Orientation Tracking

Chung-Pang, Wang
University of California San Diego
Department of Electrical and Computer Engineering

***Abstract*—In this project, we aim to leverage IMU data for tracking object orientation. Despite providing a rough estimate, IMU data may deviate from true orientation due to noise and bias. We introduce a gradient descent algorithm (GD) to accurately track the 3-D orientation of a rotating body using IMU measurements. Our approach ensures that consecutive orientations align with IMU angular velocity and gravity acceleration. Subsequently, we utilize the optimal orientation to construct panoramic images by stitching images over time.**

***Index Terms*—Optimization, Quaternions**

I. Introduction

In this project, we're trying to utilize the data obtained from IMU to track the orientation of the object. We can obtain a rough orientation from the IMU data. However, the orientation might be deviate from the true orientation due to the noise and bias of the IMU data. In this project, we introduce gradient descent algorithm (GD) to track the 3-D orientation of a rotating body using measurements from an inertial measurement unit. In short, we are using GD to estimate orientation based on the fact every two consecutive orientations should agree with angular velocity from IMU and the fact that purely rotating body should also agree with gravity acceleration. After obtaining optimal orientation, we utilize the orientation to construct a panoramic image by stitching the images over time.

II. Problem Formulation

A. Orientation Tracking

In the first part of the project, our goal is to estimate the orientation of the body over time using the IMU angular velocity ω_t and linear acceleration a_t measurements. We will use quaternion q_t to represent the body-frame orientation at time t since quaternion is an orientation representation that has no singularity(e.g. Gimbal Lock in Euler Angle Representation). Using the IMU angular velocity measurements ω_t and the differences between consecutive time stamps τ_t, we can predict the quaternion at the next step q_{t+1} using the quaternion kinematics. From the trajectory $R(t)$ of continuous rotation model, we have:

$$R^T R = I \Rightarrow \dot{R}^T R + R^T \dot{R} = 0 \tag{1}$$

Since $R^T \dot{R}$ is skew-symmetric:

$$R\dot{R}^T R + R R^T \dot{R} = 0 \Rightarrow -R\hat{\omega} + \dot{R} = 0 \tag{2}$$

We have continuous rotation kinematics:

$$\dot{R} = R\hat{\omega} \tag{3}$$

For this project, we have Discrete-time Motion Model:

$$q_{t+1} = f(q_t, \tau_t \omega_t) := q_t \circ exp([0, \tau_t \omega_t / 2]) \tag{4}$$

In this project, the body is undergoing pure rotation. Therefore,the acceleration of the body should be approximately $[0, 0, -g]$ in the world frame of reference, where g is $-9.81 m/s^2$ the gravity acceleration. Hence, the measured acceleration at in the IMU frame should agree with gravity acceleration after it is transformed to the IMU frame using the orientation q_t, we can then construct the observation model by transforming gravity vector from world fram into body frame (IMU frame):

$$a_t = h(q_t) := q_t^{-1} \circ [0, 0, 0, -g] \circ q_t \tag{5}$$

With the motion model (1) and observation model (2), we can now construct an objective function to formulate orientation tracking problem as a optimization problem for all orientation over time $q_{1:T} := q_1, q_2, ..., q_T$. The cost function of the orientation tracking optimization problem:

$$\begin{aligned} c\left(\mathbf{q}_{1:T}\right) := \frac{1}{2} \sum_{t=0}^{T-1} \left\| 2 \log\left(\mathbf{q}_{t+1}^{-1} \circ f(\mathbf{q}_t, \tau_t \boldsymbol{\omega}_t)\right) \right\|_2^2 \\ + \frac{1}{2} \sum_{t=1}^{T} \left\| \mathbf{a}_t - h\left(\mathbf{q}_t\right) \right\|_2^2 \end{aligned} \tag{6}$$

The first term measures the error between the estimated orientation from optimization updates and the motion model prediction, $q_{t+1}^{-1} \circ f\left(q_t, \tau_t \omega_t\right)$ indicates the difference in quaternion representation, if those two quaternion are the same, that operation will give us identity quaternion. Besides, the log function map quaternion to axis angle representation, then the error in axis angles can be compute by norm directly. The Second term measures the error between the world frame gravity vector transformed back to IMU frame and the readings from the accelerometer on the IMU. Since the quaternion can represent orientation correctly only when the quaternion is a unit quaternion. Therefore, we need to implement projected gradient decent algorithm to minimize the cost function.

