ECE 276A Project 1: Motion Models and Constrained Optimization

Justin Kane-Starr / A15554881

Abstract-In our first project for ECE 276A we are attempting to use primitive data and motion models to extract different state information from the vehicle we are attached to. This is primarily done by fitting quarternion motion models with angular acceleration and velocity readings from an IMU to a constrained optimization. We then display our fitted readings to a panorama to visually identify the effectiveness of our approach.

I. Introduction

This project begins with a problem statement and an initial set of data. Given a set of noisy IMU readings and an initial state is it possible to find the pose of our robot frame at any point in time? After optimizing our readings, can we create an accurate panorama of the space from images taken during this motion segment? It is intended for us to use topics learned from class to solve these two problems as well as having the freedom to approach them in other ways we see fit that may improve the model. Throughout this report, I will state my findings.

II. Problem Formulation

In this project we have three sensors: an IMU recorder, a VICON and a camera. These sensors record angular acceleration/velocity, orientation and images respectively. From the recordings of the IMU we are to reconstruct the ground truth data, tested against the vicon, and use this information to create a panorama.

Our initial conditions of the motion model that we use is a pose of

$$q_0 = [1, 0, 0, 0]$$

We then use the angular velocity and timestamps from the IMU in the following motion model to predict our next state in the sequence.

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

Where:
$q_{(t+1)}$ is the pose in the next time stamp
q_t is the pose in the current time stamp
τ_t is the time stamp difference
ω_t is the current angular velocity

The goal is to optimize our IMU data to match the VICON data in the first half of this project. We optimize this according to the cost function:

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

Where:

$$h(q_t) := q_t^{-1} * [0, 0, 0, 10] * q_t$$

The first summation of this cost function finds the motion model error and the second summation finds the angular acceleration error from our motion model and the IMU readings.

Our constrained optimization is of the following set

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

We constrain our resulting quarternions at all time stamps to ensure that the resulting set is not normalized.

III. Technical Approach

We split the work of this problem into three steps. The first of the three is implementing the motion model equations. This required lots of preprocessing as we had to take the IMU readings and adjust them to radial terms; it involved reading the data and moving it to a signed datatype, flipping the x and y readings then converting the electrical impulses to radial terms with this scalar multiple.

$$\text{value} = (\text{raw} - \text{bias}) * \text{scale factor}$$

$$\text{scale factor} = \text{Vref}/1023/\text{sensitivity}$$

After preprocessing the IMU data, it was fed through the motion model for all datasets and all timesteps. Onward I will address this motion model set of data as the "raw set." The raw set was a list of quarternions spanning the recording time and was converted yaw, pitch and roll over time that will be graphed and compared later.

Once we had a working motion model and a full set of the raw IMU data it was time to optimize the dataset according to the cost function as described in **II. Problem Formulation**. We achieve this by displaying using gradient descent on on the objective function C(q). This is done in the form

$$q_{t+1} = q_t + \alpha \nabla_{q_t} C(q_{1:T})$$

Where:

α is the learning rate.

In my experience the best learning rate over 10 epochs, a limit I had to impose due to unoptimal code, a learning rate of .05 yielded the best results over all datasets.

The implementation of the gradient descent algorithm relied heavily upon autograd libraries in python. In the early stages JAX was used however the lack of GPU and understanding of properly setting up Just In Time functions made the code exceptionally inefficient. The final product utilized autograd.numpy open source libraries. I had attempted but was unsuccessful without assistance to analytically solve for the gradient of the cost function as derivations of quarternion derivatives was much more complex than initially thought.

