Orientation Tracking and Panorama Reconstruction Using Projected Gradient Descent

Anthony Nguyen
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA, USA
ann048@ucsd.edu

***Abstract*—This report considers an orientation tracking problem, where a robot undergoing pure rotation is equipped with an Inertial Measurement Unit (IMU) and a camera. We formulate an optimization model based on motion and observation models corresponding to these sensors, and we showcase projected gradient descent methods to solve this problem. We also show how our estimated trajectories can then be used to create panorama images based on our camera image data. Finally, we discuss effective implementation techniques for the aforementioned methods.**

***Index Terms*—Orientation tracking, gradient descent, projected gradient descent, constrained optimization, quaternions, panorama reconstruction**

I. Introduction

In robotics, one of the fundamental problems is the estimation of the robot's current state over time. This is often done using information from the robot's sensor measurements, its control input history, and, in some cases, maps of the environment. Solving such state estimation problems is key for robots to interact effectively with their respective environments, both in successfully accomplishing their objectives and protecting themselves from damage.

This report focuses on orientation tracking, a special case of the robot state estimation problem where we estimate the orientation over time of a robot that is undergoing pure rotation. In our specific problem, we estimate the orientations, which we represent as quaternions, using projected gradient descent on a cost function derived from motion and observation models based on angular velocity and linear acceleration measurements, respectively, from an Inertial Measurement Unit (IMU) on our robot. We then use our estimated orientation trajectory to stitch images obtained from the robot's camera into a panorama. In summary, this report discusses the following:

- Calculation of IMU biases using known robot states.
- Motion-model-based initialization conditions and well-performing hyperparameters for implementing projected gradient descent.
- Time-stamp matching and coordinate-frame transformations for mapping camera image pixels to panorama image pixels.
- Strategies for effective and efficient implementation.

II. Problem Formulation

Consider a robot (rigid body) that is undergoing pure rotation. We can represent the orientation of the robot's coordinate frame $\{B\}$ relative to a fixed coordinate axis (dubbed the "world-frame") $\{W\}$ as a unit-norm quaternion ${}_W\mathbf{q}_B$, which can simply be written as

$$\mathbf{q} = q_s + q_1 i + q_2 j + q_3 k = [q_s, \mathbf{q}_v] \in \mathbb{H}_*,$$

where $\mathbb{H}_* \coloneqq \{\mathbf{q} \in \mathbb{H} \mid q_s^2 + \mathbf{q}_v^\top \mathbf{q}_v = 1\}$, $\mathbb{H} \coloneqq \mathbb{C} \oplus \mathbb{C}j$, and $\mathbf{q}_v \coloneqq \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^\top \in \mathbb{R}^3$. When considering time, the orientation of $\{B\}$ in $\{W\}$ at the tth time stamp can be depicted as $\mathbf{q}_t = [q_{s,t}, \mathbf{q}_{v,t}]$, where $\mathbf{q}_t$ is shorthand for ${}_W\mathbf{q}_{B,t}$.

In our specific problem, we want to estimate the orientation trajectory $\mathbf{q}_{1:T} \coloneqq \mathbf{q}_1, ..., \mathbf{q}_T \in \mathbb{H}_*$ using angular velocity measurements $\boldsymbol{\omega}_t \in \mathbb{R}^3$ (rad/s) and linear acceleration measurements $\mathbf{a}_t \in \mathbb{R}^3$ (m/s^2) from an Inertial Measurement Unit (IMU) on our robot. To simplify calculations, the IMU frame will be $\{B\}$.

If we define τ_t (sec) to be the difference between the tth and $(t+1)$th time stamps, we can find the axis-angle rotation vector $\boldsymbol{\theta}_t = \tau_t \boldsymbol{\omega}_t$, which describes $\{B\}$'s change of orientation in $\{W\}$ from time stamp t to $(t+1)$. We can then convert $\boldsymbol{\theta}_t$ into a quaternion in $\mathbb{H}_*$ using (15) as $\exp([0, \boldsymbol{\theta}_t/2]) = \exp([0, \tau_t\boldsymbol{\omega}_t/2]) = {}_{B,t}\mathbf{q}_{B,t+1}$. Subsequently, we can predict the orientation in $\{W\}$ at the next time stamp, $\mathbf{q}_{t+1} = {}_W\mathbf{q}_{B,t+1}$, using the following motion model:

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

where the $\circ$ operation represents quaternion multiplication (12) and $\exp(\cdot)$ represents the quaternion exponential function (16).

The motion model is based off of the idea that the multiplication of two quaternions representing orientations in successive reference frames (${}_W\mathbf{q}_{B,t+1}$ and ${}_{B,t}\mathbf{q}_{B,t+1}$, in this case) results in a quaternion ${}_W\mathbf{q}_{B,t+1}$ that corresponds to a rotation between the non-adjacent frames (see Appendix C and (20)). It is important to note, however, that $\{B\}$ is still a single reference frame that simply has different orientations at different times, and we simply treat it as multiple reference frames for the purpose of this explanation.

To make use of information from our measured IMU frame linear acceleration, $\mathbf{a}_t$, as our robot is only undergoing rotation, the 3D linear acceleration vector in $\{W\}$ is a constant $\begin{bmatrix} 0 & 0 & -g \end{bmatrix}^\top$ m/s^2, which is $\mathbf{a}_W = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$ in gravity units. As a result, the inverse of our orientation, $\mathbf{q}_t^{-1} = {}_{B,t}\mathbf{q}_W$, should correspond to a rotation from $\{W\}$ to $\{B\}$ at time

stamp t that maps $\mathbf{a}_W$ into $\mathbf{a}_t$. This is captured in the following observation model:

$$[0, \mathbf{a}_t] = h(\mathbf{q}_t) \coloneqq \mathbf{q}_t^{-1} \circ [0, \mathbf{a}_W] \circ \mathbf{q}_t, \tag{2}$$

which is simply quaternion rotation (11) with $\mathbf{q}_t^{-1}$.

