ECE276A Project 1: Orientation Tracking

Richard Oliveira
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A

I. Introduction

In the context of autonomy, more specifically in robotics, the idea of being able to control and track the position and motion of a robot becomes crucial as the use cases and the environments in which these systems are being used become increasingly more sophisticated. In autonomous vehicles for example, we wish to track the car while it autonomously navigates through the streets of a city, such that it avoids pedestrians, or buildings, or other cars. Other examples such as warehouse robotics, health care robotics, all require some sort of tracking mechanism such that the systems do not produce undesirable outcomes in their respective applications. Therefore the need to control and track the position and motion of these systems involves safety concerns. In this report, we present a method where we are able to track the orientation of a stationary robot undergoing pure rotation, by minimizing an objective function which represents the error between measured sensor data and model predictions. This mathematical formulation will lead us to be able to track the orientation of the robot fairly well when compared to ground truth data.

II. Problem Formulation

Consider a stationary robot undergoing pure rotation over discrete time steps t. Furthermore, the robot has an associated angular velocity $\boldsymbol{\omega}_t \in \mathbb{R}^3$.

First, we define two different frames of reference. A so called *world frame* $\{W\}$, which remains fixed throughout this experiment, and a *body frame* $\{B\}$ which is attached to the robot and it is rotating with the robot as well.

From a mathematical point of view, one of the main challenges is being to able to represent the rotations of a robot in a consistent manner, without running into issues such as *Gimbal locks* [3].

An extremely important mathematical tool which will be used throughout this report is the *quaternion* $q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$, a, b, c, $d \in \mathbb{R}$, which is an extension of a complex number in four dimensions. More compactly, the quaternion can be written as $\boldsymbol{q} = [q_s \ \boldsymbol{q}_v]$, where $q_s = \text{Re}\{q\} = a \in \mathbb{R}$, and $\boldsymbol{q}_v = \text{Im}\{q\} = [b \ c \ d] \in \mathbb{R}^3$.

To represent a rotation at time t [3] we want $\boldsymbol{q}_t$ to belong in the following set $\forall t$:

$$\mathbb{H}_* = \{\boldsymbol{q}_t | \|\boldsymbol{q}_t\|_2 = 1\}$$

Now, we introduce an equation that describes *state evolution in time* (1) and a *measurement model* (2):

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

$$\boldsymbol{a}_t = h(\boldsymbol{q}_t) = \boldsymbol{q}_t^{-1} \circ [0 \ 0 \ 0 \ g] \circ \boldsymbol{q}_t \tag{2}$$

The equations mentioned above, will give us *model predictions*, where (1) will predict a rotation at the next time step $t+1$ given the current rotation $\boldsymbol{q}_t$, current angular velocity $\boldsymbol{\omega}_t$ and time difference between current and last time step τ_t. Equation (2) on the other hand, measures a three dimensional acceleration $\boldsymbol{a}_t$ at the current time.

As mentioned in the introduction, we will formulate an optimization problem, where the minimization of this objective function will lead to estimating an optimal trajectory of rotations $\boldsymbol{q}_{1:T}$ over T time steps. The sequence $\boldsymbol{q}_{1:T}$ is the trajectory that minimizes the following optimization problem, subject to the constraint (4):

$$\min_{\boldsymbol{q}_{1:T}} c(\boldsymbol{q}_{1:T}) \tag{3}$$

$$s.t \ \boldsymbol{q}_t \in \mathbb{H}_* \ \forall t \in \{1, \ \dots, T\} \tag{4}$$

Here the objective function $c(\boldsymbol{q}_{1:T})$ has the following form:

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

Now, the first term involves two different types of transformations: exponential and a logarithmic transformation. The first is performed by the function $f(\boldsymbol{q}_t, \tau_t, \boldsymbol{\omega}_t)$, where the input is an angle given by $\tau_t \boldsymbol{\omega}_t$. The function f takes the angle and transforms into the space of quaternions in such a way that the quaternion multiplication "$\circ$" can be performed. We also note that the angular velocity $\boldsymbol{\omega}_t$ is given to us by the IMU measurements which will be discussed later in the report. The logarithmic transformation on the other hand has the following inverse mapping $log : \mathbb{H} \to \mathbb{R}^3$, where the quaternion is mapped to the space of Euler angles. The second term $\|\boldsymbol{a}_t - h(\boldsymbol{q}_t)\|_2^2$, involves a difference between the IMU measurements that we have of $\boldsymbol{a}_t$ and the model prediction $h(\boldsymbol{q}_t)$. (2) represents a measurement of acceleration in the *body frame* $\{B\}$, where there is an absence of acceleration in the $x-y$ directions, but there is only an acceleration in the negative z - direction due to gravitational acceleration.

