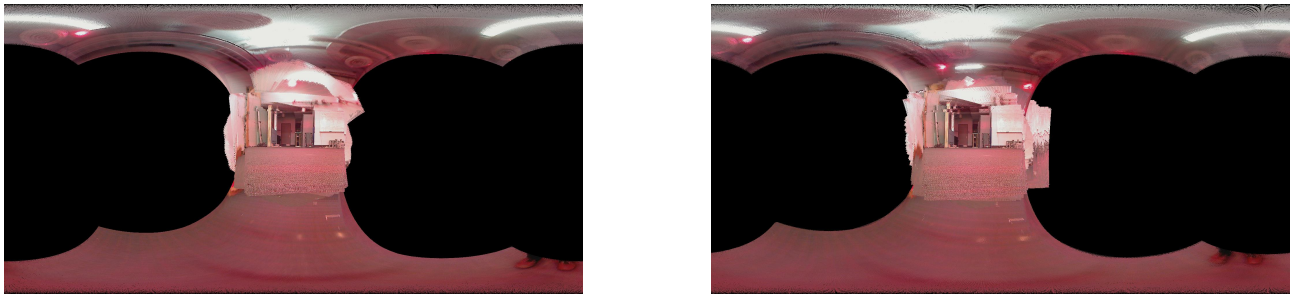


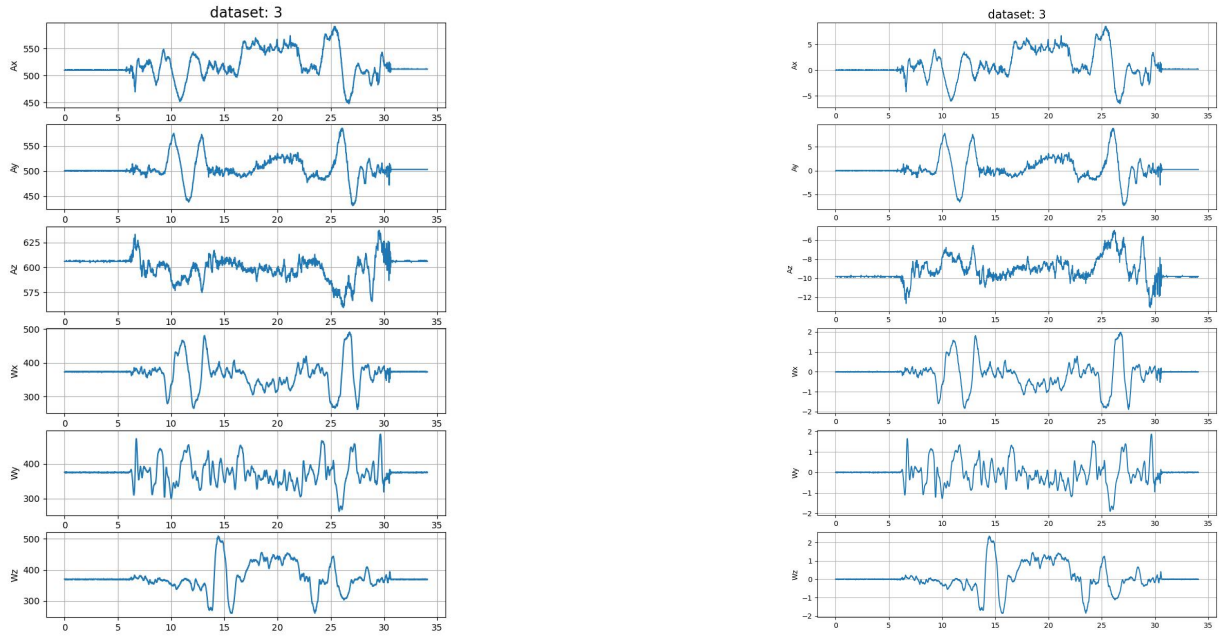
Figure 9: Loss function curve during optimization



(a) VICON (b) Optimizations

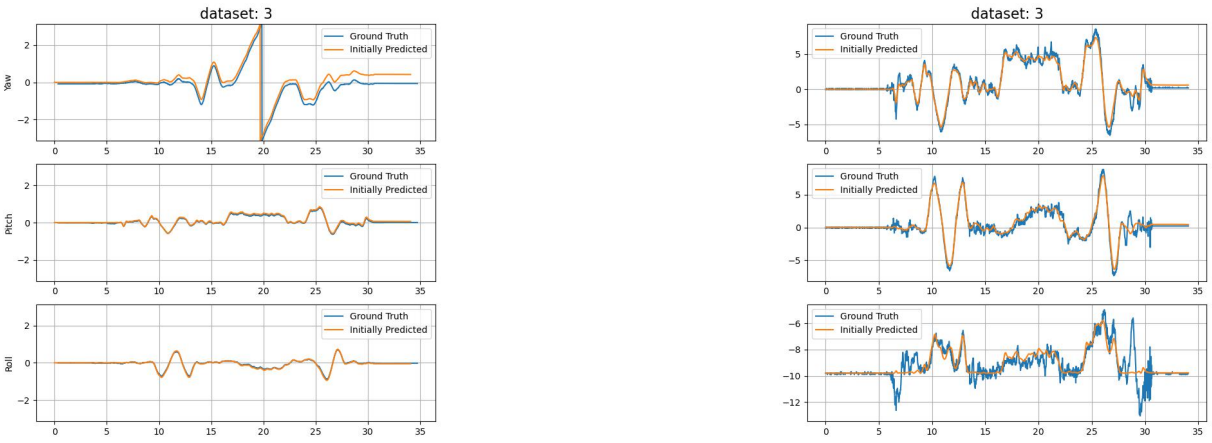
Figure 10: Panorama Image with Rotations from VICON and optimizations

4.3 dataset 3



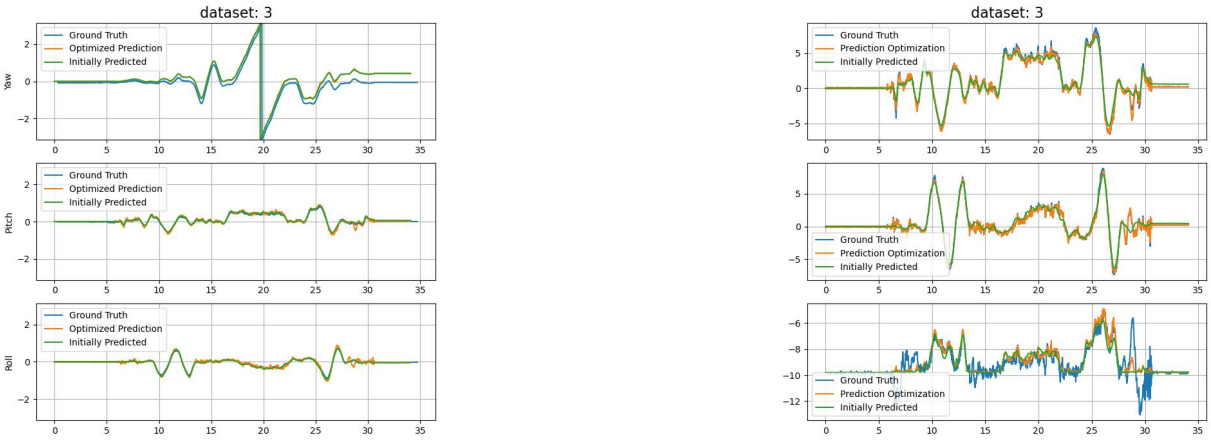
(a) Raw (b) Calibrated

Figure 11: IMU-data



(a) Euler (b) Acceleration

Figure 12: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 13: Euler and rotated acceleration after optimization