Considering both (1) and (2) lets us define the following cost function, which equally weights two error totals:

$$\begin{aligned} c(\mathbf{q}_{1:T}) = \frac{1}{2} \sum_{t=0}^{T-1} ||2 \log(\mathbf{q}_{t+1}^{-1} \circ f(\mathbf{q}_t, \tau_t \boldsymbol{\omega}_t))||_2^2 \\ + \frac{1}{2} \sum_{t=1}^{T} ||[0, \mathbf{a}_t] - h(\mathbf{q}_t)||_2^2. \end{aligned} \tag{3}$$

The first term is the sum of errors between our estimated $\mathbf{q}_{t+1}$ and its corresponding motion model prediction, $f(\mathbf{q}_t, \tau_t \boldsymbol{\omega}_t)$ for all time steps. This error is found specifically by recovering an axis-angle rotation vector (using (17)) from the relative rotation between the two aforementioned terms and finding its squared norm. If we let $[0, \boldsymbol{\theta}_{t+1}] = [0, \theta_{t+1} \boldsymbol{\eta}_{t+1}] = 2 \log(\mathbf{q}_{t+1}^{-1} \circ f(\mathbf{q}_t, \tau_t \boldsymbol{\omega}_t))$, then we can find that this error is simply the squared angle of the axis-angle rotation vector: as $\boldsymbol{\eta}_{t+1}$ is a unit vector, $||[0, \theta_{t+1} \boldsymbol{\eta}_{t+1}]||_2^2 = \theta_{t+1}^2$.

Similarly, the second term is the sum of errors between the measured linear acceleration and our corresponding observation model prediction of it, found as the squared norm of the difference between $[0, \mathbf{a}_t]$ and $h(\mathbf{q}_t)$.

Finally, we optimize (3) as follows,

$$\begin{aligned} &\min_{\mathbf{q}_{1:T}} c(\mathbf{q}_{1:T}) \\ &\text{s.t. } ||\mathbf{q}_t||_2 = 1, \quad t = 1, ..., T, \end{aligned} \tag{4}$$

noting that the constraint $||\mathbf{q}_t||_2 = 1$ enforces that all our quaternions $\mathbf{q}_{1:T} \in \mathbb{H}_*$. Afterwards, we can use our estimated orientation trajectory $\mathbf{q}_{1:T}$ to stitch together K 320×240 RGB camera images as a panorama, where the camera regular frame $\{R\}$ has origin at ${}_{\{B\}}\mathbf{p}_{\{R\}} = \begin{bmatrix} 0 & 0 & 0.1 \end{bmatrix}^\top$ relative to $\{B\}$.

III. Technical Approach

This session will discuss a technical approach to the problem established in Section II. First, it simplifies calculations to choose the world frame $\{W\}$ such that it always has the same origin as $\{B\}$ (i.e. the relative position of the origin of $\{B\}$ in $\{W\}$ is ${}_{\{W\}}\mathbf{p}_{\{B\}} = 0$), and this fact will be used throughout our approach.

A. Orientation Tracking

1) Calculating the IMU Biases: When extracting useful measurements from an IMU, one must first calculate the IMU's gyroscope and accelerometer biases, which are constant with time in this problem. As discussed in Section II, the linear acceleration of the robot in $\{W\}$ is a constant $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$ gravity units. As we can see from the plots in Appendix D, the IMU frame $\{B\}$ is aligned with $\{W\}$ and undergoes no rotation for a few seconds at the start of each training dataset, meaning that we can expect the IMU to measure linear acceleration $\mathbf{a}_t = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$ and angular velocity $\boldsymbol{\omega}_t = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$ during this time.

The raw values, $\mathbf{a}_t^{(raw)}$ and $\boldsymbol{\omega}_t^{(raw)}$, that the IMU returns from its Analog/Digital Converter (ADC) can be converted to their true physical units values, $\mathbf{a}_t$ and $\boldsymbol{\omega}_t$, via the following:

$$\begin{aligned} \mathbf{a}_t &= (\mathbf{a}_t^{(raw)} - \mathbf{b}_a) \frac{V_{ref}}{(2^n - 1) s_a}, \\ \boldsymbol{\omega}_t &= (\boldsymbol{\omega}_t^{(raw)} - \mathbf{b}_g) \frac{V_{ref}}{(2^n - 1) s_g}, \end{aligned} \tag{5}$$

where $\mathbf{b}_a$ and $\mathbf{b}_g$ are the accelerometer and gyroscope biases, respectively, s_a and s_g are the accelerometer and gyroscope sensitivities, respectively, and n is the number of bits of the ADC. In our problem, the ADC was 10-bit ($n = 10$), and the reference voltage for the ADC was $V_{ref} = 3.3$ V $= 3300$ mV, corresponding to $s_a = 330$ mV/g, where g is gravity units, and $s_g = 3.33 \times 180/\pi$ mV/rad/s.

