Orientation Tracking

Renu Krishna Gutta, PID: A59018210, ECE 276A, Project-1

1. INTRODUCTION

The goal of this project is to track the 3D-orientation of a rigid body undergoing pure rotation. The Inertial Measurement Unit (IMU) is a sensor attached to the body which comprises an accelerometer and a gyroscope. It measures the linear acceleration and angular velocity in 3D space with respect to IMU (body) frame. Ideally, we can employ kinematics equations over these values and compute the orientation of the body.

However, the accelerometer and gyroscope add intrinsic noises to the readings and there can be situations where their noise behaviour changes due to external factors. For example, a rover sent onto Mars. We cannot replace or recalibrate its sensors at out will. In such a scenario, we are left to deal with readings that are available and perform optimizations to reduce the noise.

In this project, I implemented a projected gradient descent algorithm to track the 3D-orientation of a purely rotating body using the readings from the IMU. Later, I generated panoramic images by stitching the images captured by camera attached to the body.

2. PROBLEM FORMULATION

Following observations are available to us:

a. Inertial Measurement Unit (IMU) – recorded at regular intervals. (w.r.t body/IMU frame)
   i. Linear acceleration: A_x, A_y, A_z
   ii. Angular velocity: W_x, W_y, W_z
b. Images – The camera attached to the body captures images at regular intervals.

2.1. Symbols definitions

Symbol	Description
ω_t	IMU angular velocity
a_t	IMU linear acceleration
q_t	Unit norm quaternion – to represent the body-frame orientation at time *t*
τ_t	Difference between consecutive time stamps
g	Magnitude of acceleration due to gravity (9.8 m/s^2)

2.2. Quaternion Kinematics Model

This is a Kinematics based equation relating orientation with angular velocity and time elapsed. It calculates the angle turned during a time period with a particular angular velocity.

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

$\circ$: denotes quaternion multiplication
exp(.): denotes quaternion exponential function

2.3. Observation Model

Since the body is stationary and no external translational forces are applied, there is the force of gravity acting in downward (-z) direction. That is, body's acceleration is *g* in the -z direction with respect to the world frame. The Observation Model outputs the linear acceleration in the body frame, by applying quaternion rotation to the linear acceleration in the world frame.

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

2.4. Cost Function

We formulate the following two objectives so as to estimate trajectory based on noisy IMU readings:

i. Minimize the relative rotation between q_{t+1} and $f(q_t, \tau_t \omega_t)$ because we want the real orientation to be similar to the one calculated by the Quaternion Kinematics Model.
ii. Minimize the squared error between the observed linear acceleration in the body frame (IMU) and the one calculated using the Observation Model.

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

$q_{1:T}$: $[q_1, q_2, \dots \dots, q_T]^t$
log(.): quaternion logarithmic function

We have to minimize the cost function by maintaining the unit norm condition of the quaternion.

$$\min_{q_{1:T}} c(q_{1:T})$$
$$such\ that\ \|q_t\|_2 = 1\ for\ all\ t = 1, 2, \dots, T$$

3. TECHNICAL APPROACH

3.1. Data Processing

- The training data included IMU readings, camera images and VICON ground truth orientation matrices at multiple time stamps in sequence.
- **IMU bias estimation:** By plotting the yaw, pitch and roll from the VICON data, I found the initial rest period and considered the average of IMU readings during this period as its bias. Since the setup is same

for all the training sets, I calculated the bias based on first training set and applied it to all the sets.

- **Conversion formula and sensor parameters:**
  a. $Value_{imu} = (raw_{imu} - bias) \times sacle\ factor$
  b. $scale\ factor = \frac{Vref}{1023 \times sensitivity}$
  c. raw_{imu}: This is a 10-bit ADC value
  d. $Vref$ = 3300 mV
  e. $sensitivity$:
     i. Gyroscope (all 3 axes): $3.33 \times 180/\pi$ mV/radian/s
     ii. Accelerometer (all 3 axes): Typical value, 300 mV/g
- The acceleration values are converted into g units. Initially, when the body is at rest, body frame and world frame are the same. So, the A_z value is ensured to be -1g, so that it is in agreement with the Observation Model (sec. 2.3) when I pass the initial quaternion [1, 0, 0, 0] into the model.