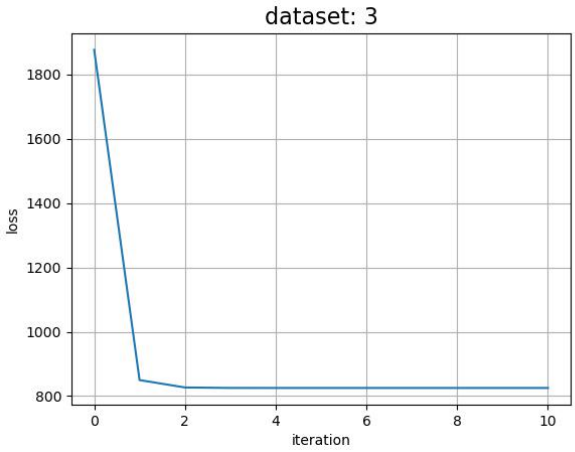
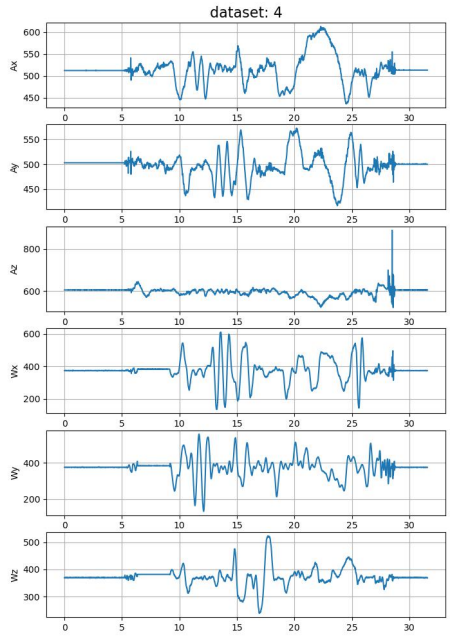
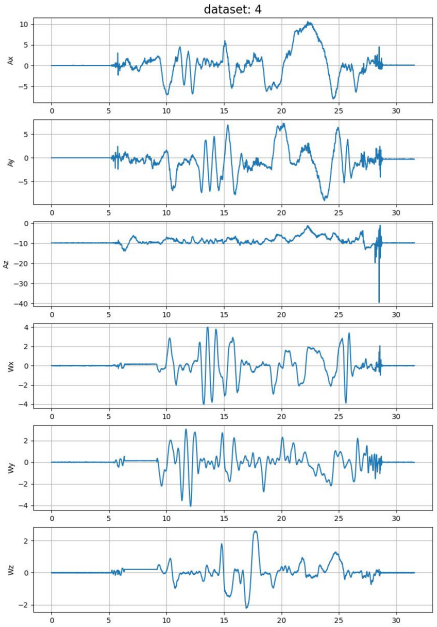


Figure 14: Loss function curve during optimization

4.4 dataset 4

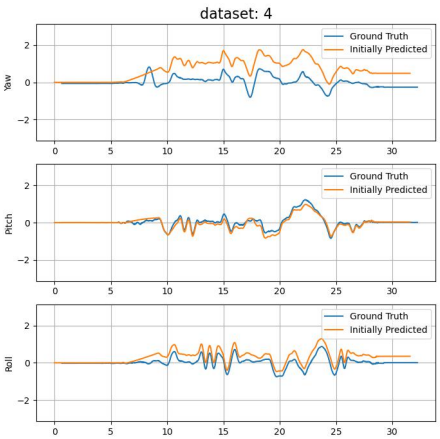


(a) Raw

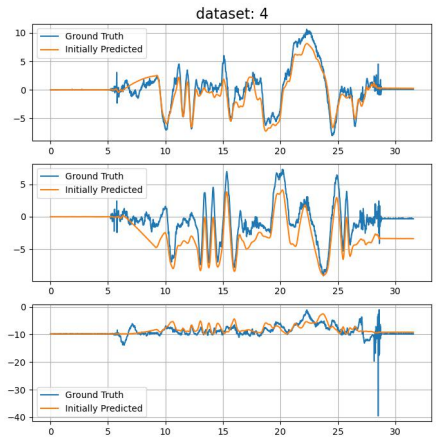


(b) Calibrated

Figure 15: IMU-data

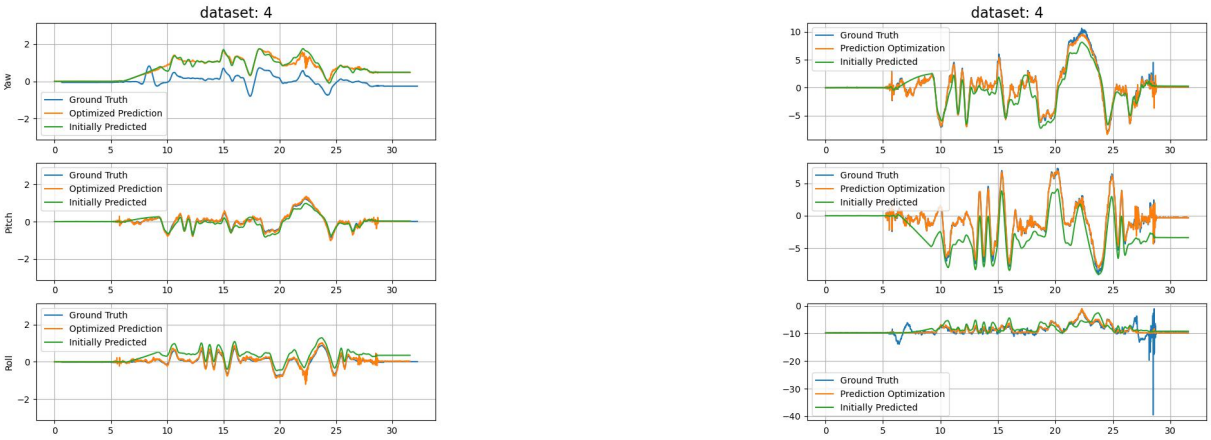


(a) Euler



(b) Acceleration

Figure 16: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 17: Euler and rotated acceleration after optimization

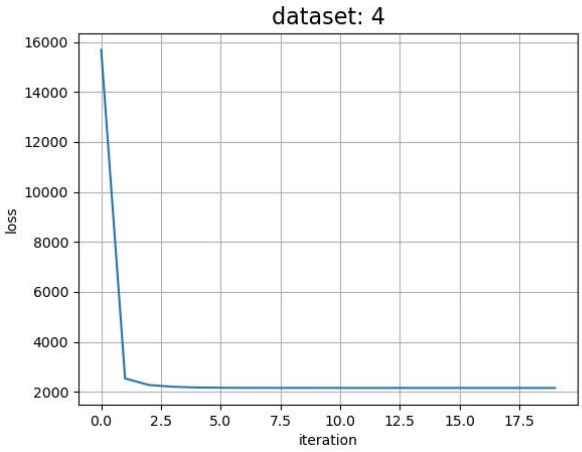
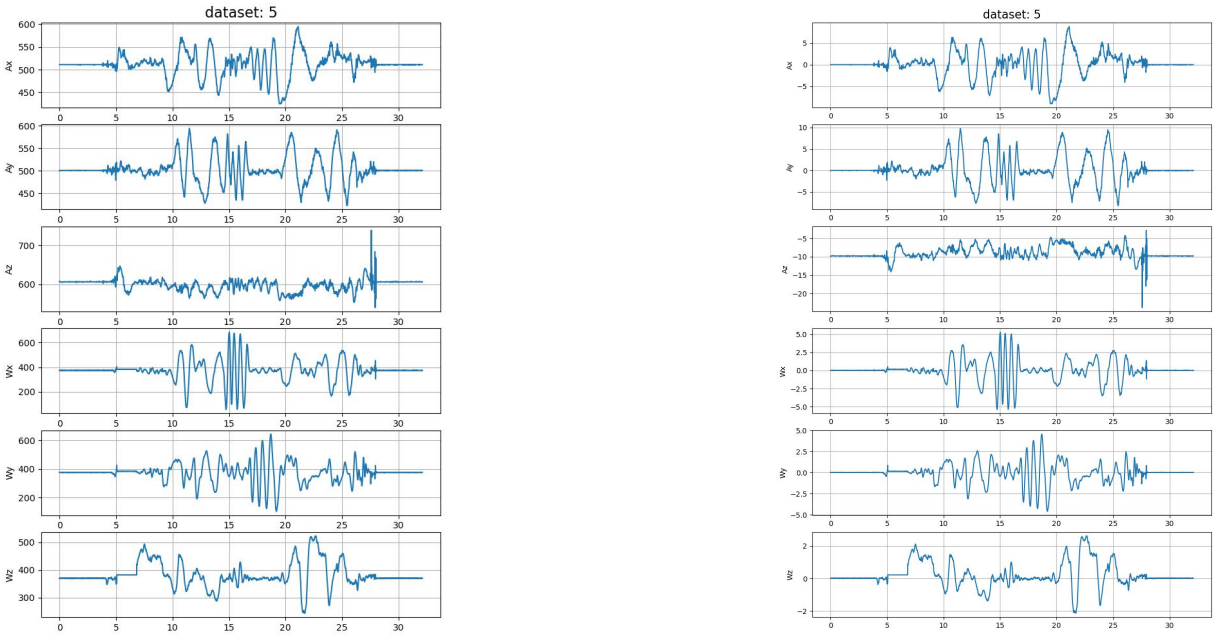