With all of these values and our aforementioned expected $\mathbf{a}_t$ and $\boldsymbol{\omega}_t$ in the early seconds of each dataset, we can now solve for $\mathbf{b}_a$ and $\mathbf{b}_g$, which we can then use to convert $\mathbf{a}_t^{(raw)}$ and $\boldsymbol{\omega}_t^{(raw)}$ to $\mathbf{a}_t$ and $\boldsymbol{\omega}_t$, respectively, for all t. While this makes little to difference in the overall problem, the biases used were averages of the biases over a large range of time stamps at the start of each dataset.

2) Projected Gradient Descent: One way to solve the constrained optimization problem in (4) is with projected gradient descent, which, at time stamp t, is given by

$$\mathbf{q}_t^{(i+1)} = \Pi_{\mathbb{H}_*} \left(\mathbf{q}_t^{(i)} - \alpha_t^{(i)} \nabla_{\mathbf{q}_t} c(\mathbf{q}_{1:T}) \big|_{\mathbf{q}_{1:T} = \mathbf{q}_{1:T}^{(i)}} \right), \tag{6}$$

where $\alpha_t^{(i)} > 0$ is the learning rate corresponding to the tth time stamp and the ith gradient descent step, and $\Pi_{\mathbb{H}_*}$ is the projection function that preserves $\mathbf{q}_t^{(i+1)} \in \mathbb{H}_*$ at each step i. It is also important to notice that projected gradient descent has to be done on each $\mathbf{q}_t$ in $\mathbf{q}_{1:T}$ in parallel, as all orientation values, $\mathbf{q}_{1:T}^{(i)}$, are factored into each step of the gradient descent. In implementation, the gradients $\nabla_{\mathbf{q}_t} c(\mathbf{q}_{1:T}) \big|_{\mathbf{q}_{1:T} = \mathbf{q}_{1:T}^{(i)}}$ were calculated using `torch.autograd.functional.jacobian` from the PyTorch library, with more implementation details discussed in Sections IV-C and IV-D.

Additionally, there are multiple projection operations $\Pi_{\mathbb{H}_*}$ that can ensure $\mathbf{q}_t \in \mathbb{H}_*$ at all time stamp t, but we chose the simple projection

$$\Pi_{\mathbb{H}_*}(\mathbf{q}) = \frac{\mathbf{q}}{||\mathbf{q}||_2}. \tag{7}$$

As further discussed in Section IV-D, we implemented this gradient descent using a single loop that increments i. All other operations are batched. For hyperparameters (further discussed in Section IV-C), we set $\alpha_1^{(i)} = \alpha_2^{(i)} = \cdots = \alpha_T^{(i)} = \alpha^{(i)}$, and chose $\alpha^{(i)} = 3\text{e-}3$. For initial conditions $\mathbf{q}_{1:T}^{(0)}$, we chose a motion model approximation, where we set $\mathbf{q}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\top$ and computed the rest of the trajectory using (1) as follows:

$$\begin{aligned} \mathbf{q}_1 = f(\mathbf{q}_0, \tau_0 \boldsymbol{\omega}_0), \quad \mathbf{q}_2 = f(\mathbf{q}_1, \tau_1 \boldsymbol{\omega}_1), \\ ..., \mathbf{q}_T = f(\mathbf{q}_{T-1}, \tau_{T-1} \boldsymbol{\omega}_{T-1}). \end{aligned}$$

As seen in Figure 1 and the training data roll, pitch, and yaw figures in Appendix D, the motion model approximation of $\mathbf{q}_{1:T}$ (orange) is already relatively close to the true trajectory most of the time, which makes it a great initialization point that helps us avoid much of the complex geometry of $c(\mathbf{q}_{1:T})$ (3) when doing gradient descent.

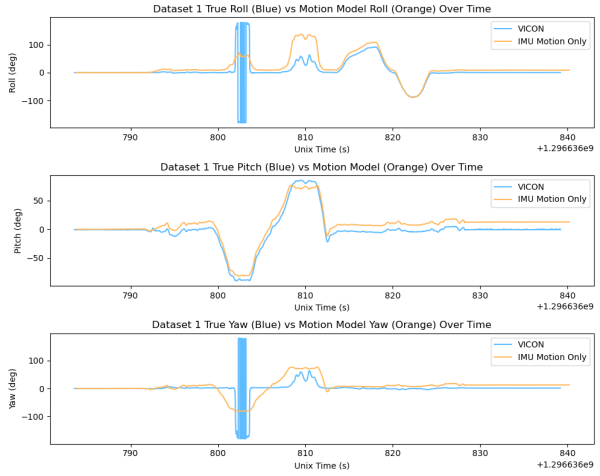

Fig. 1: Dataset 1 (Training) Roll, Pitch, and Yaw Comparison between Ground Truth (Blue) and Motion Model (Orange)

B. Panorama Reconstruction

To begin creating our panorama from the camera images, we must first find the orientation the camera was at when it took each image to be able to accurately place the images in the panorama. As $K < T$ in all of our datasets, and the camera and IMU data overlap in time ranges but often not in exact timestamp values, we must "sample" orientations in $\mathbf{q}_{1:T}$ at IMU timestamps t closest to the camera timestamps k. One simple method of doing this is by finding the "closest-in-the-past" timestamp t:

$$t^* = \arg\min_{t<k} |t - k| \tag{8}$$

This makes it so that the orientation described by $\mathbf{q}_{t^*}$ should correspond to the image taken by the camera at timestamp k. We can denote the new orientation trajectory corresponding to the images as $\mathbf{q}_{(0:T)^*}$.