III. Technical Approach

We approach the problem in the following manner:

- *Orientation Tracking*
  - Data Pre-Processing
  - Optimization problem
- *Panorama*
  - Implementation
  - Projection

In the *Orientation Tracking* part we discuss how we process IMU the data prior to using it. In the second step we explain how we have approached the minimization of the cost function and what approaches have been used. Lastly in *Panorama* we discuss how we have created a panormaic view of the robot's orientation given the optimal estimates $\boldsymbol{q}_{1:T}$.

A. Orientation Tracking

1) Data Pre-Processing: In this step we access the IMU data and we calibrate it in such a way that it can be used for later. The IMU data consists of gyroscope measurements, which measure the angular velocity $\boldsymbol{\omega}_t \in \mathbb{R}^3$ at some time step t, and accelerometer measurements which measure acceleration $\boldsymbol{a}_t \in \mathbb{R}^3$ at some time step t. We also have UNIX time stamps, which indicate the time at which the IMU measurement was taken. The raw IMU data is organized according to Fig. 1 across N samples.
Additionally the raw IMU data, has the x and y acceleration measurements flipped. This was also corrected in the Data Pre - Processing step. Another issue in the raw IMU data is that both devices record non-zero numbers in the absence of rotation, therefore this is a bias that will be corrected. We choose the bias as the mean of the IMU measurements and we use the IMU reference sheets to establish the following sensitivities: Vref = 3300mV, sensitivity for accelerometer as 300 mV/g, sensitivity for the gyroscope as 3.33 mV/deg/s. The last measurement will later on be converted into radians. We also note that both devices have scale factors. The scale factor K is computed according to the following equation:

$$K = \frac{V_{ref}}{1023 * sensitivity} \tag{6}$$

Once the scale factor has been determined, we compute the new, calbrated IMU data point $\boldsymbol{\omega}_t^{new}$, $\boldsymbol{a}_t^{new}$ as follows, where $bias \in \mathbb{R}^6$:

$$(\boldsymbol{\omega}_t^{new}, \boldsymbol{a}_t^{new}) = K * ((\boldsymbol{\omega}_t^{raw}, \boldsymbol{a}_t^{raw}) - bias) \tag{7}$$

Fig. 1. IMU Data Format

2) Optimization Problem: In this part of the project we attempt to solve the Optimization problem (3), subject to the constraint (4). Before doing this, we will talk about the several implementation details of (1) and (2).

The term $exp\{[0 \ \frac{\tau_t}{\boldsymbol{\omega}_t}]\}$ was computed as follows [4]:

$$exp(\boldsymbol{q}+\delta) = e^{q_s+\delta}[cos(\|\boldsymbol{q}_v+\delta\|_2), \ \frac{\boldsymbol{q}_v+\delta}{\|\boldsymbol{q}_v+\delta\|_2} \ sin(\|\boldsymbol{q}_v+\delta\|_2)] \tag{8}$$

Where δ corresponds to a small perturbation to the order of 10^{-3} which is added element-wise to $\boldsymbol{q}$. This was done to avoid dividing vectors by zero-valued norms. A similar approach was done when implementing the logarithmic map. We have created a function which checks if the elements of $\boldsymbol{q}_v$ are less than 10^{-4}. If the condition is true then we compute the log map like so:

$$log(\boldsymbol{q}) = [log(\|\boldsymbol{q}\|_2) \ 0, \ 0, \ 0] \tag{9}$$

If the condition is not satisfied then we compute the log map as [4]:

$$log(\boldsymbol{q}) = [log(\|\boldsymbol{q}\|_2), \ \frac{\boldsymbol{q}_v}{\|\boldsymbol{q}_v\|_2} arccos(\frac{q_s}{\|\boldsymbol{q}\|_2})] \tag{10}$$

Now, since we wish to solve (3) such that we find a sequence of quaternions $\boldsymbol{q}_{1:T}$ that minimize the errors defined in (5), it is natural to solve this problem by using *Projected Gradient Descent* (PGD). But we notice that (3) has an equality constraint in (4), therefore we initially thought of re-defining the problem in terms of *Lagrange multipliers* [2]. The conversion is trivial. First we define a vector $\boldsymbol{g}(\boldsymbol{q}_{1:T}) = [\|\boldsymbol{q}_1\|_2 - 1, \ \dots \ , \|\boldsymbol{q}_T\|_2 - 1]^T \in \mathbb{R}^T$, then we define $\boldsymbol{\lambda} = [\lambda_1, \ \dots \ , \lambda_T]^T \in \mathbb{R}^T$.
Now by definition, the *Lagrangian* is:

$$L(\boldsymbol{q}_{1:T}, \boldsymbol{\lambda}) = c(\boldsymbol{q}_{1:T}) - < \boldsymbol{\lambda}, \boldsymbol{g}(\boldsymbol{q}_{1:T}) > \tag{11}$$