B. Panorama

With the optimized quaternions, we can now construct a panoramic image by stitching the RGB images over time based on the body orientation $q_{1:T}$. We first need to convert the image coordinate to a spherical coordinate to match the orientation representation of a quaternion which is a vector at the center of a sphere (half sphere) pointing at the surface of the sphere.

Once we project the images on the sphere, we can inscribe the sphere in a cylinder to project images on it and finally unwrap the cylinder surface to a large rectangular image.

III. Methods

A. Projected Gradient Decent

With (6) and calibrated data, we can now implement PGD to optimized our estimated quaternions. We first have regular GD:

$$\widetilde{q}_{1:T} = q_{1:T} - \alpha \nabla_{q_{1:T}} c\left(\mathbf{q}_{1:T}\right) \tag{7}$$

where $q_{1:T}$ are the previous/initial quaternions, α is the step size and $\nabla_{q_{1:T}} c\left(\mathbf{q}_{1:T}\right)$ is the gradient of the cost function. In my approach, I projected the updated quaternion back to unit quaternion by simply normalizing $\widetilde{q}_{1:T}$, forcing the updated quaternion to be unit quaternions, ensures valid orientation representation of quaternions.

B. Panorama

For the construction of Panorama, we first need to transform the image coordinates (u, v) to spherical coordinates $(\lambda, \phi, 1)$ to project image onto the sphere. To find the longitude and latitude of the each pixel, we need to consider horizontal and vertical field of view (FOV), setting horizontal FOV $60°$ and vertical FOV $45°$. Therefore, the angle that a single pixel have in vertical and horizontal will be $\frac{45}{240}$, $\frac{60}{320}$ respectively (since the image have 320 pixels vertically and 240 pixels horizontally). Besides, since the origin of the image coordinate are at the top left. Thus, setting the center of be the origin will make originally origin $(-159, -119)$. With the spherical coordinates, we can convert it to Cartesian coordinates, enabling us to apply rotation to images. Then, we can project the coordinates on a cylinder and unwrap it to a rectangle image with width 2π and height π radians. Rotated Cartesian coordinate to Cylinder then unwrapped image coordinate can be written as:

$$u = (\arctan \frac{y_{cart}}{x_{cart}} - \pi)\frac{1919}{2\pi} \tag{8}$$

$$v = (\frac{2}{\pi} - z_{cart})\frac{959}{\pi} \tag{9}$$

Where (u, v) are the index of the large canvas where the image stitched at. The pixel intensity of the large canvas are initially zero and the size of the canvas is 1920x960 to match the FOV (e.g. $\frac{60}{320} = \frac{360}{1920}$). The first term of (10) represent arc length of the cylinder(setting radius = 1), then transform to image coordinate by subtracting half circumference of the cylinder. Finally, convert the degree per pixel of each pixel to pixel coordinate (index) of the large canvas and updating the intensity value at each pixel index to get a panoramic image.

C. Data

1) IMU calibration: Since the raw value from the IMU are A/D values, we need to convert A/D values to physical units.

$$value = (raw - bias) * scale\_factor$$

where

$$scale\_factor = Vref/1023/sensitivity$$

sensitivity for gyro is $3.33mV/deg/s$, and sensitivity for accelerometer is $300mV/g$ Therefore, value for gyro and accelerometer in physical units are:

$$\text{Value}_{\text{gyro}} = \frac{(raw - bias)3300\pi}{1023 \cdot 3.33 \cdot 180} \quad \text{(rad/s)} \tag{10}$$

$$\text{Value}_{\text{acc}} = \frac{(raw - bias)3300}{1023 \cdot 300} \quad \text{(g)} \tag{11}$$

Where bias are the average of the first 200 terms in each IMU data based on the fact that there is no rotation on the body in the few seconds of each data, meaning that gyro's value should be 0 and the accelerometer should be $[0, 0, 1] ingunits$ in the first few seconds. Besides, there are some glitch in dataset4. We can see from the top right in Fig.1, Gyro all 3 readings stays at a fixed value and then come back to normal. I fixed this by setting the first 600 values to be 0 and utilize (7) to obtain normal values. For the accelerometer glitch, there is a pulse in Az in the last few seconds. Thus, I use a clip function to clip the Az value between 0.5 and 1.5.