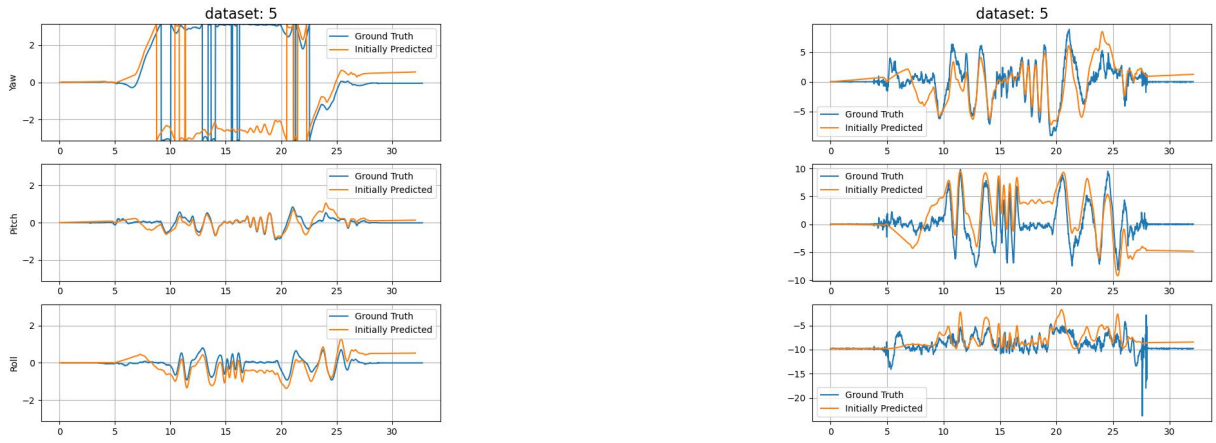
Figure 18: Loss function curve during optimization

4.5 dataset 5



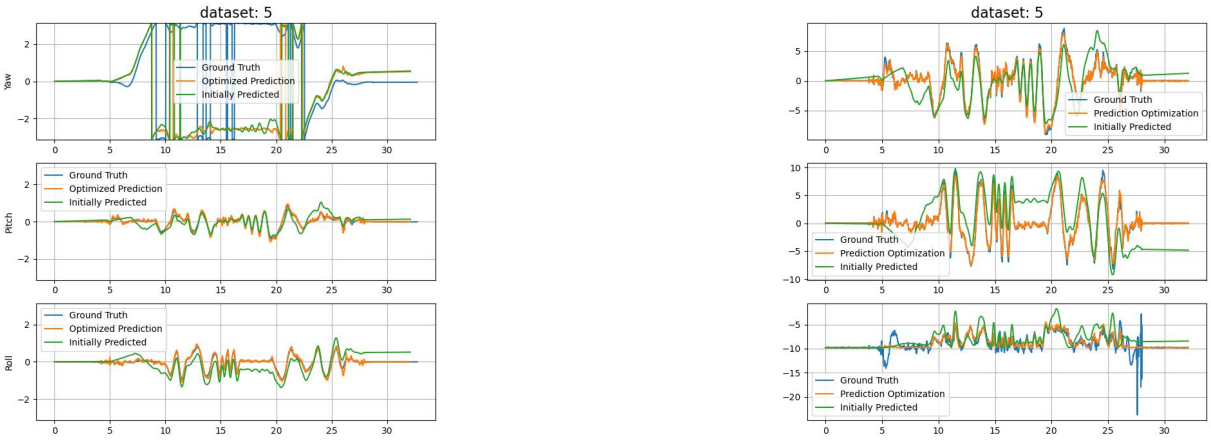
(a) Raw (b) Calibrated

Figure 19: IMU-data



(a) Euler (b) Acceleration

Figure 20: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 21: Euler and rotated acceleration after optimization

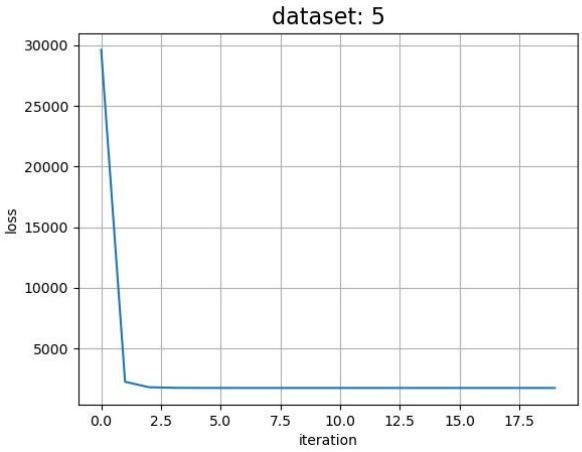
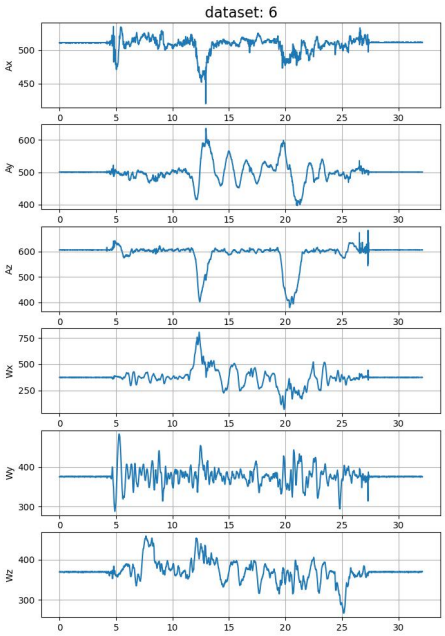
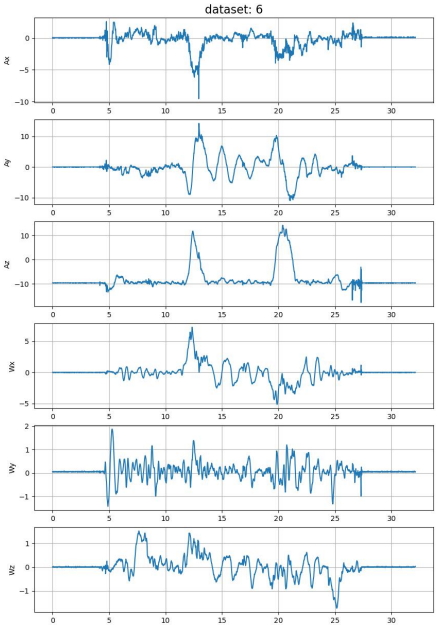