Where $< \boldsymbol{\lambda}, \boldsymbol{g}(\boldsymbol{q}_{1:T}) >= \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{q}_{1:T})$ is simply the standard *inner product*. Once the Lagrangian was defined, we aim to compute the following partial derivatives:

$$\nabla_{\boldsymbol{q}_j}\{L(\boldsymbol{q}_{1:T}, \boldsymbol{\lambda})\} = 0 \tag{12}$$

$$\nabla_{\boldsymbol{\lambda}}\{L(\boldsymbol{q}_{1:T}, \boldsymbol{\lambda})\} = 0 \tag{13}$$

such that:

$$\boldsymbol{q}^*_{1:T}, \boldsymbol{\lambda}^* \in \min_{\boldsymbol{q}_{1:T}, \boldsymbol{\lambda}} L(\boldsymbol{q}_{1:T}, \boldsymbol{\lambda}) \tag{14}$$

Unfortunately, the gradients in (12) and (13) yield complicated expressions that seem intractable when performing PGD. Even though the re-formulation of the problem with equality constraint in (4) is straightforward, the final expressions are intractable.

Since the *Langrange multipliers* approach failed, we turn our attention to regular PGD. Where the parameter of interest which minimizes (5) needs to belong in the set $\mathbb{H}_*$. We know

by definition that the regular *Gradient update step* at some step k is given by [6]:

$$\boldsymbol{q}_{1:T}^{k+1} \leftarrow \boldsymbol{q}_{1:T}^{k} - \alpha \nabla_{\boldsymbol{q}_{1:T}} \{c(\boldsymbol{q}_{1:T}^{k})\} \tag{15}$$

Once this step is performed, in order to ensure that $\boldsymbol{q}_{1:T}^{k+1} \in \mathbb{H}_*$ we simply divide $\boldsymbol{q}_{1:T}^{k+1}$ by its norm. This method seems to be a valid approach as well. We also note here that the step-size α was chosen to be constant at each iteration k ($\alpha = 10^{-3}$). An annealing parameter $T(k)$ (popular in Deep Learning) has been thought of and implemented but yielded poor results.

B. Panorama

In this step, we are working with RGB images with fixed size, of dimension 240x320. The resolution of the panorama is fixed to be 2000x3000. We have a camera mounted at the top of the robot frame, rotating with the robot itself. It is positioned at 10cm above the IMU device.

1) Implementation: We assume that the images lie on a sphere, and each pixel (i, j) of the 240x320 image has the following spherical coordinates [1]:

$$(i, j) \leftrightarrow (\lambda, \phi, r) \tag{16}$$

The depth r is arbitrary, but we choose $r = 1$ for convenience. Here λ represents the *longitude* and ϕ represents the *latitude* of the pixel [5]. Graphically, this representation is illustrated in Fig. 2. In terms of sign convention we assume negative angle clock-wise and positive angle counter clock-wise. The field of view is given by a 60° horizontal sweep, and 45° in the vertical direction. Thus, this convention allows us to define the following distances from pixel to pixel, in the horizontal (17) and vertical direction (18):

$$\Delta\alpha = \frac{60^\circ}{320} \tag{17}$$

$$\Delta\beta = \frac{45^\circ}{240} \tag{18}$$

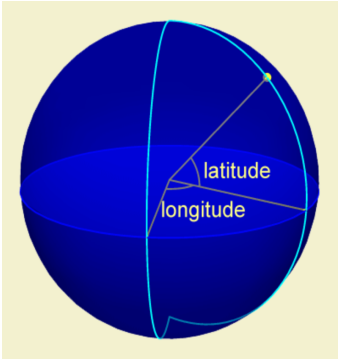

Fig. 2. Longitude and Latitude representation

Once we have defined what the spherical coordinates of each pixel are, we convert these coordinates into *Cartesian* coordinates. The transformation is given by the following set of equations:

$$\begin{aligned} x &= sin(\phi)cos(\lambda) \\ y &= sin(\phi)sin(\lambda) \\ z &= cos(\phi) \end{aligned} \tag{19}$$

Once we have a conversion to *Cartesian* coordinates, we aim to rotate these coordinates into the *world frame* coordinates. To do so we define the pose matrix T_t [7]. Which will have a rotation matrix R_t. The rotation matrices will define the transformation from a *non-inertial* frame of reference to an *inertial* frame, in our case it will be $\{W\}$. We note that the set of rotation matrices $R_{1:T}$ are computed by using the *ground truth* trajectory $\boldsymbol{q}_{1:T}$ obtained by the VICON data. The transformation from quaternions to rotation matrices is handled by the library function *scipy.spatial.transform* [8].