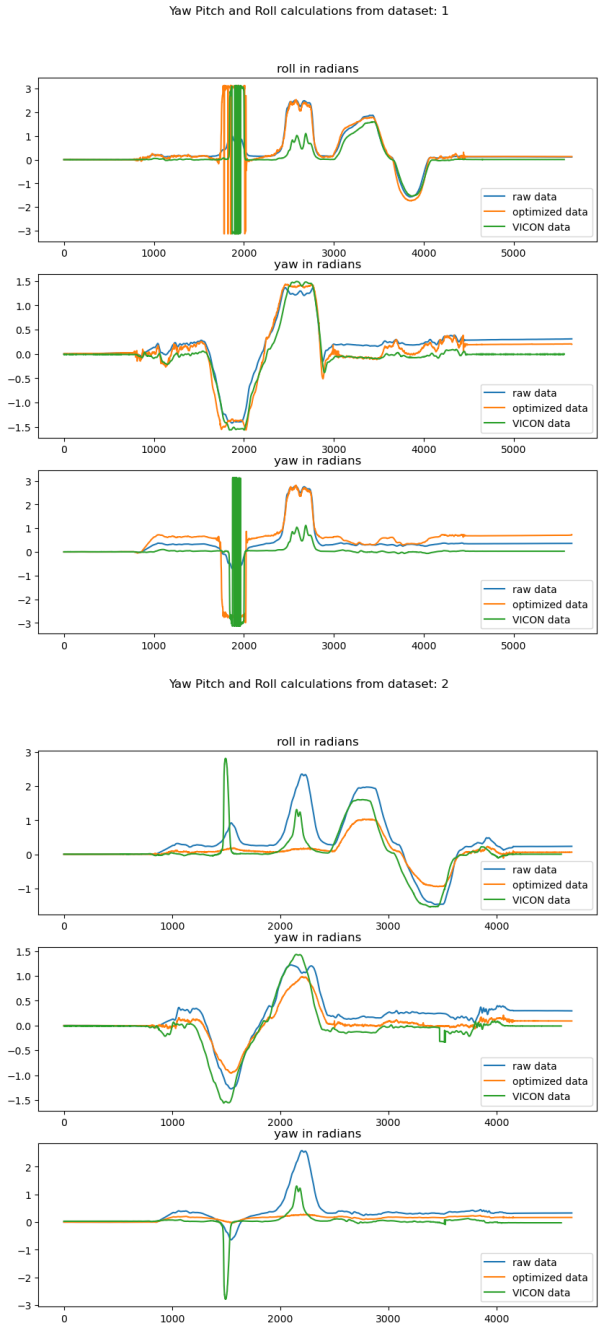
Gradient descent is an unconstrained optimization technique and as suggested in the document and in class we normalized the quartanion set after every epoch. However, in the early stages of implementation I attempted to solve this problem by applying a gradient ascent lagrangian of the following form.

$$\lambda_{t+1} = \lambda_t + \nabla_{\lambda_t} L(q_{1:T}, \lambda_{1:T})$$

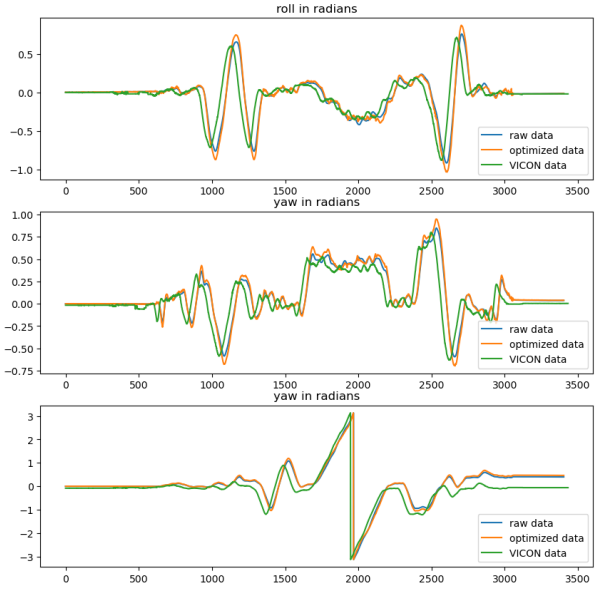
I believe this would've yielded better results as the constrained would be implicitly applied into the gradient ascent problem.

The resulting plots show the yaw, pitch and roll from each of the datasets for the raw data, optimized data and ground truth if available.