One way to begin placing images in the panorama is to construct a "virtual environment" that we place our images along such that the image at the kth timestamp will align with the camera's field of view (FOV) when it has orientation $\mathbf{q}_{t^*}$. As we do not have any supplementary sensors, such as LiDARs, to measure distance, we can freely choose the shape and scale of this environment. For simplicity, we can choose a sphere of radius $\rho = 1$, which allows us to match image pixels to spherical coordinates (see Appendix A for the convention used) by varying ψ and λ based on the camera FOV and pixel location.

Our camera has a total horizontal FOV of 60° (or $\pi/3$ radians) and total vertical FOV of 45° (or $\pi/4$ radians), along with having its optical axis (z-axis) in the camera optical frame $\{O\}$ parallel to the the x-axis of $\{B\}$. The optical axis in $\{O\}$, which intersects the center of images, corresponds to the x-axis in the camera regular frame $\{R\}$ ($\hat{x}_r = \hat{z}_o$), along with $\hat{y}_r = -\hat{x}_o$ and $\hat{z}_r = -\hat{y}_o$.

In $\{R\}$, following the spherical coordinates convention described in Appendix A, the x-axis, which now corresponds to the center of the image, will have coordinates $\mathbf{s}_R = (\rho_R, \psi_R, \lambda_R) = (1, 0 \text{ rad}, \frac{\pi}{2} \text{ rad})$. Based on the size of the image and the camera FOV, the image pixels will then be evenly distributed, with fixed $\rho_R = 1$, across 320 evenly-spaced ψ in the range $-\frac{\pi}{6} \leq \psi_R \leq \frac{\pi}{6}$ and 240 evenly-spaced λ in the range $-\frac{\pi}{8} \leq \lambda_R \leq \frac{\pi}{8}$, resulting in an image-pixel to spherical coordinates mapping. It should be noted that the aforementioned coordinates in $\{R\}$ are constant with k ($\mathbf{s}_{R,0} = \mathbf{s}_{R,1} = \cdots = \mathbf{s}_{R,K}$), but this changes when we begin transforming them to $\{W\}$. As a result, we will start indexing the coordinates with k from now on.

To then find the corresponding coordinates of a given image pixel in $\{W\}$ at time stamp k, we must first convert it from spherical ($\mathbf{s}_{R,k}$) to cartesian coordinates, giving us $\mathbf{c}_{R,k} = (x_{R,k}, y_{R,k}, z_{R,k})$ and allowing us to perform linear-algebra-based rotations and translations. We start by transforming from $\{R\}$ to the IMU frame $\{B\}$. Since our camera is 10 cm above but aligned with the IMU, ${}_{\{B\}}R_{\{R\}} = I$ and ${}_{\{B\}}\mathbf{p}_{\{R\}} = \begin{bmatrix} 0 & 0 & 0.1 \end{bmatrix}^\top$, which results in the coordinates of the image pixel in $\{B\}$ at time stamp k, $\mathbf{c}_{B,k}$, being given by

$$\mathbf{c}_{B,k} = I\mathbf{c}_{R,k} + \begin{bmatrix} 0 & 0 & 0.1 \end{bmatrix}^\top.$$

It should also be noted that this transformation has a negligible effect on the final panorama results due to such a small transformation, but it still important to note conceptually. Additionally, $\{B\}$ and $\{R\}$ have changing orientation with time stamp k, while $\{W\}$ has fixed orientation.

Similarly, the transformation from $\{B\}$ to $\{W\}$ is

$$\mathbf{c}_{W,k} = R(\mathbf{q}_{t^*})\mathbf{c}_{R,k},$$

with ${}_{\{W\}}R_{\{B\}} = R(\mathbf{q}_{t^*})$, where $R(\mathbf{q}_{t^*})$ the rotation matrix (11) corresponding to $\mathbf{q}_{t^*}$, and no affine term (i.e., ${}_{\{B\}}\mathbf{p}_{\{R\}} = \mathbf{0}$, as mentioned at the beginning of this section).

We should have cartesian coordinates $\mathbf{c}_{W,k}$ in $\{W\}$ corresponding to each pixel for the kth camera image, and we can place it on our panorama image by first converting back to spherical coordinates $\mathbf{s}_{W,k}$. All pixels should now have $\{W\}$ coordinates along the sphere defined earlier. We can inscribe this sphere in a cylinder such that a point $\mathbf{s}_{W,k} = (1, \psi_{W,k}, \lambda_{W,k})$ has height $\lambda_{W,k}$ on the cylinder and longitude $\psi_{W,k}$ along the cylinder circumference. Then, we can get a rectangular image with width 2π radians and height π radians by unwrapping this cylinder. In practice, we go from the sphere directly to the rectangular image, which is a discrete array of H columns and V rows. Indices (h, v) on this rectangular image are related to spherical coordinates $(1, \psi_{W,k}, \lambda_{W,k})$, where $-\pi \leq \psi_{W,k} < \pi$ and $0 \leq \lambda_{W,k} \leq \pi$, by the following:

$$h = \left\lfloor \frac{H}{2\pi}(\psi_{W,k} + \pi) \right\rfloor, \quad v = \left\lfloor \frac{V}{2\pi}\lambda_{W,k} \right\rfloor, \tag{9}$$

where $h \in \{0, ..., H\}$, $v \in \{0, ..., V\}$, and $\lfloor \cdot \rfloor$ is a flooring operation. Each pair of indices (h, v) corresponds to an RGB vector representing the color of the corresponding pixel on the rectangular image. By default all RGB vectors on the image are initialized to $(0, 0, 0)$, and, when we map the $\mathbf{s}_{W,k}$ coordinates of a camera image pixel to a pixel on the rectangular image, we override the current RGB value with the camera image pixel's RGB value.