Next we use the closest-in-past timestamp to make sure that the rotations being applied at some time t closely match the time at which the images were captured. Once we do so, we perform the following operation in *homogeneous coordinates* (here we omit the 1 at the end):
$(x, y, z)^{pixel,world} = R_t * (x, y, z)^{pixel,body} + [0,\ 0,\ 0.1m]^T$
After this we perform a conversion back to spherical coordinates by using the following equations:

$$\lambda = arctan(\frac{x}{z}) \tag{20}$$

$$\phi = arctan(\frac{x}{x^2 + z^2}) \tag{21}$$

$$r \equiv 1$$

2) Projection: For the projection part, we have attempted the standard cylindrical method. In this method we note that the spherical projection, after the rotation has $-\pi$ to π range in the longitude direction and a $-\frac{\pi}{4}$ to $\frac{\pi}{4}$ in the latitude direction. We align the center of the partial sphere with the center of the panorama matrix. Next we determine the number of pixels covered by 1 radian in latitude and longitude direction. This is given by the "number of pixels in panorama width" over "longitude range in spherical coordinates".
Next, we multiply each cylinder coordinate by pixels per radian to get the (x, y) distance from the center of the panorama matrix (in this case it is the (1000, 1500) entry).

IV. Results

For the results section, we first present the results for *Orientation Tracking* and then we present *Panorama* results.

A. Orientation Tracking

Prior to the PGD step. We needed to ensure that the calibrated IMU data was infact calibrated correctly. To do so once we had the IMU data calibrated and ready, we used the *motion model* in (1) to see how well (1) was close to the VICON data. We also note that the sequence $\boldsymbol{q}_{1:T}$ that we obtain from (1) was then converted into *Roll*, *Pitch* and *Yaw* angles by using the library transforms3d in Python.
For *Data Set 1* the results are illustrated in Fig. 3.

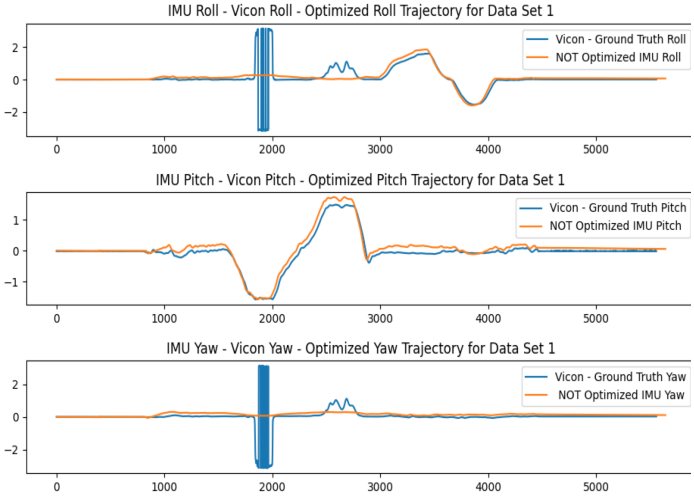

Fig. 3. Motion Model and Ground Truth data

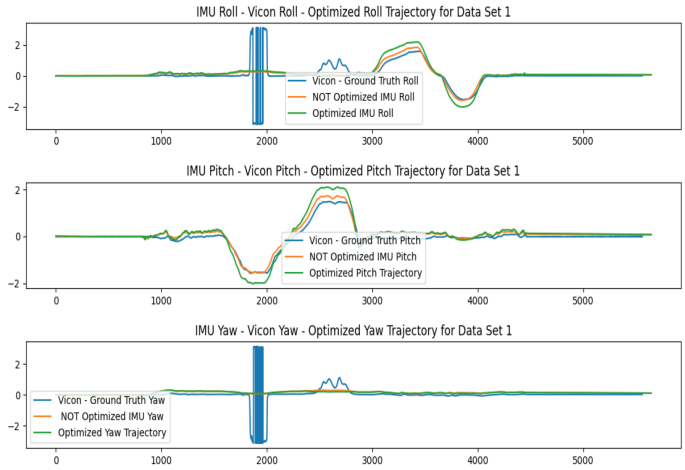

Fig. 4. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 1

We observe that when there is no "sudden" rotation, the un-optimzed *motion model* is able to track the *Ground truth* data fairly well. We can see this in the *Pitch* angle case.

Now, once we ensured that (1) "tracks" the Vicon Data relatively well. We begin to perform PGD to solve (5) subject to the constraint in (4). We initialize PGD by choosing $\boldsymbol{q}_0$ to be equal to the vector $[1, 0, 0, 0]^T$. Then to obtain the rest of the sequence $\boldsymbol{q}_{1:T}$ we simply use (1). Once the entire sequence is obtained we choose the total number of iterations for PGD to be $K = 7$ and then we perform (15) K times to get a new optimized sequence $\boldsymbol{q}_{1:T}^{new}$ which we plot against the *Ground truth* in *Data Sets* 1 to 9 (Fig. 4 - Fig. 13). In Fig. 14 and Fig. 15 we have the optimized sequence and the *Test Data* number 10 and 11 on the same plot. Additionally, we also have the a plot of how the values of the Cost function (5) decreases as the number of iterations increase for *Data Set 1* (Fig. 16).