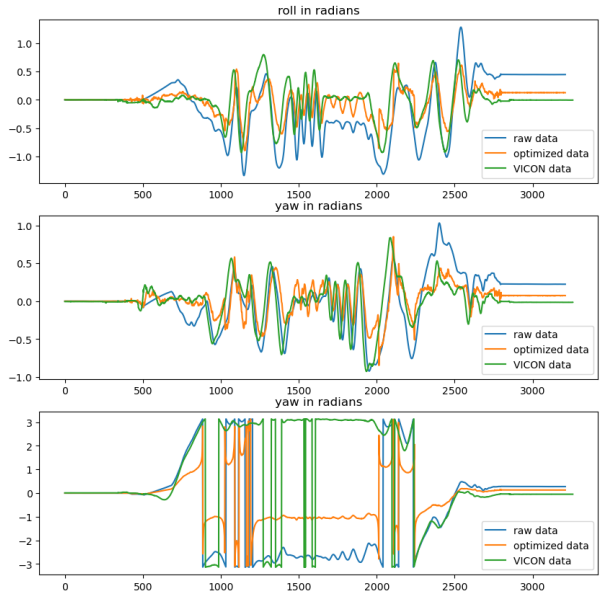
Yaw, Pitch, Roll Results



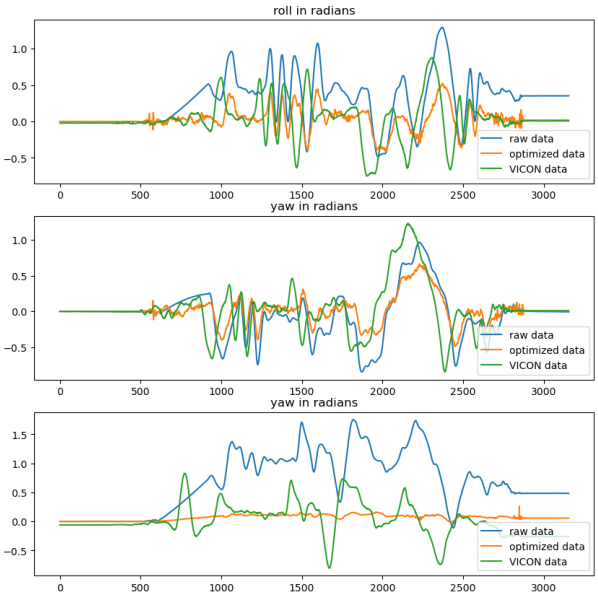
Yaw Pitch and Roll calculations from dataset: 3



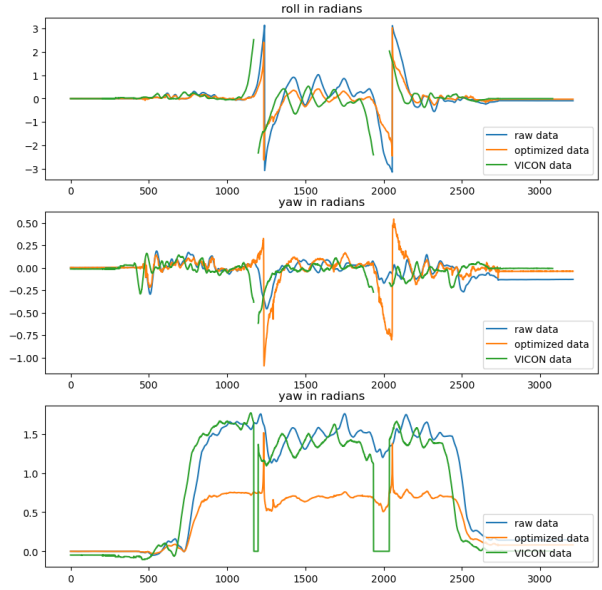
Yaw Pitch and Roll calculations from dataset: 5