Figure 22: Loss function curve during optimization

4.6 dataset 6

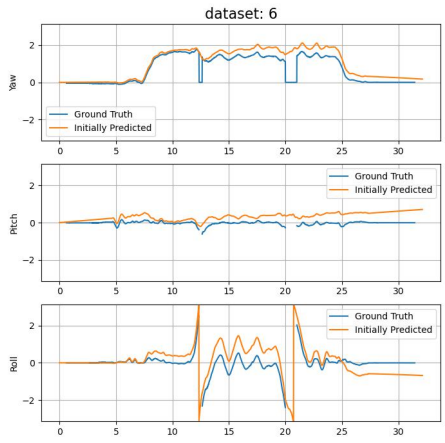


(a) Raw

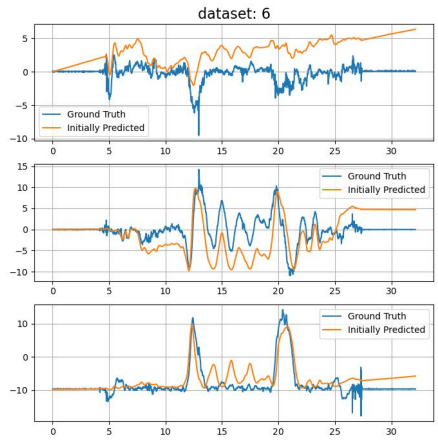


(b) Calibrated

Figure 23: IMU-data

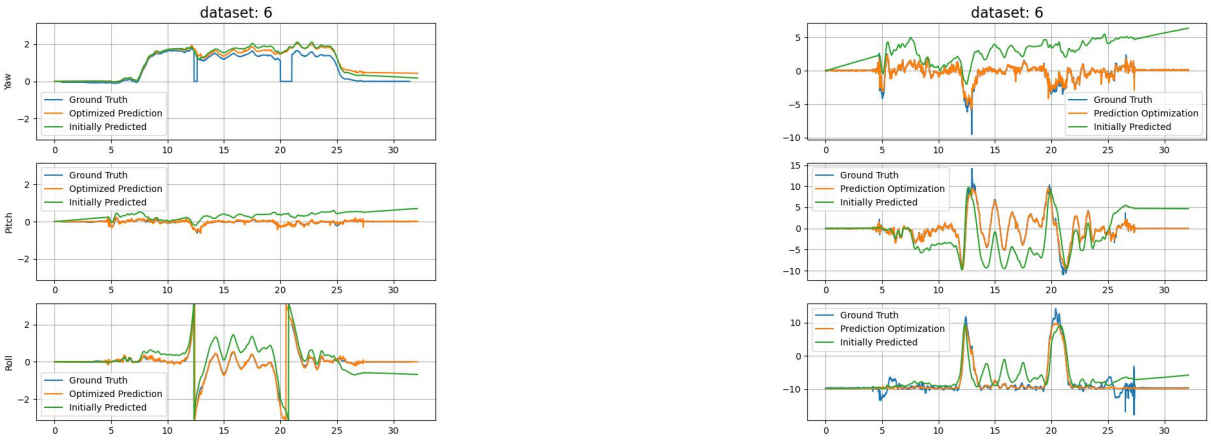


(a) Euler



(b) Acceleration

Figure 24: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 25: Euler and rotated acceleration after optimization

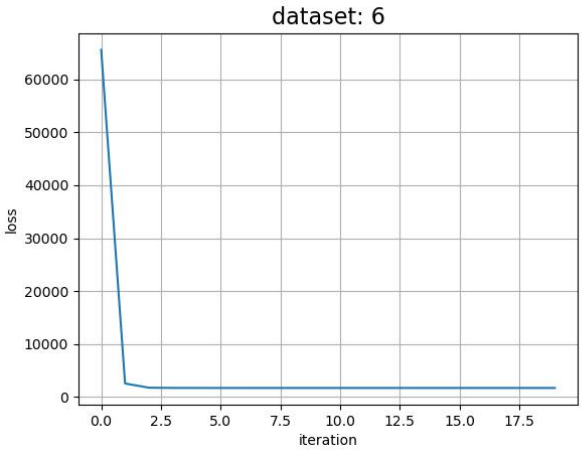
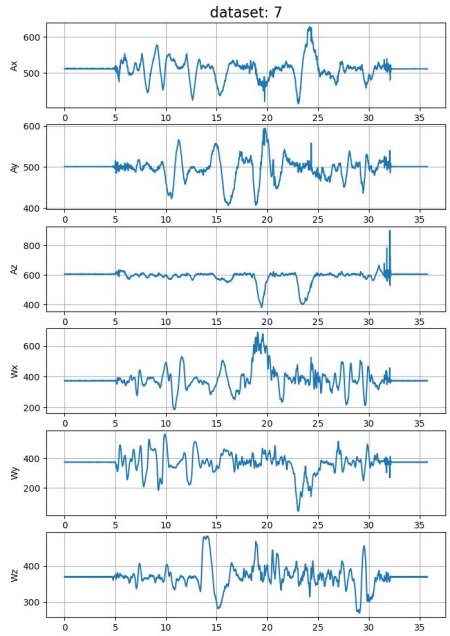
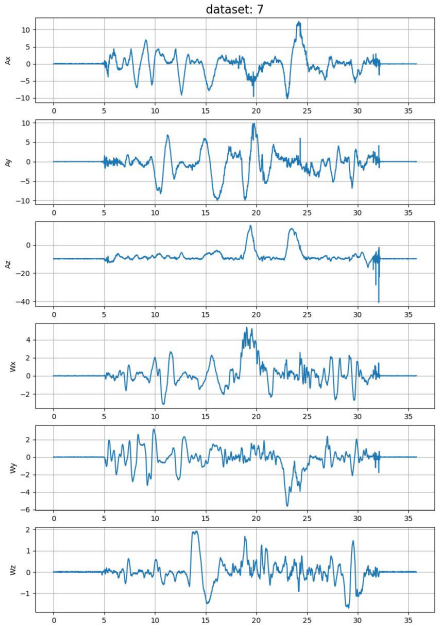


Figure 26: Loss function curve during optimization

4.7 dataset 7

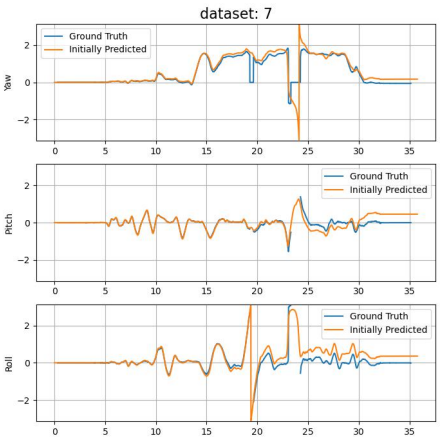


(a) Raw

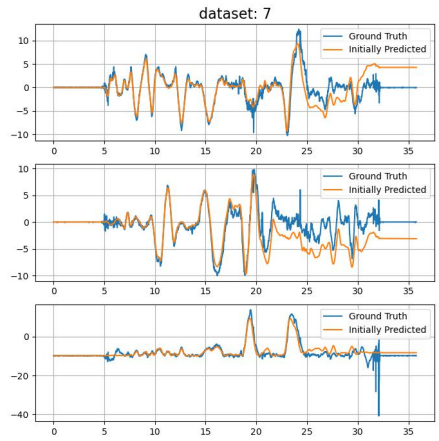


(b) Calibrated

Figure 27: IMU-data

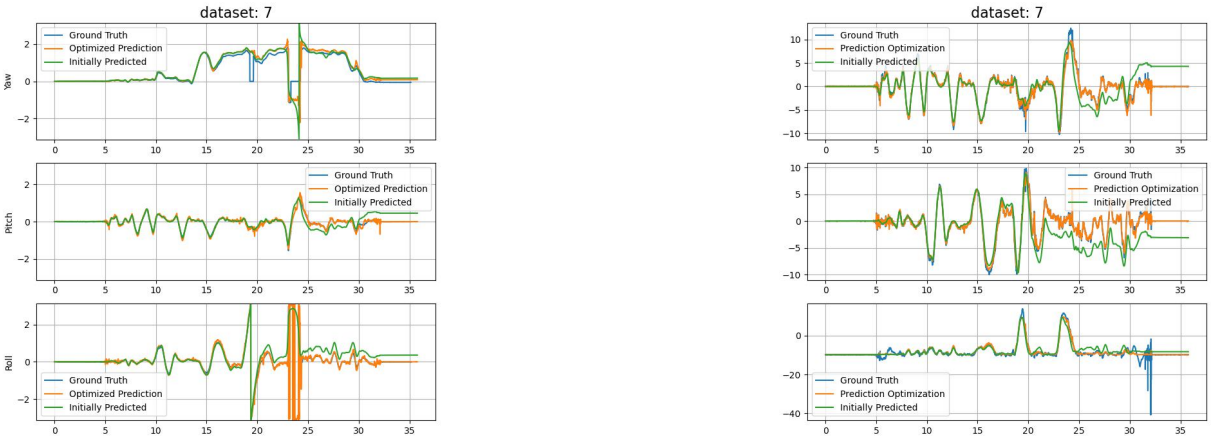


(a) Euler



(b) Acceleration

Figure 28: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 29: Euler and rotated acceleration after optimization