3.2. Orientation Tracking (Quaternion Estimation)

- **Parameter to be estimated:** $q_{1:T}$: $[q_1, q_2, \dots\dots, q_T]^t$, called *orientation trajectory*. Here T is the number of data samples collected as time progressed.
- **Initialize:** $q_0 = [1, 0, 0, 0]$, $q_{1:T}$ = 4xT matrix is initialized to the values computed using the Quaternion Kinematics Model (sec. 2.2)
- $\alpha^{(0)} = 1$, the step size for gradient based parameter update.
- **Compute gradient:** Using *autograd.grad* in Python, I computed the gradient of cost function (sec. 2.4) with respect to $q_{1:T}$
- **Gradient:** $grad^{(k)}$ is a 4xT matrix, where T is the number of time samples.
- **Parameter update:** Let, *k* is the iteration index;

$$q_{1:T}{}^{(k+1)} = q_{1:T}{}^{(k+1)} - \alpha^{(k)} \times grad^{(k)}$$

- **Unit norm constraint:** Each column of 4xT matrix, $q_{1:T}{}^{(k+1)}$ is a quaternion and is normalized to unit magnitude by dividng with its respective magnitude, i.e.,

$$q_i{}^{(k+1)} \leftarrow \frac{q_i{}^{(k+1)}}{\|q_i{}^{(k+1)}\|_2}, for\ all\ i = 1, 2, \dots, T$$

- **Step size update:** At every iteration, **if cost increases**, discard the new parameter and re-run the iteration with a decreased step size. **If the cost decreases** and magnitude (frobenius norm) of the gradient is sufficiently large, then increase the step size.

IF $c(q_i{}^{(k+1)}) \leq c(q_{1:T}{}^{(k+1)})$, **THEN**
  IF $\|grad^{(k)}\|_F > 0.1$: $\alpha^{(k+1)} \leftarrow \alpha^{(k)} \times 2$ *// increase the step size only if the magnitude of gradient > 0.1*
ELSE
  $\alpha^{(k+1)} \leftarrow \alpha^{(k)} / 10$ *// decrease the step size to take smaller steps*
  $q_i{}^{(k+1)} \leftarrow q_i{}^{(k)}$ *// discard the new parameters and run next iteration with current parameters*

- **Stopping criteria:** If the reduction in the cost value is less than 0.01 for 10 consecutive iterations, then stop the optimization.

3.3. Panorama

Following is the visualization of projecting camera image onto a unit sphere

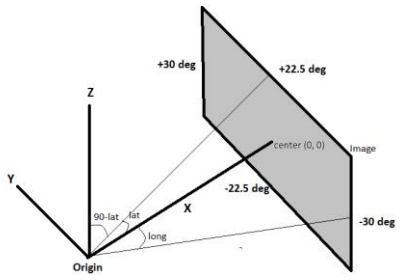

Figure – 3.1: *Assigning latitude and longitude to image pixels*

- The camera image size is 240x320, i.e., 240 rows and 320 columns.
- Horizontal (longitudinal) field of view is 60° and vertical (latitudinal) field of view is 45°
- Every pixel is assigned (latitude, longitude) coordinates.
- I took radial distance (*r*) as 1. Hence, I did not explicitly mention *r* in the equations.
- Based on Fig.3.1, latitude is the angle between the XY plane and the line joining origin and the location on the image.
- To be in accordance with the assumed X-Y-Z axes, I chose upper half to be positive latitude and left half to be positive longitude.
- **Spherical to Cartesian conversion:** Based on Fig.3.1, for a given (*lat*, *long*), the (*x*, *y*, *z*) coordinates are;

$$z = \cos(90° - lat) = \sin(lat)$$
$$x = \cos(lat) . \cos(long)$$
$$y = \cos(lat) . \sin(long)$$

- **Cartesian to Spherical conversion:** Based on Fig.3.1, for a given (x, y, z), the (lat, long) coordinates are;

$$long = sign(y) \arccos\left(\frac{x}{\sqrt{x^2 + y^2}}\right)$$
$$lat = 90° - arccos\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right)$$

- **Spherical to Cartesian (Body-Frame):** Using the above equations, I converted the spherical coordinates of each pixel into cartesian coordinates.
- **Body to World Frame:** I used the estimated orientation at the timestamp that is nearest to that of the camera image, to transform the body frame coordinates into the world frame.