Finally, by repeating the entire process described in this section for all camera images k, the $H \times V$ rectangular image will become our panorama image. In our specific implementation, we chose $H = 2000$ and $V = 1000$.

IV. Results and Discussion

In our problem, there are 11 total datasets. The first 9 are training sets, while the last 2 (datasets 10 and 11) are test sets. There is only ground truth (VICON) orientation data for the 9 training sets, and there is only camera image data for training datasets 1, 2, 8, 9, and the two test sets.

A. Euler Angle (Roll, Pitch, and Yaw) Comparisons

The plots for our orientation estimates over all time stamps t in Euler angle (roll, pitch, yaw) convention can be found in Appendices D (Training) and E (Test). The training plots also have the corresponding ground truth (VICON) orientation data plotted for comparison. It should also be noted that any sections with vertical slopes just depict the angle's transition from -180° to 180° (or $-\pi$ radians to π radians).

We can see in all the Euler Angle training plots that our estimate (red) matches the ground truth (blue) almost perfectly for roll and pitch. The yaw seems to be less accurate, and there are noticeable deviations between the estimated and ground truth yaw. However, the inaccuracy of the yaw is expected, as our observation model only factors in linear acceleration from gravity, which applies an acceleration parallel to the z-axis. As a result, changes in the robot's yaw (rotation around the z-axis) will have no effect on the linear acceleration measurement, meaning the linear acceleration data will not capture any yaw information for us to factor into our orientation estimation.

Additionally, we can see that the motion-model-only estimate (see Section III-A2 for more on how it is calculated) provides an already good approximation of the ground-truth orientations most of the time, with the optimized trajectory often not being much of an improvement from it, which makes the motion-model-only estimate a very good initialization condition to optimize from.

B. Panoramas

The panorama images can be found in Appendices F (Training) and G (Test). For the training panoramas, we can see that, along with providing relatively coherent and consistent pictures of the environment, they match their corresponding ground-truth panoramas (made using the same method described in Section III-B but with the VICON ground truth data) very well. The black regions correspond to areas not covered by images, and this agrees with our plots in Appendices D (Training) and E (Test). For example, dataset 8 (Figures 9 and 15) had very little movement in roll and pitch, and this agrees with the corresponding panorama having no imagery outside of the middle strip of the image.

It is apparent in the panorama images that there is always an image "stacking" effect in the center of the panorama, which corresponds to $\psi = 0^\circ = 0$ rad and $\lambda = 90^\circ = \frac{\pi}{2}$ rad. This also agrees with our roll ϕ, pitch θ, and yaw ψ plots, as it can be seen in the plots in Appendices D and E that all Euler angles stay relatively close to 0°, with some slight variations due to noise, most of the time. As a result, most of the images taken by the camera would have been taken with the optical axis intersecting the spherical coordinates $(1, 0^\circ, 90^\circ)$, with a relatively upright image orientation due to having approximately 0° roll in $\{W\}$.

C. Hyperparameter Tuning

The main tunable hyperparameters in this problem are the gradient descent learning rates. There are many possible ways to tune the learning rates $\alpha_t^{(i)}$, including using second-order methods like Newton's method for adaptive learning rates and, since we are performing gradient descent on multiple parameters $\mathbf{q}_{1:T}$ at once, we could even use different learning rates for different $\mathbf{q}_t$. However, in the interest of time, we only tested the case where the learning rate is both constant across all $\mathbf{q}_{1:T}$ and gradient descent steps i, resulting in $\alpha_1^{(i)} = \alpha_2^{(i)} = \cdots = \alpha_T^{(i)} = \alpha^{(i)}$. After many tests, we made the following observations:

- 1e-3 $\leq \alpha^{(i)} <$ 1e-1 converge within 1000 iterations (before $i = 1000$)
- $\alpha^{(i)} <$ 1e-3 do not converge within 1000 iterations
- $\alpha^{(i)} >$ 1e-1 do not converge (loss stops decreasing and begins increasing)

In our code, we chose to use $\alpha^{(i)} =$ 3e-3.

D. Implementation Considerations

When implementing this problem, there are many subtle details that can significantly affect whether we get valid values. There are many quaternion operations used, which results in the presence of many functions such as $\arccos$, which are only defined for specific ranges of values. $\arccos$, specifically, is only defined for inputs between -1 and 1, returning invalid values (NaN) otherwise. Due to the volatile nature of values during differentiation, it is best to clamp the inputs to $\arccos$ to be between -1 and 1, which, for our specific implementation, helped prevent invalid values. Additionally, the gradient values tend to get very small (in the range of 1e-9 and 1e-10), and to get higher precision for such small numbers, we opted to enforce values to be of type `float64`. In fact, during testing, the PyTorch default type `float32` was often the cause for loss not decreasing.

There can also be many performance and runtime issues, mainly when calculating the gradients and searching for nearest indices as described in (8). Each dataset has thousands of IMU time stamps t, which would make individually looping through each t very slow for both gradient calculation and

index searching. Fortunately, PyTorch is highly optimized for batching operations, and we took advantage of this to rewrite many quaternion functions and our cost function to work for batches of data, allowing for operations like gradient descent to be performed almost entirely with matrix/tensor operations, resulting in must faster runtime.