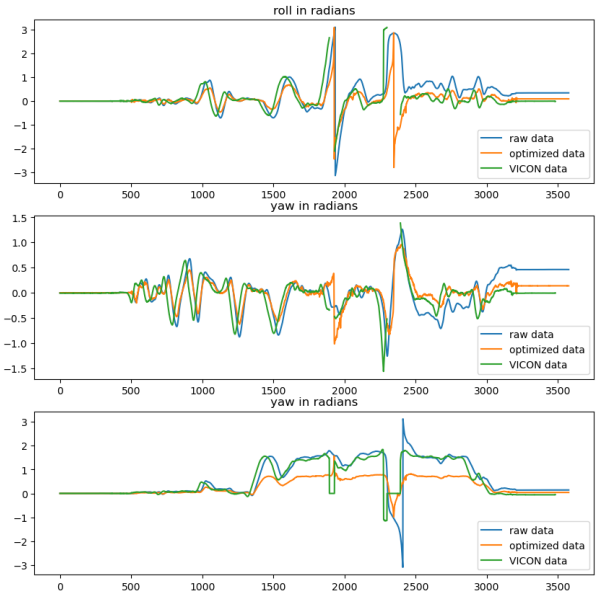
Yaw Pitch and Roll calculations from dataset: 4



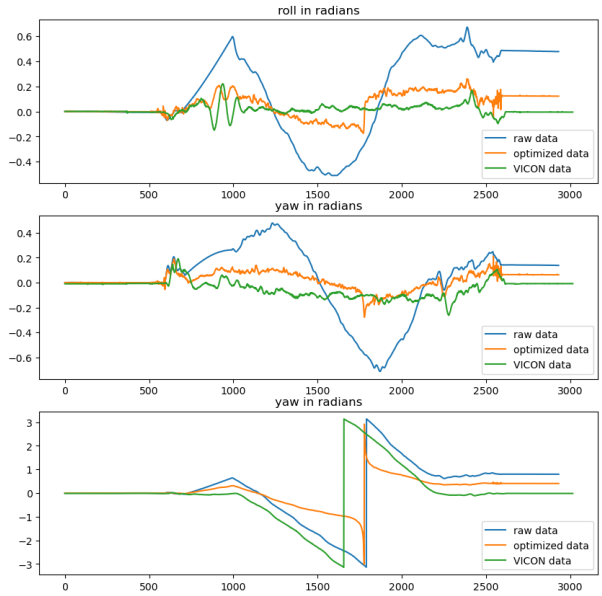
Yaw Pitch and Roll calculations from dataset: 6



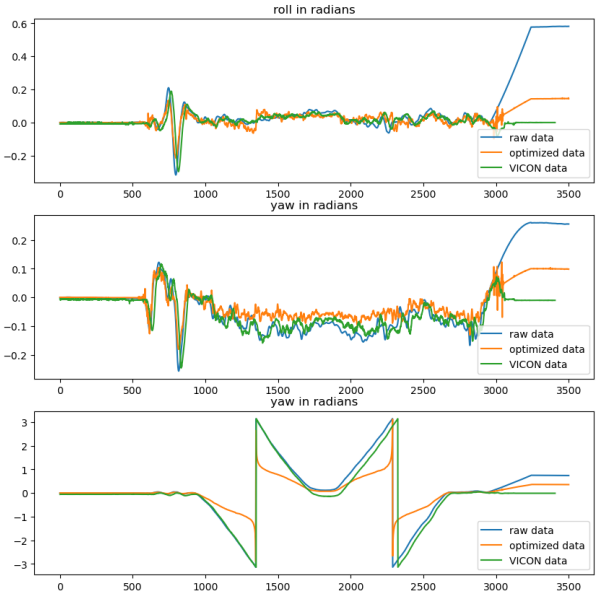
Yaw Pitch and Roll calculations from dataset: 7