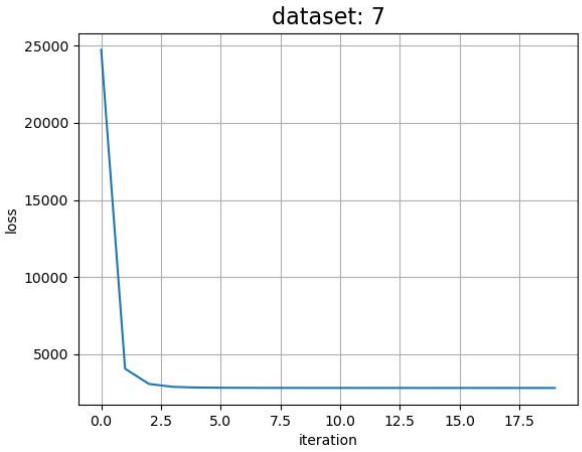
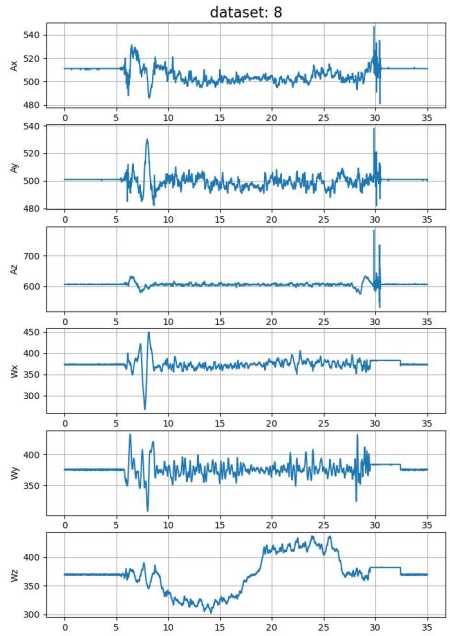
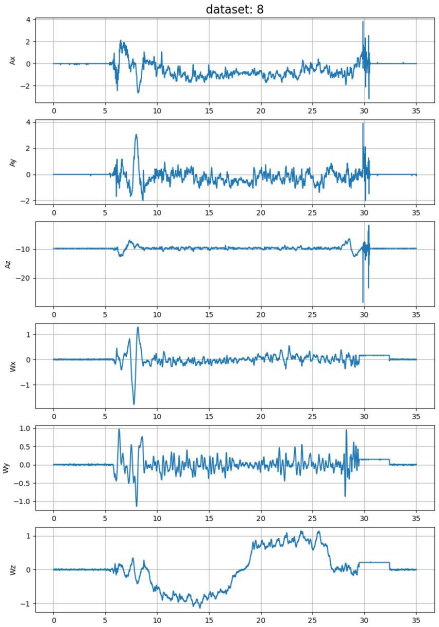


Figure 30: Loss function curve during optimization

4.8 dataset 8

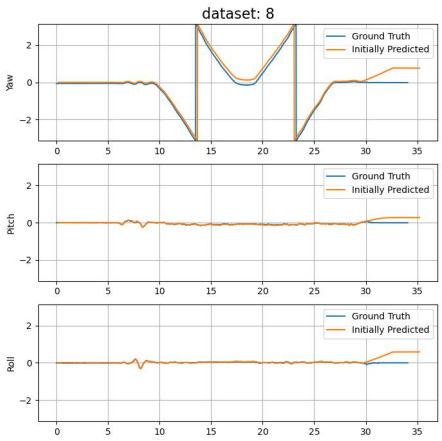


(a) Raw

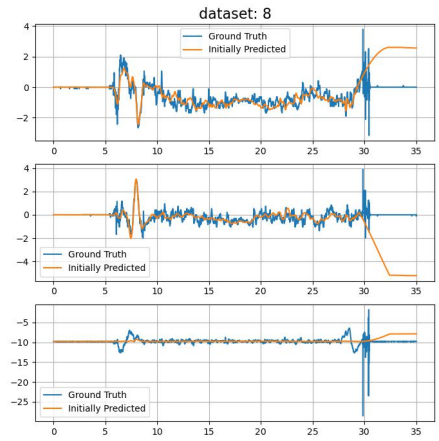


(b) Calibrated

Figure 31: IMU-data

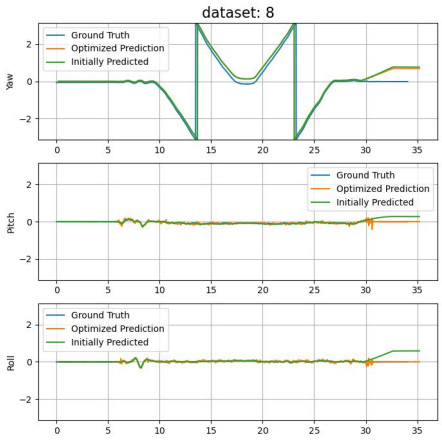


(a) Euler

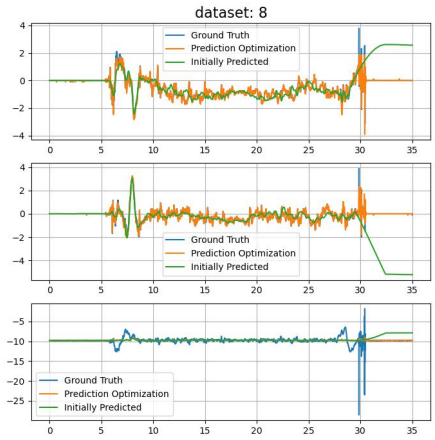


(b) Acceleration

Figure 32: Euler and rotated acceleration after initialization



(a) Euler



(b) Acceleration

Figure 33: Euler and rotated acceleration after optimization

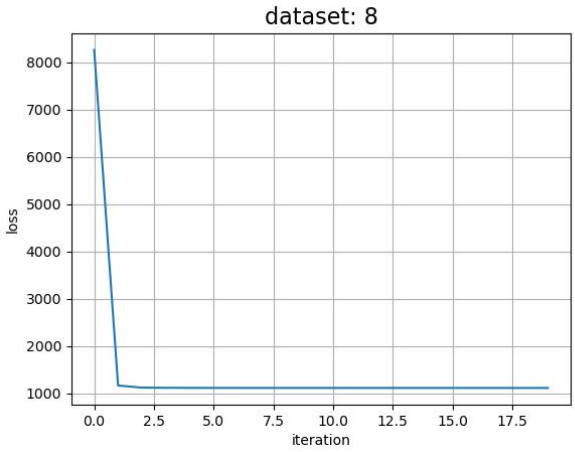
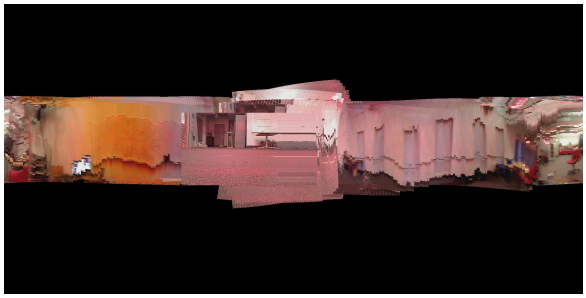