- **Cartesian to Spherical (World-Frame):** Performed using the above equations.
- **Cylindrical projection and unwrapping:** Depicted in Fig.3.2 below:

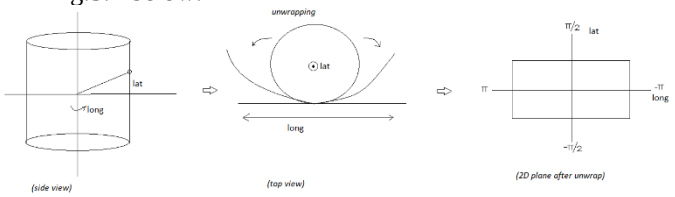

Figure – 3.2: *Cylinder unwrapping*

- The unwrapped 2D plane in Fig.3.2 is scaled to a 720x1080 panorama image.
- The (lat_w, $long_w$) coordinates in the world-frame are the measure of vertical and horizontal distances from the centre of the panorama image.
- **Pixel locations on the panorama image:**
  - **i.** Size of panorama image $(N_v, N_h) = (720, 1080)$ – number of pixels in vertical and horizontal directions respectively
  - **ii.** Resolution $(res_v, res_h) = \left(\frac{N_v}{\pi}, \frac{N_h}{2\pi}\right)$ pixel/radian
  - iii. Assuming top left corner of the grid to be (0, 0)
  - iv. The centre will be at $(C_v, C_h) = (359, 539)$
  - v. For any point with coordinates $(lat_w, long_w)$:
    - a. Vertical displacement from centre
      $$d_v = lat_w \times res_v$$
    - b. Horizontal displacement from centre
      $$d_h = long_w \times res_h$$
  - **vi.** **The final pixel coordinates are:**
    $$(C_v - d_v, C_h - d_h)$$
    Values are rounded off to nearest integers
- Once I obtained the final pixel coordinates, I assigned the corresponding RGB values from original image to that location on the panorama image.
- The above discussed operations were performed to all pixels simultaneously by using vectorization techniques in Python.

4. RESULTS

4.1. Plots

Yaw-Pitch-Roll plots
X-axis: Time
Y-axis: Angle in radian

Figure: 4.1: For the training data, the estimated quaternions are converted into Euler angles as Yaw, Pitch and Roll (in radians) and plotted against the ground truth values provided in Vicon.

Figure: 4.2: Estimated Yaw, Pitch and Roll plots for the testing data.