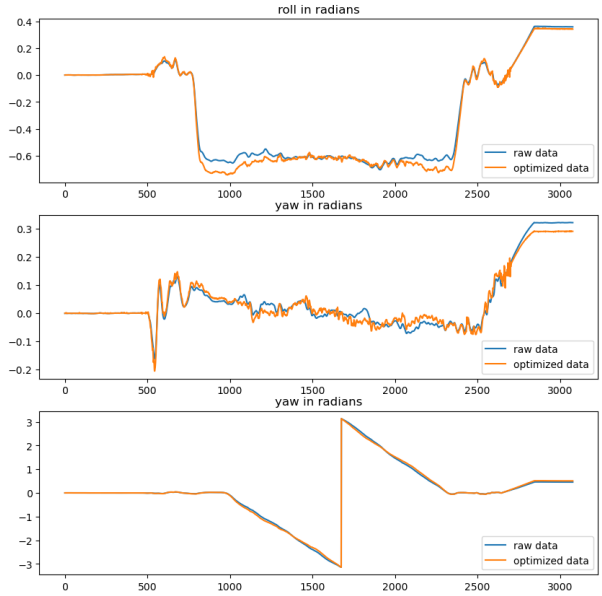
Yaw Pitch and Roll calculations from dataset: 9

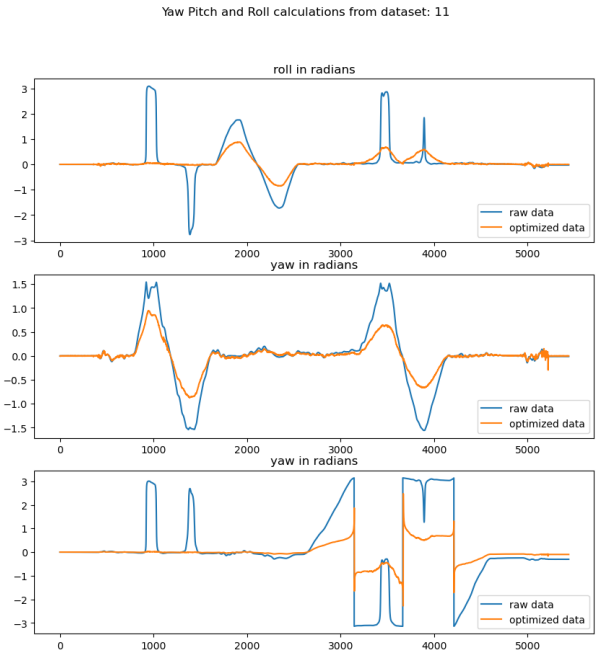


Yaw Pitch and Roll calculations from dataset: 8



Yaw Pitch and Roll calculations from dataset: 10





The results were not great for the optimized dataset. I believe this is because our motion model is quite primitive, not taking into account the acceleration that we're recording, and I did not implement any denoising algorithms into the preprocessing. Regardless, we have our dataset and can now attempt to create panoramas from our readings.

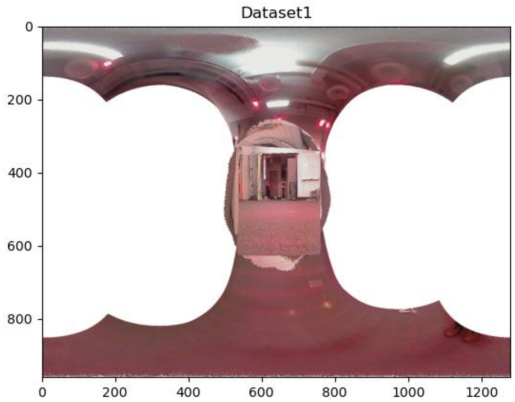
IV. Panorama Generation

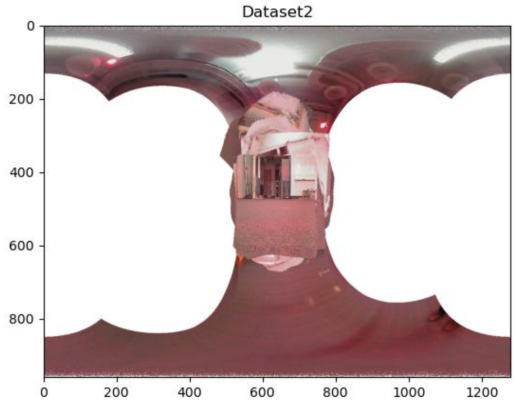
The panoramas were created using the algorithm given to us by the professor at the end of lecture 6. This involves projecting the optimized quarternions data from its ideal sphere to a cylinder, we then "unwrap" this cylinder to create a panorama. Our longitude and latitude values for this camera are 60 and 45 degrees respectively.
The formula for sphere to cylinder conversion is the following equation for our 960x1280 resulting photo.