Figure 34: Loss function curve during optimization

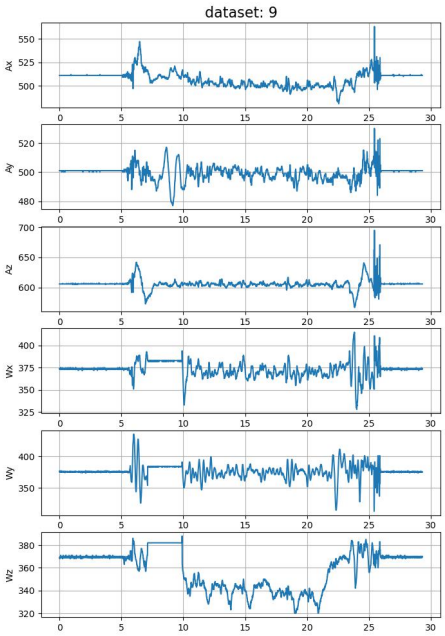
(a) VICON



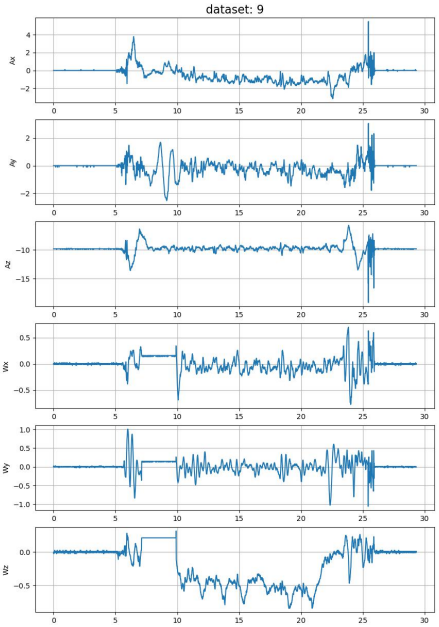
(b) Optimizations

Figure 35: Panorama Image with Rotations from VICON and optimizations

4.9 dataset 9

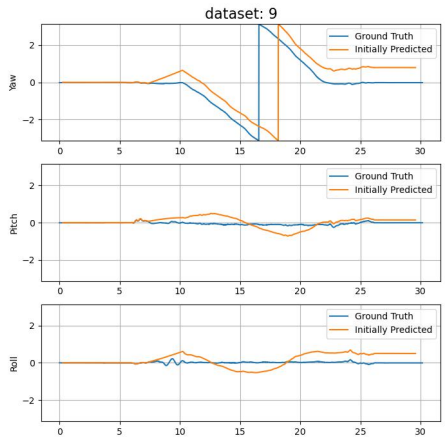


(a) Raw

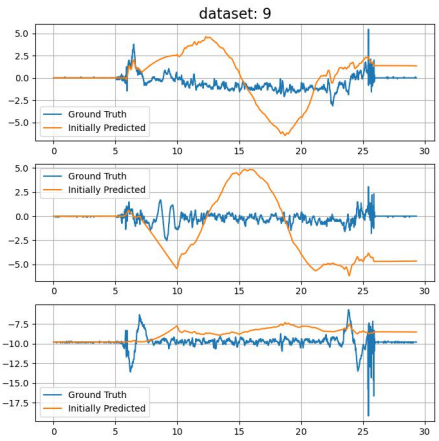


(b) Calibrated

Figure 36: IMU-data

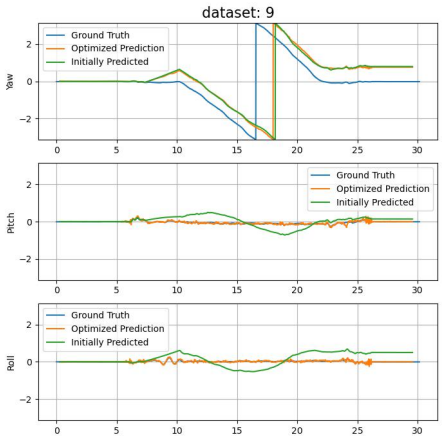


(a) Euler

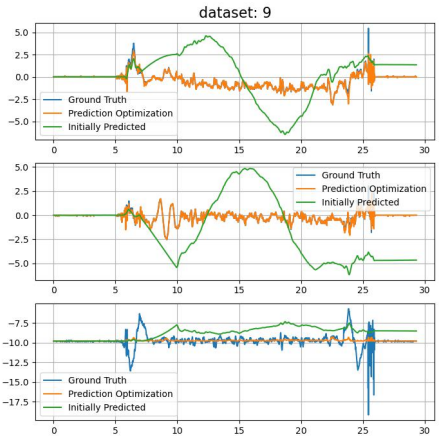


(b) Acceleration

Figure 37: Euler and rotated acceleration after initialization



(a) Euler



(b) Acceleration

Figure 38: Euler and rotated acceleration after optimization

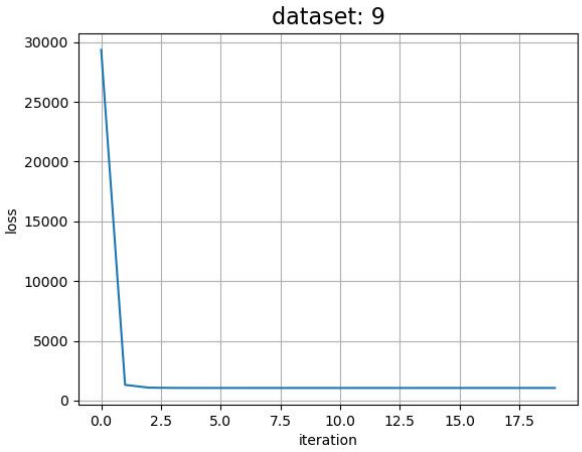
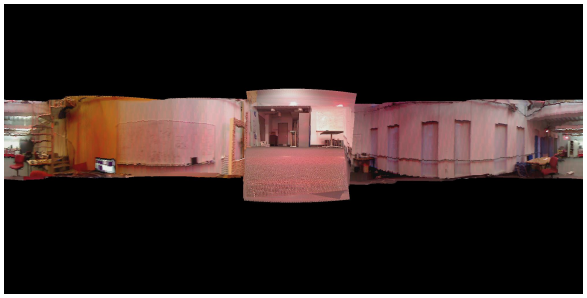