Fig. 1. Top left and Top right are the accelerometer and gyro glitch respectively, bottom are the calibrated data

D. Implementation Details

1) Orientation Tracking: For the implementation of orientation tracking, since most of the quaternions operations library in Python are not differentiable, I implement most of the quaternion operation such as quaternion addition, exponential map, logarithm map, etc. Furthermore, I use Jax library [4] to implement gradient decent for the optimization part of orientation tracking, hence the quaternion operation are also in Jax format. For the initialization of (6), I use motion model

to initial and found the results of the optimized quternions are outperform than initialization with unit quaternions.

2) Panorama: For the implementation of Panorama, I first utilize OpenCV to write all of the images in a video to get a good understanding of how the orientation should look like in 3D space. For the coordinates transformation, I use Astropy library [3] to transform spherical coordinate to Cartesian coordinate and apply rotation to coordinate (xR^T). Finally swap 3 channels intensity values of the large canvas at transformed image indexes for all timestamps to get a panoramic image.

IV. Results

A. Orientation Tracking

The iterations of optimization loop are simply determined by observing the lowest loss from the optimal quaternion. Figure 2 shows how cost function converge for all dataset. We can tell that since we are using fixed step size, the converge rate for all dataset are quiet similar. An interesting observation is that when I tried to choose unit quaternion as initial quaternion, the lowest losses are much larger than using motion model to initialized. I believed the reason is that unit quternions are too far from the actual orientation, hence the optimization algorithm will easily stuck at local minimum, while the initial motion model are already very closed to the true orientation. Figure 3 shows the optimal orientation compare with Vicon ground truth orientation. Besides, I believed that quaternion is a better way to represent orientation than Euler angles since there is no singularity in quaternion.

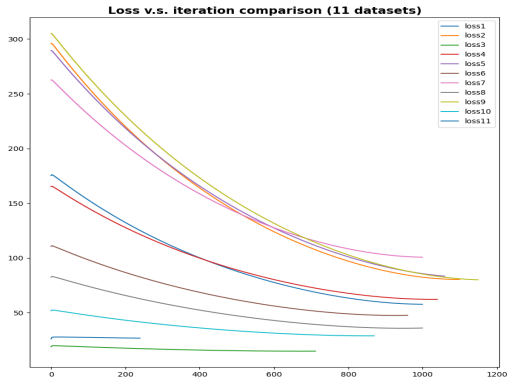

Fig. 2. Left: Euler angles representation of orientation ; Right: quaternion representation of orientation (Testing Dataset).