In the specific case of timestamp matching, we opted to base our implementation on the `numpy.searchsorted` function, which performs binary search to find indices i that, in our case, satisfy $t_{i-1} < k \leq t_i$, with $i-1$ being our desired "closest-in-the-past" timestamp. This method implemented (8) much better performance-wise than our initial batched implementation which created a $K \times T$ array of the differences between each t and k and then used axis-based `argmin` across the T dimension.

It should also be noted that there are some time stamps in the camera datasets that are outside of the range covered by the IMU data, which means we cannot determine the orientation at which these images were taken. This, however, is not an issue, as it is clear in each dataset that the body has stopped rotating before this point.

ACKNOWLEDGMENT

I want to acknowledge that I got some help with Python programming from ChatGPT, consisting of syntax help and ideas on how to avoid NaNs during gradient descent. Additionally, I got some ideas from Piazza, such as the idea to clamp inputs to the arccos and add small epsilons to denominators to avoid division by 0 from Post @53. My explanation for the optimal yaw performing worse compared to the roll and pitch also takes ideas from the discussion in post @53.

APPENDIX

A. Spherical Coordinates Convention

In this report, spherical coordinates will be described as (ρ, ψ, λ), where $\rho \geq 0$ is the radial distance, $-\pi \leq \psi < \pi$ ($-180° \leq \psi < 180°$) is the azimuthal angle, and $0 \leq \lambda \leq \pi$ ($0° \leq \lambda \leq 180°$) is the polar angle. In regular frames, ψ measured relative to the positive x-axis on the xy-plane, and λ is measured relative to the positive z-axis on the plane containing the given point and the z-axis. The conversions between spherical and cartesian coordinates (x, y, z) are as follows:

$$\begin{aligned} x &= \rho\cos(\psi)\sin(\lambda) & \rho &= \sqrt{x^2+y^2+z^2} \\ y &= \rho\sin(\psi)\sin(\lambda) & \psi &= \arctan\left(\frac{y}{x}\right) \\ z &= \rho\cos(\lambda) & \lambda &= \arccos\left(\frac{z}{\sqrt{x^2+y^2+z^2}}\right) \end{aligned}$$

B. Notable Quaternion Operations

Quaternions, $\mathbf{q}$, can be depicted as either $\mathbf{q} = q_s + q_1 i + q_2 j + q_3 k \in \mathbb{H}$ or as $[q_s, \mathbf{q}_v]$, where $\mathbf{q}_v \coloneqq \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^\top \in \mathbb{R}^3$.

The rotation matrix corresponding to $\mathbf{q}$ is given by

$$R(\mathbf{q}) = E(\mathbf{q})G(\mathbf{q})^\top, \tag{10}$$

where $E(\mathbf{q})$ and $G(\mathbf{q})$ are given by

$$G(\mathbf{q}) \coloneqq \begin{bmatrix} -\mathbf{q}_v & q_s I - \hat{\mathbf{q}}_v \end{bmatrix} \text{ and } E(\mathbf{q}) \coloneqq \begin{bmatrix} -\mathbf{q}_v & q_s I + \hat{\mathbf{q}}_v \end{bmatrix}.$$

If we have a vector $\mathbf{x} \in \mathbb{R}^3$, we can then rotate it as

$$[0, \mathbf{x}'] = \mathbf{q} \circ [0, \mathbf{x}] \circ \mathbf{q}^{-1} = [0, R(\mathbf{q})\mathbf{x}], \tag{11}$$

where $\circ$ is quaternion multiplication, given by

$$\mathbf{q} \circ \mathbf{p} \coloneqq \begin{bmatrix} q_s p_s - \mathbf{q}_v^\top \mathbf{p}_v, q_s \mathbf{p}_v + p_s \mathbf{q}_v + \mathbf{q}_v \times \mathbf{p}_v \end{bmatrix}, \tag{12}$$

$(\cdot)^{-1}$ is the quaternion inverse, given by

$$\mathbf{q}^{-1} \coloneqq \frac{\bar{\mathbf{q}}}{||\mathbf{q}||^2}, \tag{13}$$

and $\bar{\mathbf{q}}$ is quaternion conjugation, given by

$$\bar{\mathbf{q}} \coloneqq [q_s, -\mathbf{q}_v] \tag{14}$$

We can also convert axis-angle rotation vectors $\boldsymbol{\theta} \in \mathbb{R}^3$ into unit-norm quaternions $\mathbf{q} \in \mathbb{H}_*$ using

$$\mathbf{q} = \exp([0, \boldsymbol{\theta}/2]), \tag{15}$$

where $\exp(\cdot)$ is the quaternion exponential function determined by

$$\exp(\mathbf{q}) \coloneqq e^{q_s}\left[\cos||\mathbf{q}_v||, \frac{\mathbf{q}_v}{||\mathbf{q}_v||}\sin||\mathbf{q}_v||\right]. \tag{16}$$

We can then recover this rotation vector from $\mathbf{q} \in \mathbb{H}_*$ using

$$[0, \boldsymbol{\theta}] = 2\log(\mathbf{q}), \tag{17}$$

where $\log(\cdot)$ is the quaternion logarithm determined by

$$\log(\mathbf{q}) \coloneqq \left[\log||\mathbf{q}||, \frac{\mathbf{q}_v}{||\mathbf{q}_v||}\arccos\frac{q_s}{||\mathbf{q}||}\right]. \tag{18}$$