Figure 39: Loss function curve during optimization

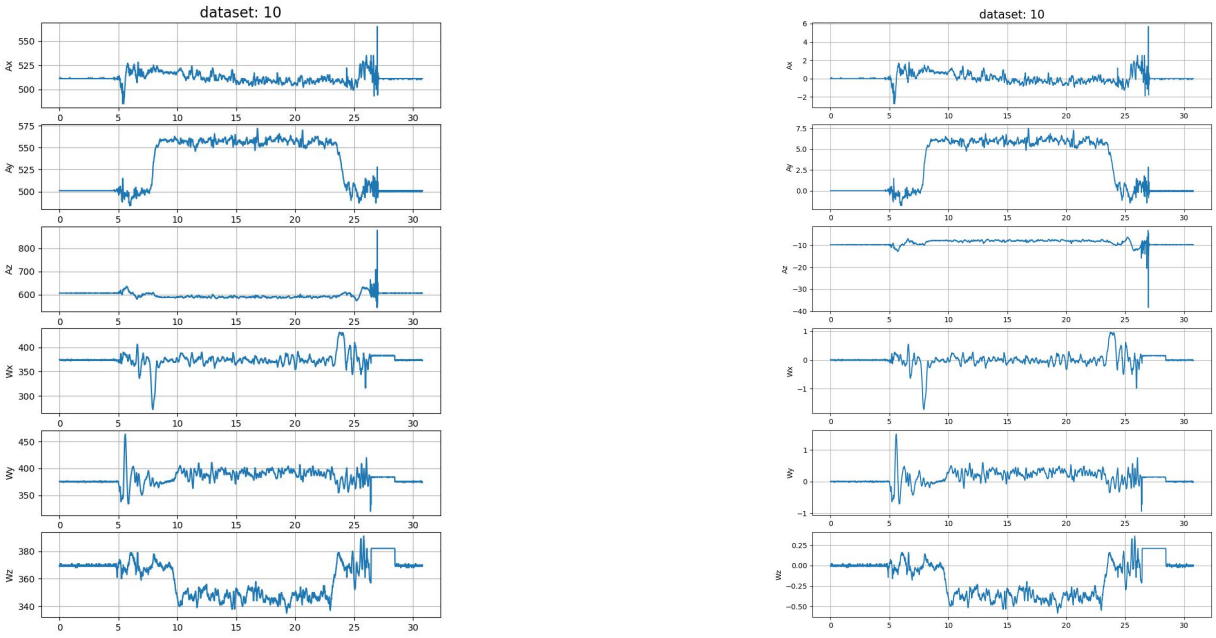


(a) VICON

(b) Optimizations

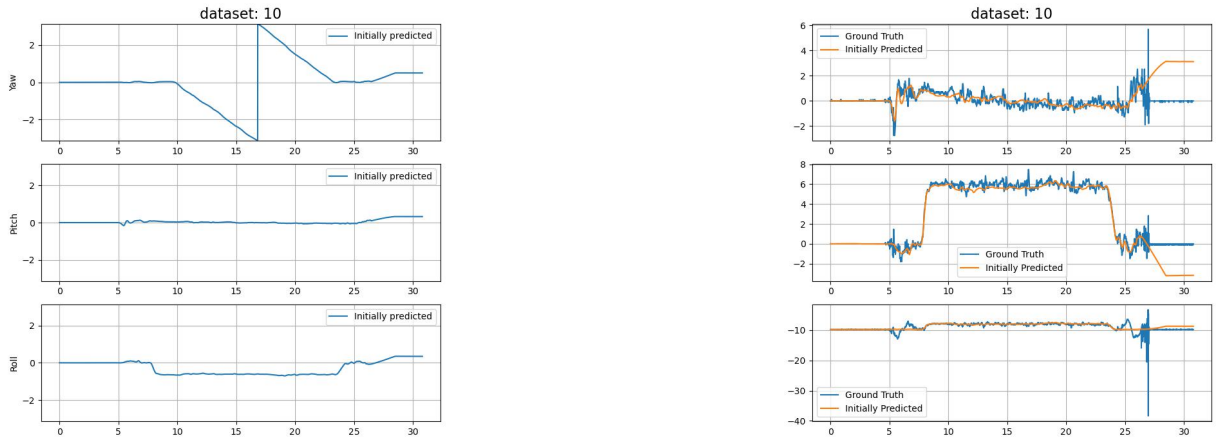
Figure 40: Panorama Image with Rotations from VICON and optimizations

4.10 dataset 10



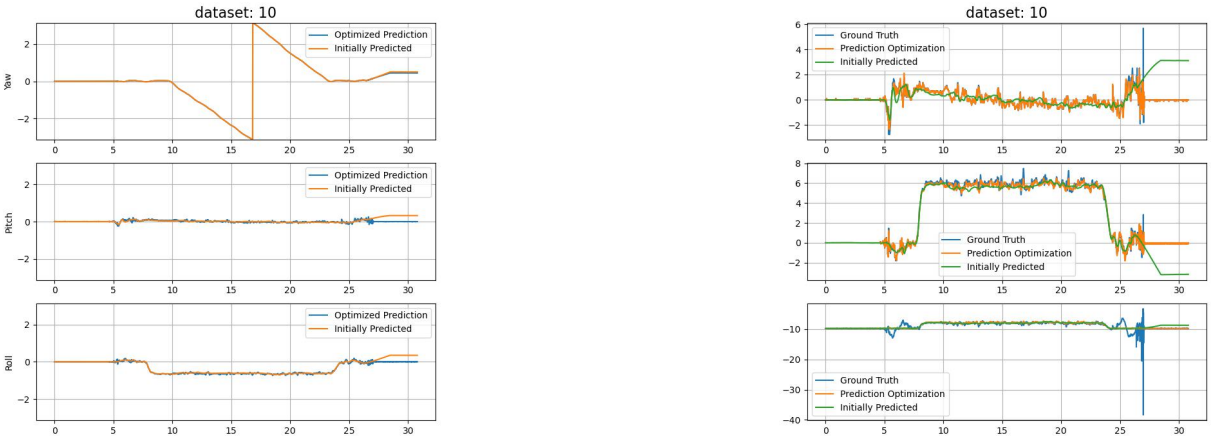
(a) Raw (b) Calibrated

Figure 41: IMU-data



(a) Euler (b) Acceleration

Figure 42: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 43: Euler and rotated acceleration after optimization

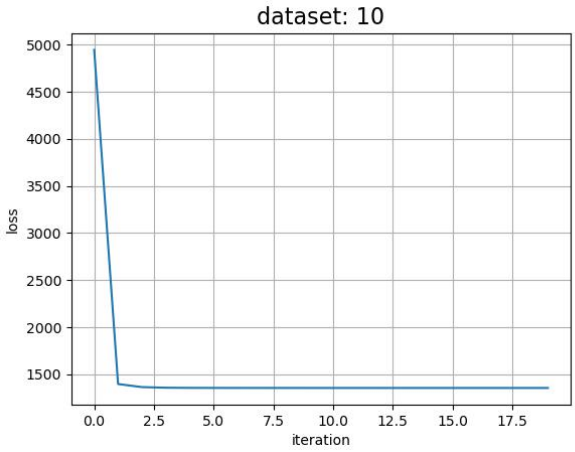
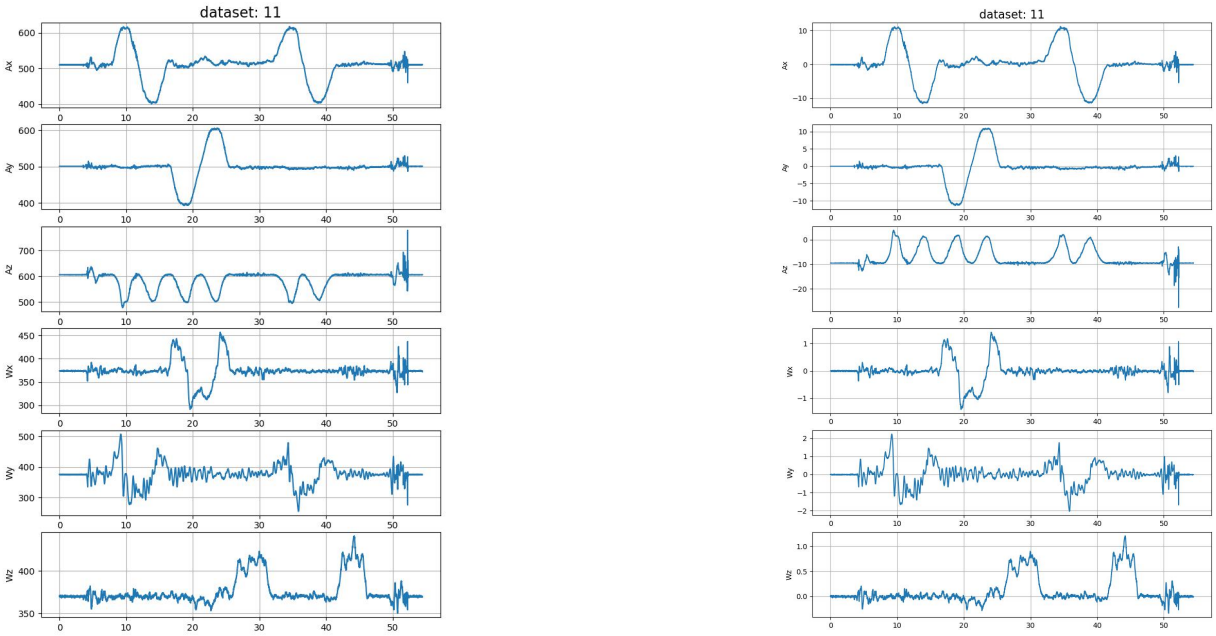