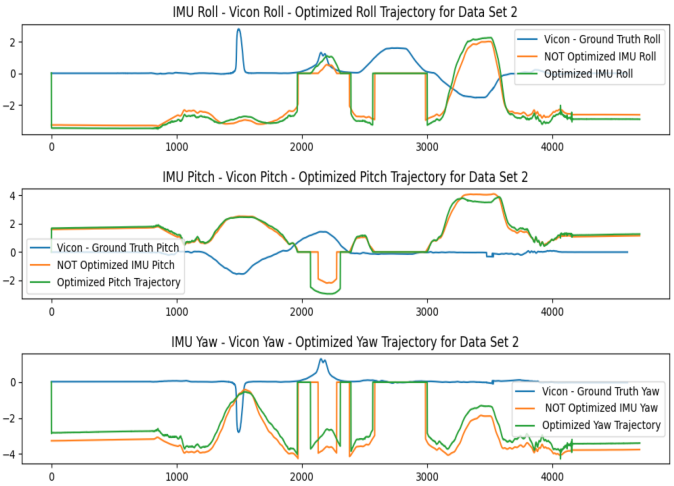

Fig. 5. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 2

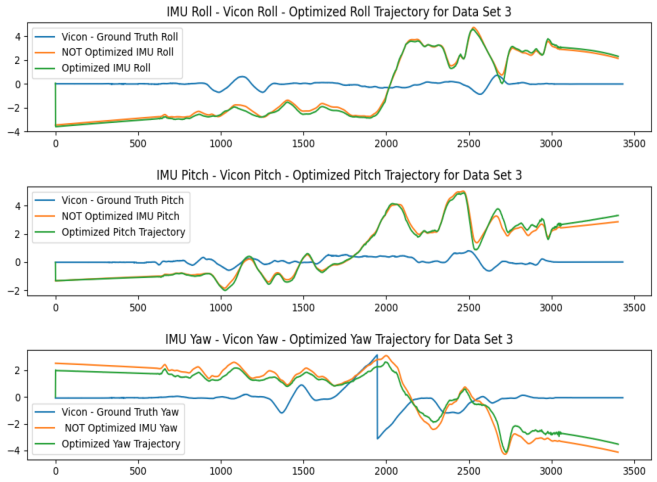

Fig. 6. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 3

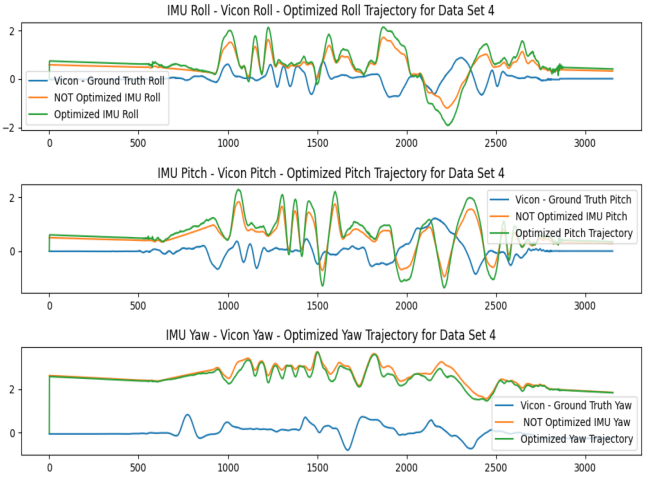

Fig. 7. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 4

Fig. 8. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 5

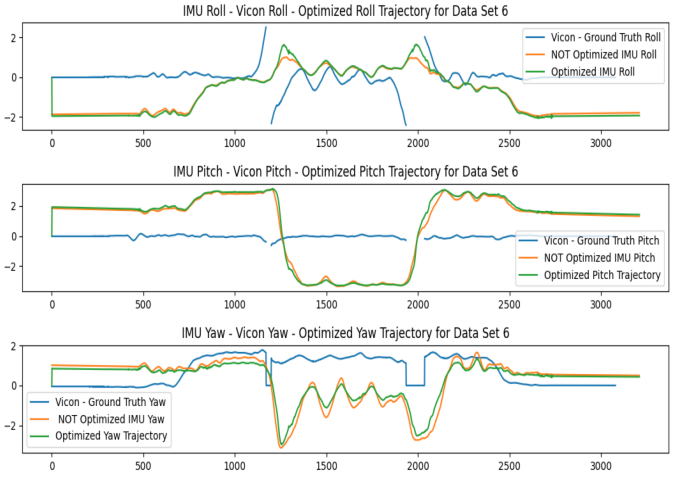

Fig. 9. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 6

Fig. 10. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 7