C. Frame-to-Frame Rotations with Quaternions

Lemma 1. *If we have quaternions $\mathbf{q}, \mathbf{p} \in \mathbb{H}$, where $\mathbb{H} = \mathbb{C} \oplus \mathbb{C}j$, then*

$$(\mathbf{q} \circ \mathbf{p})^{-1} = \mathbf{p}^{-1} \circ \mathbf{q}^{-1}. \tag{19}$$

Proof. Noting the rule that $(a\mathbf{u}) \times (b\mathbf{v}) = ab(\mathbf{u} \times \mathbf{v})$, where $a, b \in \mathbb{R}$ and $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ and that $\mathbf{u} \times \mathbf{v} = -(\mathbf{v} \times \mathbf{u})$,

$$\begin{aligned} (\mathbf{q} \circ \mathbf{p})^{-1} &= \frac{\overline{\mathbf{q} \circ \mathbf{p}}}{||\mathbf{q} \circ \mathbf{p}||} = \frac{\overline{\mathbf{q} \circ \mathbf{p}}}{||\mathbf{q}|| \cdot ||\mathbf{p}||} \\ &= \frac{1}{||\mathbf{q}|| \cdot ||\mathbf{p}||}\left[q_s p_s - \mathbf{q}_v^\top \mathbf{p}_v, -(q_s \mathbf{p}_v + p_s \mathbf{q}_v + \mathbf{q}_v \times \mathbf{p}_v)\right] \\ &= \frac{1}{||\mathbf{q}|| \cdot ||\mathbf{p}||}\big[q_s p_s - (-\mathbf{q}_v)^\top(-\mathbf{p}_v), \\ &\qquad q_s(-\mathbf{p}_v) + p_s(-\mathbf{q}_v) - (-\mathbf{q}_v) \times (-\mathbf{p}_v)\big] \\ &= \frac{1}{||\mathbf{q}|| \cdot ||\mathbf{p}||}\big[p_s q_s - (-\mathbf{p}_v)^\top(-\mathbf{q}_v), \\ &\qquad p_s(-\mathbf{q}_v) + q_s(-\mathbf{p}_v) + (-\mathbf{p}_v) \times (-\mathbf{v}_v)\big] \\ &= \frac{[p_s, -\mathbf{p}_v] \circ [q_s, -\mathbf{q}_v]}{||\mathbf{p}|| \cdot ||\mathbf{q}||} = \frac{\bar{\mathbf{p}}}{||\mathbf{p}||} \circ \frac{\bar{\mathbf{q}}}{||\mathbf{q}||} \\ &= \mathbf{p}^{-1} \circ \mathbf{q}^{-1}, \end{aligned}$$

where $(\cdot)^{-1}$ is the quaternion inverse defined in (13). □

If we define three different coordinate frames $\{A\}$, $\{B\}$, and $\{C\}$, all with the same origin, and two quaternions, ${}_A\mathbf{q}_B \in \mathbb{H}_*$, representing the orientation of frame $\{B\}$ in frame $\{A\}$, and ${}_B\mathbf{q}_C$, representing the orientation of frame $\{C\}$ in frame $\{B\}$, the quaternions will correspond to rotation matrices ${}_{\{A\}}R_{\{B\}} := R({}_A\mathbf{q}_B)$ and ${}_{\{B\}}R_{\{C\}} := R({}_B\mathbf{q}_C)$, as described in Appendix B. The rotation matrix corresponding to the orientation of $\{C\}$ in $\{A\}$ is then given by ${}_{\{A\}}R_{\{C\}}$.

If a point has cartesian coordinates $\mathbf{c}_C \in \mathbb{R}^3$ in frame $\{C\}$, we can find its coordinates in frame $\{A\}$ as $\mathbf{c}_A = {}_{\{A\}}R_{\{C\}}\mathbf{c}_C = {}_{\{A\}}R_{\{B\}\{B\}}R_{\{C\}}\mathbf{c}_C = R({}_A\mathbf{q}_B)R({}_B\mathbf{q}_C)\mathbf{c}_C$. We can then rewrite the following operation as a quaternion rotation by embedding $R({}_A\mathbf{q}_B)R({}_B\mathbf{q}_C)\mathbf{c}_C$ into a quaternion as $[0, R({}_A\mathbf{q}_B)R({}_B\mathbf{q}_C)\mathbf{c}_C]$, which we can then rewrite as the following, based on quaternion rotation defined in Appendix B and our result in Lemma 1:

$$
\begin{aligned}
&[0, R({}_A\mathbf{q}_B)R({}_B\mathbf{q}_C)\mathbf{c}_C] \\
&= {}_A\mathbf{q}_B \circ [0, R({}_B\mathbf{q}_C)\mathbf{c}_C] \circ {}_A\mathbf{q}_B^{-1} \\
&= {}_A\mathbf{q}_B \circ \left({}_B\mathbf{q}_C \circ [0, \mathbf{c}_C] \circ {}_B\mathbf{q}_C^{-1}\right) \circ {}_A\mathbf{q}_B^{-1} \\
&= \left({}_A\mathbf{q}_B \circ {}_B\mathbf{q}_C\right) \circ [0, \mathbf{c}_C] \circ \left({}_B\mathbf{q}_C^{-1} \circ {}_A\mathbf{q}_B^{-1}\right) \\
&= \left({}_A\mathbf{q}_B \circ {}_B\mathbf{q}_C\right) \circ [0, \mathbf{c}_C] \circ \left({}_A\mathbf{q}_B \circ {}_B\mathbf{q}_C\right)^{-1} \\
&= [0, R({}_A\mathbf{q}_B \circ {}_B\mathbf{q}_C)\mathbf{c}_C],
\end{aligned}
$$

which leads to

$$
R({}_A\mathbf{q}_B)R({}_B\mathbf{q}_C)\mathbf{c}_C = R({}_A\mathbf{q}_B \circ {}_B\mathbf{q}_C)\mathbf{c}_C. \tag{20}
$$