Figure 44: Loss function curve during optimization

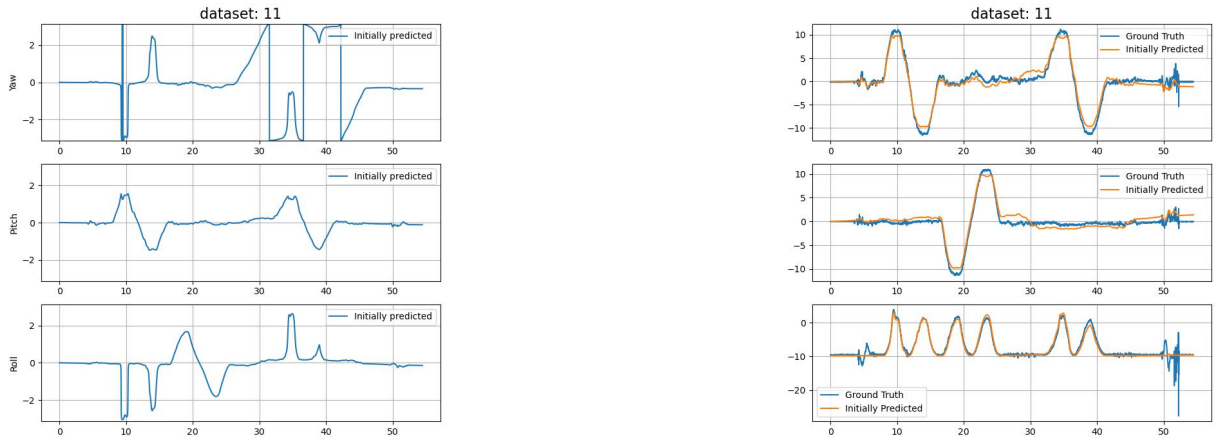
Figure 45: Panorama Image with Rotations from optimizations

4.11 dataset 11



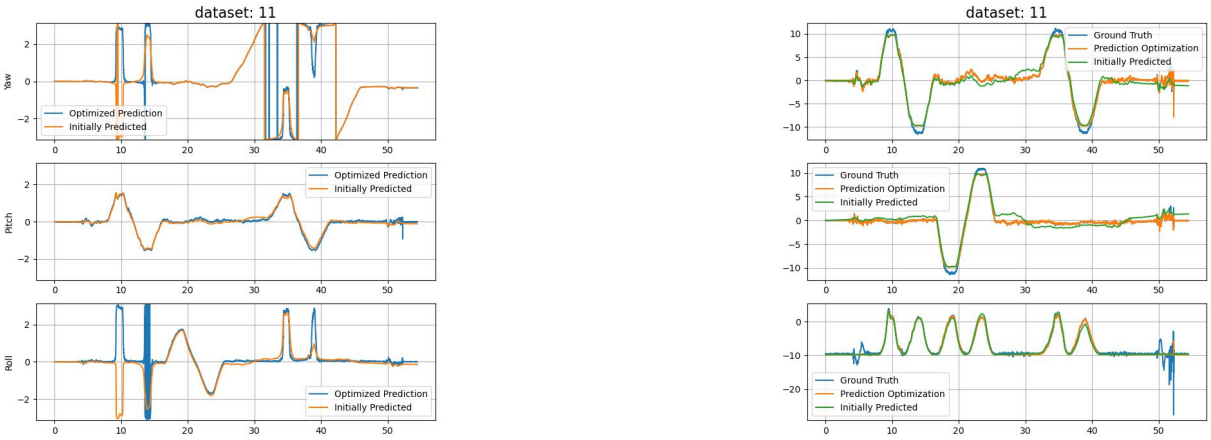
(a) Raw (b) Calibrated

Figure 46: IMU-data



(a) Euler (b) Acceleration

Figure 47: Euler and rotated acceleration after initialization



(a) Euler (b) Acceleration

Figure 48: Euler and rotated acceleration after optimization

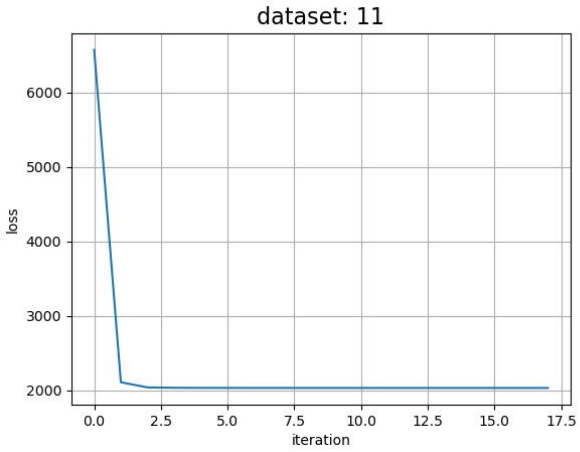


Figure 49: Loss function curve during optimization

Figure 50: Panorama Image with Rotations from optimizations

4.12 discussion

The entire calibration, training process, and panorama creation are completed in under 2 minutes. This rapidity can be attributed to two factors:

1. A high-speed deviation resulting from a batch perspective when crafting functions, thereby avoiding the use of loops.

2. The utilization of a novel optimization technique involving the calculation of the deviation of the loss function concerning α.

Across all datasets, the initial epoch yields a significant reduction in the cost function, facilitating swift convergence to a satisfactory local minimum. In a prior implementation, I employed a different optimization method involving the application of the gradient of the cost function with respect to the Quaternion array, followed by normalization. However, this method proved to be markedly slower in terms of convergence and less accurate compared to the current approach.
According to the documentation, since the object undergoes pure rotation, the acceleration should only reflect the force of gravity, resulting in an acceleration norm equal to g. However, upon plotting the norm of calibrated IMU data, I observed instances where the IMU registered accelerations greater than g. Since rotation preserves vector norms, this discrepancy couldn't be attributed to rotation alone. I suspect these anomalies stem from additional forces or measurement errors. Such cases could significantly impact the observation model, as no Quaternion should increase the norm of gravity. Through my plots, it becomes evident that my optimized Quaternions approach the expected value of g in these scenarios.
I find that my method performs admirably in terms of speed, yet it is heavily reliant on initializations and may become ensnared in local minima. In all cases, I've observed substantial enhancements in both Euler angles and acceleration projection. However, achieving complete alignment between these two objectives remains elusive. Regarding Euler angles, as they are not directly incorporated into the loss function, my model encounters difficulties in precisely fitting them. On the other hand, with regards to acceleration, I believe that rapid fluctuations and occurrences of acceleration norms exceeding g pose challenges in fitting their curves accurately.
The panorama figures generated for the training datasets, which include camera data, exhibit a close resemblance to the panoramas produced using ground truth rotation data from Vicon. This alignment underscores the effectiveness of the training pipeline. Additionally, upon examining the output panorama for the test set, it appears that my algorithm demonstrates robustness, further affirming its reliability across different datasets.
To further enhance the accuracy of the Quaternions, incorporating feedback from camera images could prove invaluable. This approach becomes particularly relevant when dealing with potentially noisy data from IMU. By leveraging alignments in the image domain, we can effectively address this issue. For instance, in dataset 9, there are periods, such as between seconds 7 and 10, where the Vicon ground truth indicates no rotation. However, the Quaternion still exhibit changes, likely attributable to noisy IMU signals. By integrating feedback from camera images during such instances, we can refine our Quaternion estimates and improve overall accuracy.