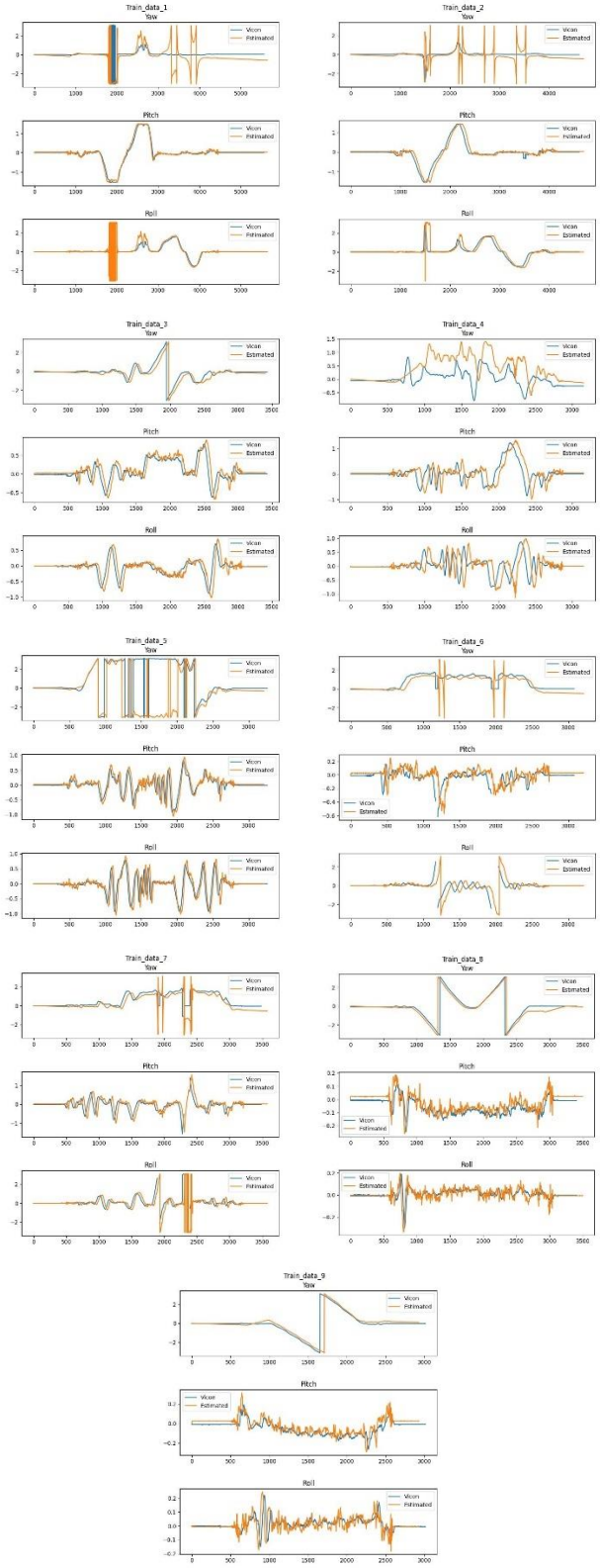

Figure – 4.1(i-ix): *Training data plots. True (blue) vs Estimated (orange)*

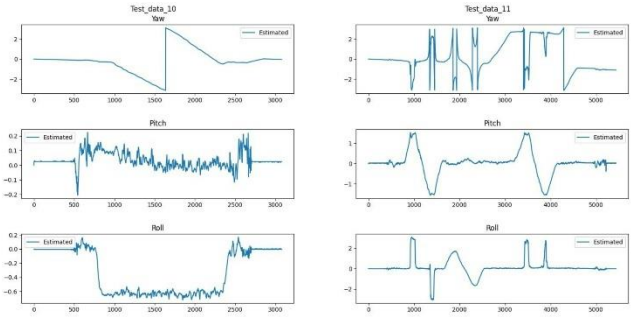

Figure – 4.2: *Testing data plots*

4.2. Panorama images

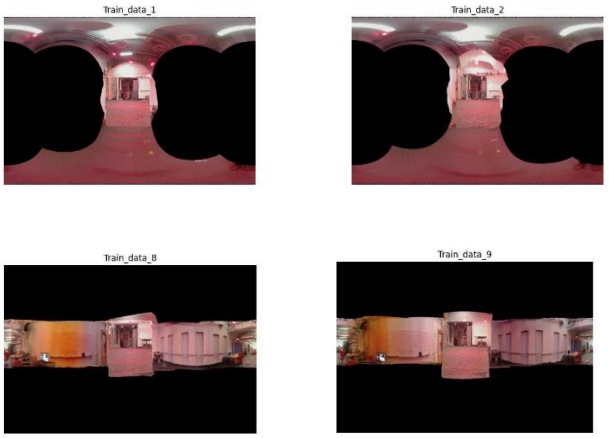

Figure – 4.3: *Panorama of training set based on VICON true orientation trajectory*