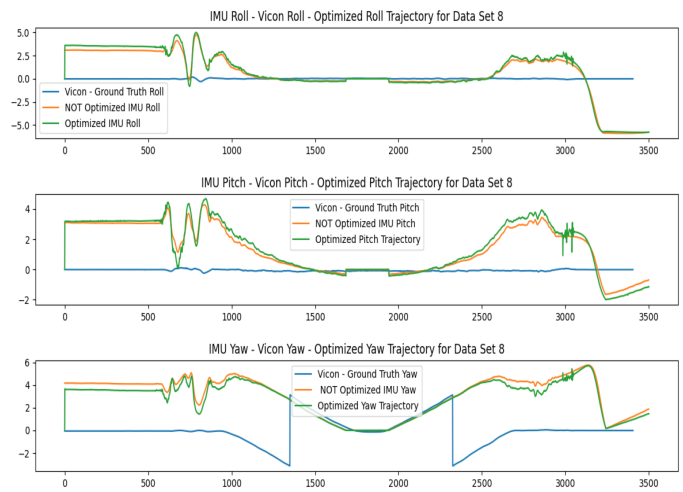

Fig. 11. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 8

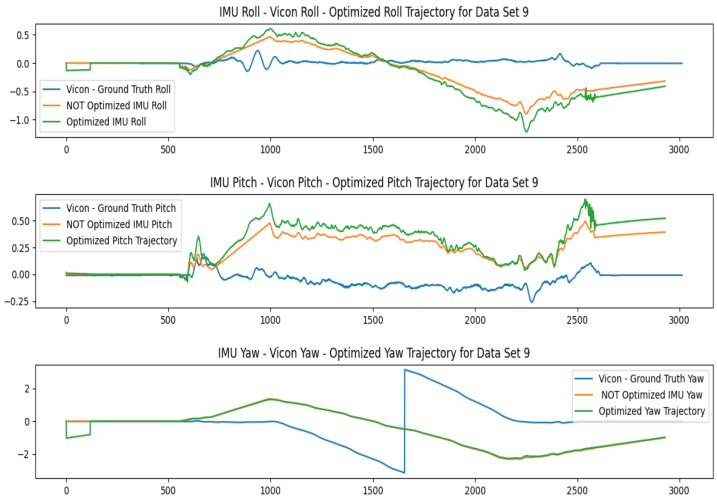

Fig. 12. Optimized Motion Model and Un-Optimized Motion Model and Ground Truth data for Data Set 9

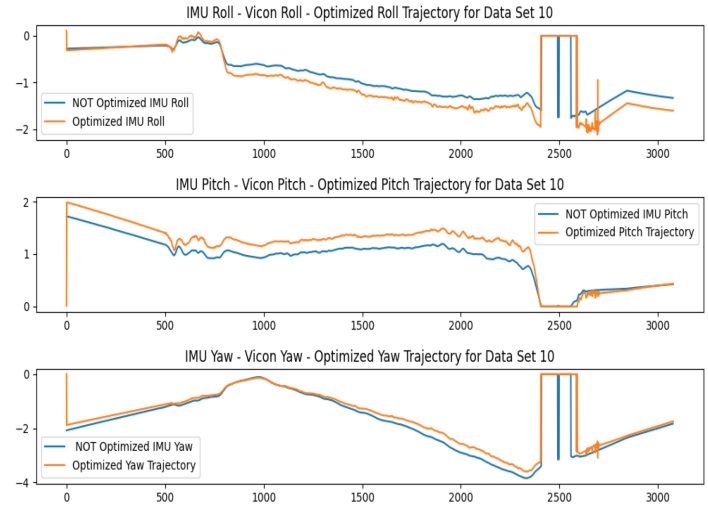

Fig. 13. Optimized Motion Model and Un-Optimized Motion Model for Test Data Set 10

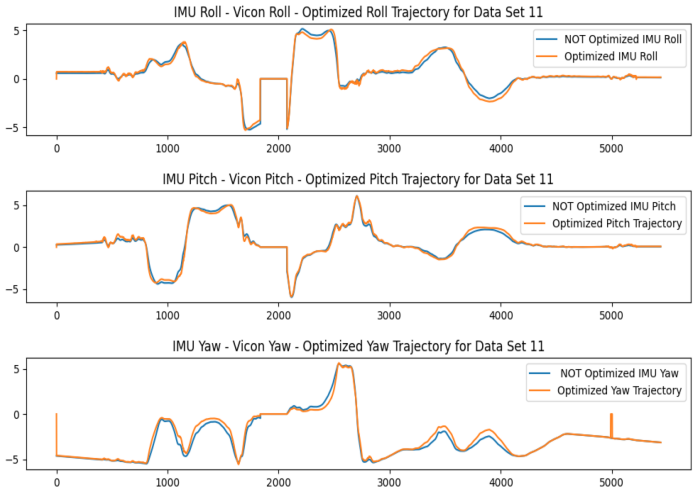

Fig. 14. Optimized Motion Model and Un-Optimized Motion Model for Test Data Set 11