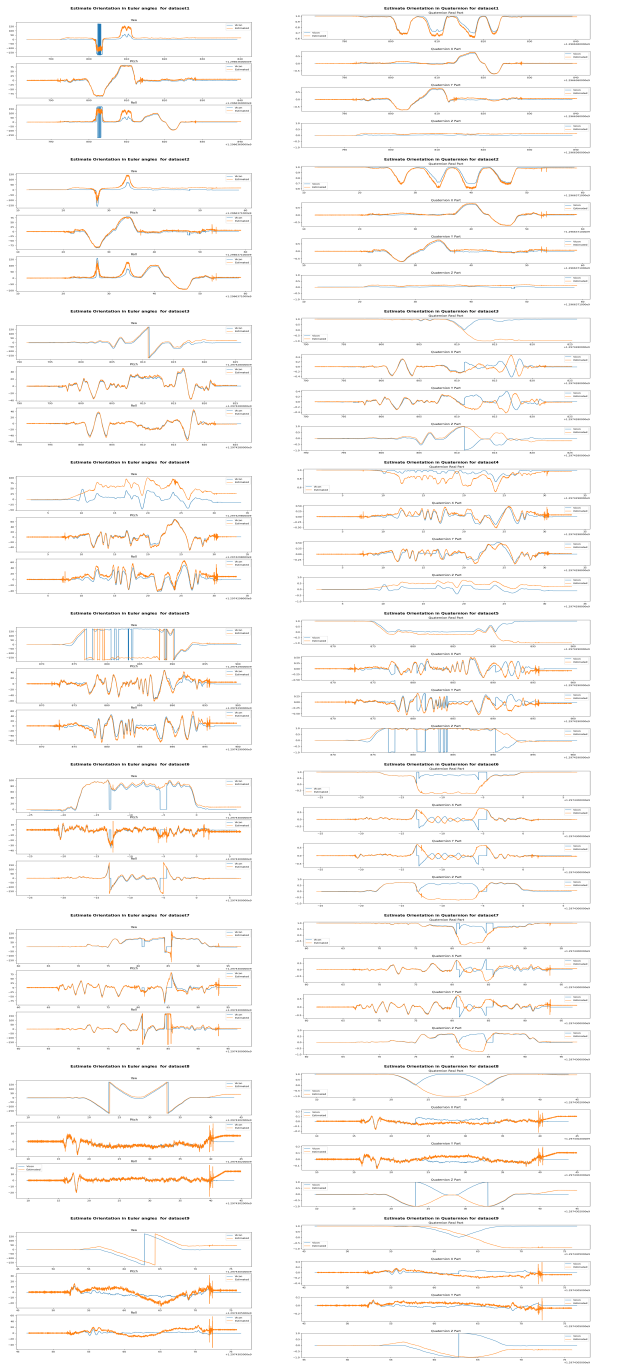

Fig. 3. Left: Euler angles representation of orientation ; Right: quaternion representation of orientation (Training Dataset).

B. Panorama

In Figure 5, we can see that there are no big difference between the Panorama using Vicon ground truth data and the Panorama estimated quaternion. We can also imagine how the camera rotate in the space from the Panorama. The color of the Panoramas are not the same as the Panorama in the project document since I convert the image from BGR to RGB which is how the image should look like in real life.

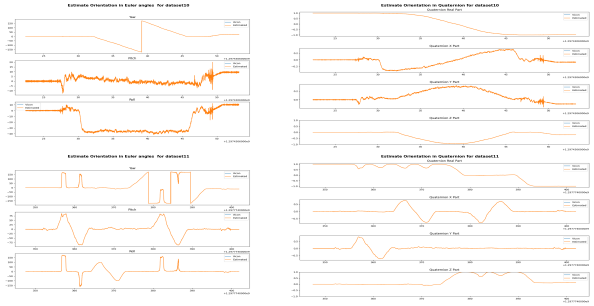

Fig. 4. Left: Euler angles representation of orientation ; Right: quaternion representation of orientation (Testing Dataset).

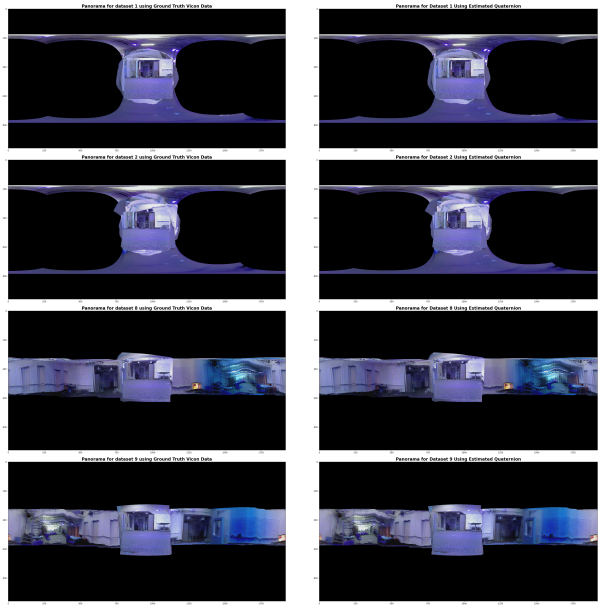

Fig. 5. Left: Panorama using Vicon Data ; Right: Panorama using optimized quaternion (Testing Dataset).

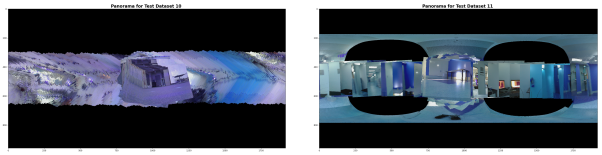

Fig. 6. Left: Panorama using optimized quaternion (dataset10) ; Right: Panorama using optimized quaternion (dataset11)

REFERENCES

[1] UCSD ECE276A slides on rotations https://natanaso.github.io/ece276a/ref/ECE276A_3_Rotations.pdf

[2] UCSD ECE276A slides on Robot Motion and Observation Modelshttps://natanaso.github.io/ece276a/ref/ECE276A_4_MotionAndObservationModels.pdf

[3] astropy library https://docs.astropy.org/en/stable/api/astropy.coordinates.spherical_to_cartesian.html

[4] Jax library https://jax.readthedocs.io/en/latest/