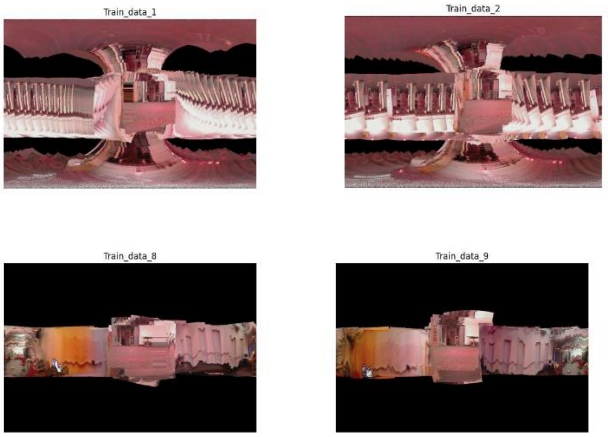

Figure – 4.4: *Panorama of training set based on estimated orientation trajectory*

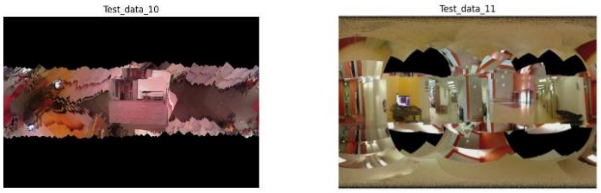

Figure – 4.5: *Panorama of testing set based on estimated orientation trajectory*

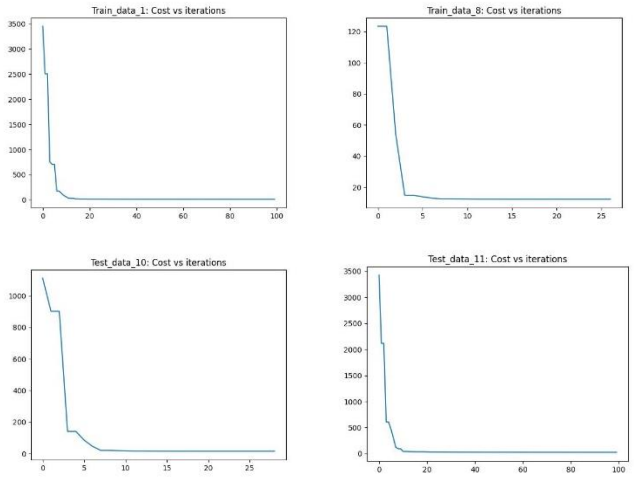

Figure – 4.6: *Cost variation with iterations for train data (#1, #8) and test data (#10, #11)*

4.3. Discussion

- I set the maximum number of gradient descent iterations to be 200 due to time limitations as one iteration of batch gradient descent took about 20sec for the dataset with highest number of samples.
- Except training set #7, all the sets hit the exit condition within 200 iterations. The usage of adaptive step size might have helped in faster convergence.
- Number of iterations:

Data set	*Number of iterations (max 200)*
Train set 1	130
Train set 2	154
Train set 3	28
Train set 4	26
Train set 5	26
Train set 6	144
Train set 7	200
Train set 8	26
Train set 9	26
Testset 10	28
Testset 11	198

- The estimated orientation trajectories were mostly matching with the ground truth ones.
- However, for the training sets 1, 2 and 4, there was considerable difference in the estimated Yaw angle compared to the ground truth.

- This was evident in the panorama images of set# 1&2 in fig.4.3 and fig.4.4 where the estimated case has image artefacts in the horizontal direction due to these differences in the Yaw angle.
- In the case of training set #4, though the algorithm exited in few iterations, there is still high error in the Yaw angle.
- Accuracy can be further increased by running more number iterations and fine tuning the exit condition.
- Also, vectorization can be employed to speed up the cost and gradient computation.
- Fig.4.6 shows the cost value variation with the iterations.
- We can see that further study is needed to improve the optimization.
- For the panorama construction, all the geometric operations were applied in a vectorized form which increased the computational speed by a large amount.

ACKNOWLEDGEMENT

I would like to thank Prof. Nikolay Atanasov, TAs – Sambaran Ghosal and Shrey Kansal for guiding me with this project.

REFERENCES

[1] https: //natanaso.github.io/ece276a
[2] https://matthew-brett.github.io/transforms3d/
[3] https://math.stackexchange.com/questions/54855/gradient-descent-with-constraints
[4] https://en.wikipedia.org/wiki/Spherical_coordinate_system