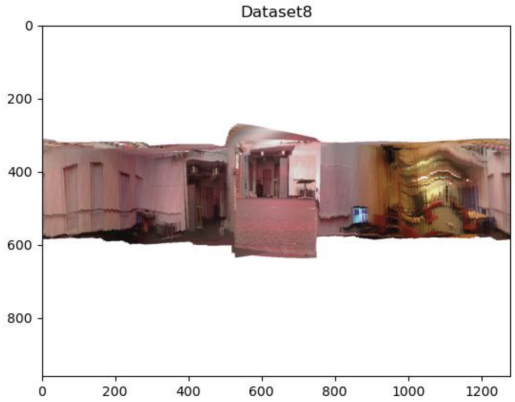
$$[x, y] = [(\phi + (\pi/2)) \times \frac{960}{\pi}, (\lambda + \pi) \times \frac{1280}{\pi}]$$

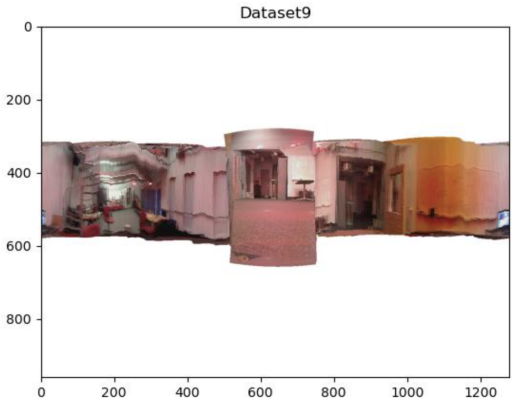
The results from our panorama are shown below.

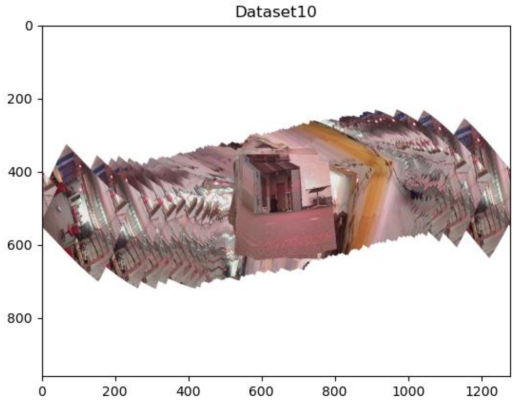
Panorama Results

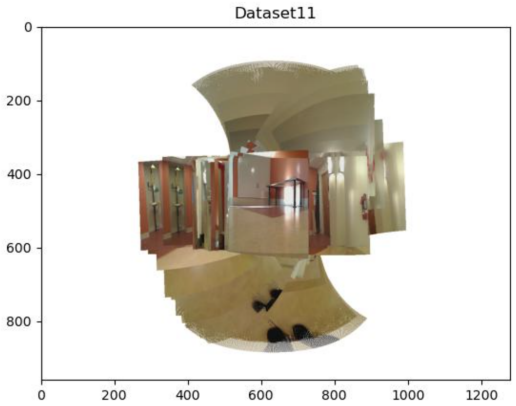












Concluding Thoughts

If given some more time and a bit more experience with analytically manipulating quarternions, I believe the pose estimation portion of this project could have yielded some excellent results. However, I feel that the panorama portion requires a lot more insight to improve especially as it wasn't discussed extensively in class until the lecture prior before the due day. It was an interesting project nonetheless but I do believe that a recorded office hour on code optimization for the gradient descent algorithm would be greatly appreciated by many of the students including myself. The panoramas took around 3 hours of computation time to complete due to a lack of knowledge of parralelizing and the quarternion gradient descent across all datasets took an hour.