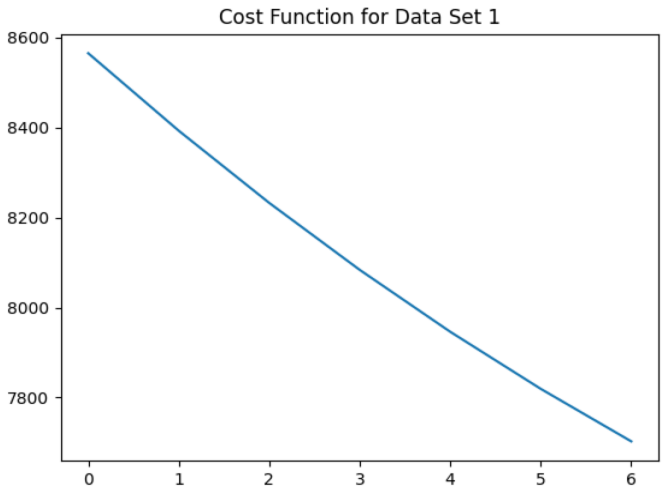

Fig. 15. Cost function vs. Iterations

As we can see, the optimized sequence at the end of K iterations seems to closely follow the *motion model* in (1) rather than the actual *Ground Truth* data. This could be attributed to the type of initialization. The PGD algorithm might be trapped in a local minima and descends by taking constant-size steps towards that minima, therefore minimizing the errors in (5) slowly. This in fact can also be seen by how the Cost function decreases as the number of iterations increase in Fig. 15. Here we have a quasi-linear decrease. An option could have been to maybe increase the number of iterations K, but we noticed that as $K > 8$ the minima was over shooted and the Cost function (5) began to increase. Another aspect which is worth mentioning is that the since (5) is not *parameterized* we are not learning a set of parameters that actually minimize (5) given the data. Therefore the minimization of (5) given the minimizer $\boldsymbol{q}_{1:T}^{new}$ does not generalize to all Data Sets 1 to 9. It is specific for the i^{th} Data Set i.

Since our optimized sequence closely resembles and/or follows (1), we notice good results in the *Test Data* sets, this is not a surprise.

B. Panorama

Now, we present the results for the *Panorama* portion. As we can see, the method used, even though valid in its own way did not yield good results. These are almost blank canvases with a strip running through the middle (Fig. 16 - Fig. 20). These poor results might have happened because of implementation issues. The method used doesn't seem to be wrong, as it was discussed in class several times. Therefore we attribute these results to the fact that there have been implementation "mistakes" when computing the spherical coordinates map.

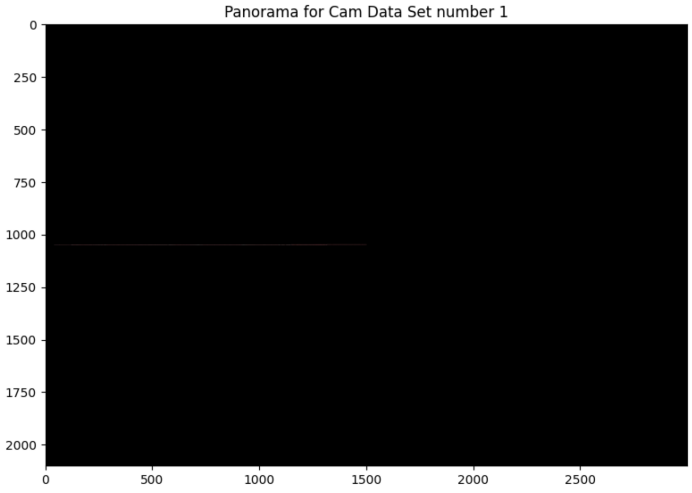

Fig. 16. Panorama for Data Set 1

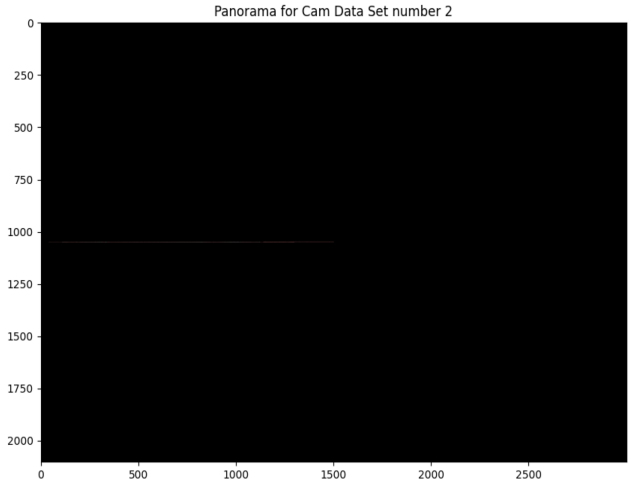

Fig. 17. Panorama for Data Set 2

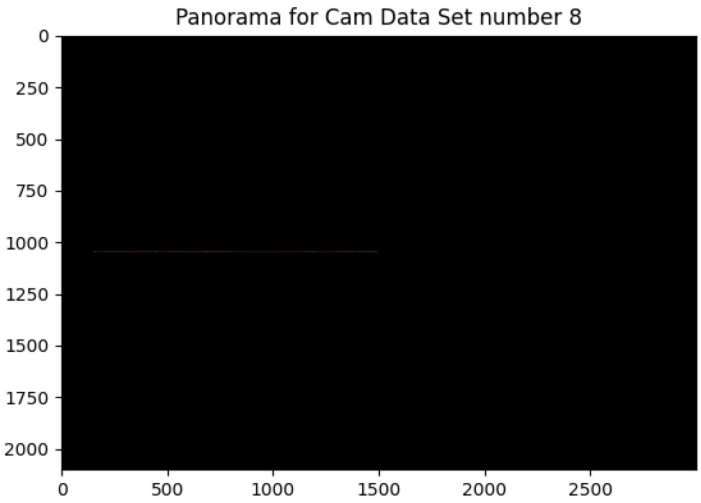

Fig. 18. Panorama for Data Set 8

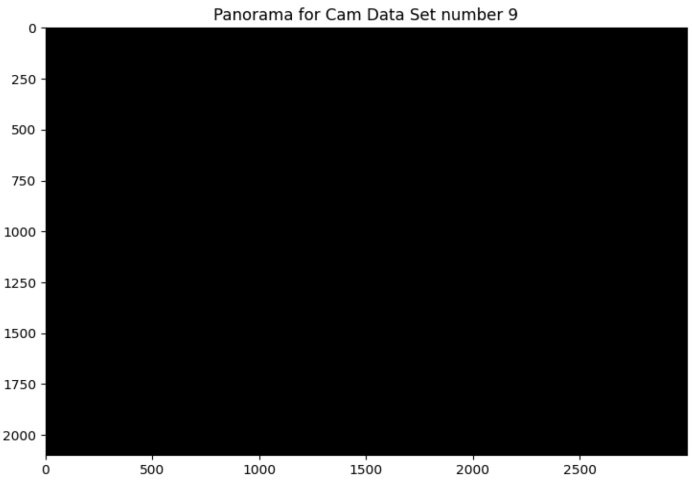

Fig. 19. Panorama for Data Set 9

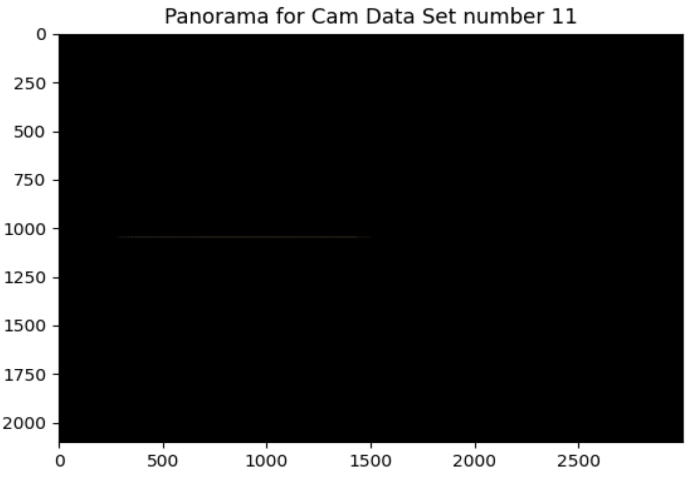

Fig. 20. Panorama for Data Set 11

By looking closely, in *Data Set* 8 for example, we notice that there is a very thin strip running from the middle left side of the panorama, all the way to its middle. In general, there have been some implementation mistakes, and possibly the cylindrical projection, even though standard it might have caused some issues since the initial creation of the *spherical coordinates* map might have been wrong. Therefore the initial mistake cascaded all the way to the final result.

REFERENCES

[1] ”https://en.wikipedia.org/wiki/Sphere”
[2] ”https://en.wikipedia.org/wiki/Lagrange_multiplier”
[3] slide14 ”https://natanaso.github.io/ece276a/ref/ECE276A_3_Rotations.pdf”
[4] slide25 ”https://natanaso.github.io/ece276a/ref/ECE276A_3_Rotations.pdf”
[5] ”https://en.wikipedia.org/wiki/Geographic_coordinate_system”
[6] slide30 ”https://natanaso.github.io/ece276a/ref/ECE276A_2_UnconstrainedOptimizatio
[7] slide30 ”https://natanaso.github.io/ece276a/ref/ECE276A_3_Rotations.pdf”
[8] ”https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.transform.Rotation.ht