D. Roll, Pitch, and Yaw Training Comparison Plots

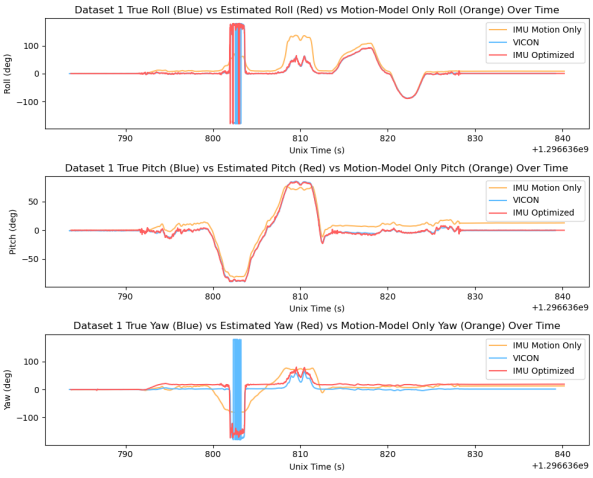

Fig. 2: Dataset 1 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

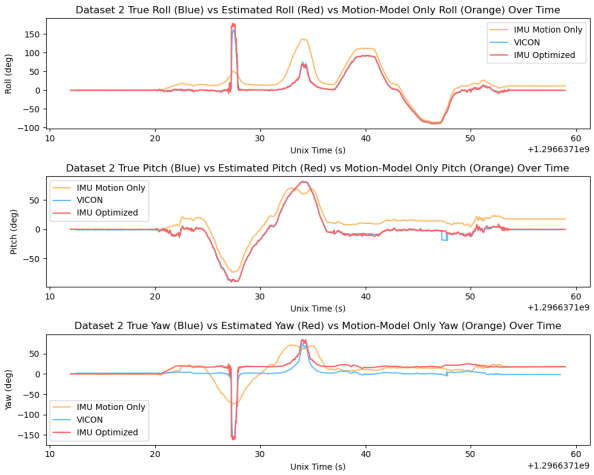

Fig. 3: Dataset 2 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

Fig. 4: Dataset 3 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

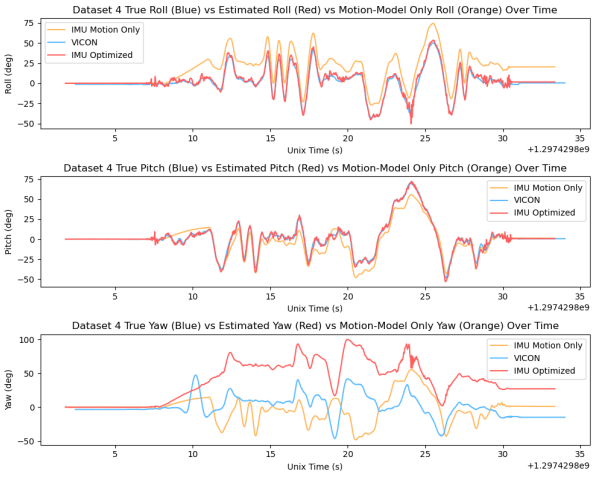

Fig. 5: Dataset 4 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

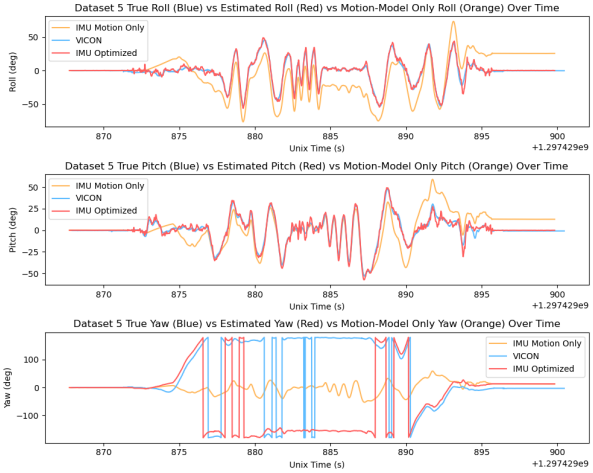

Fig. 6: Dataset 5 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

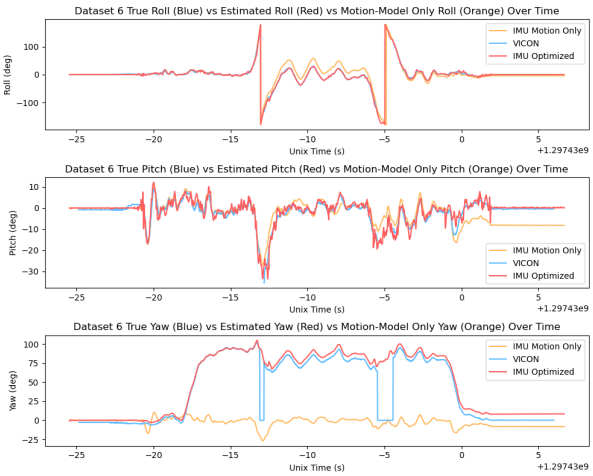

Fig. 7: Dataset 6 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

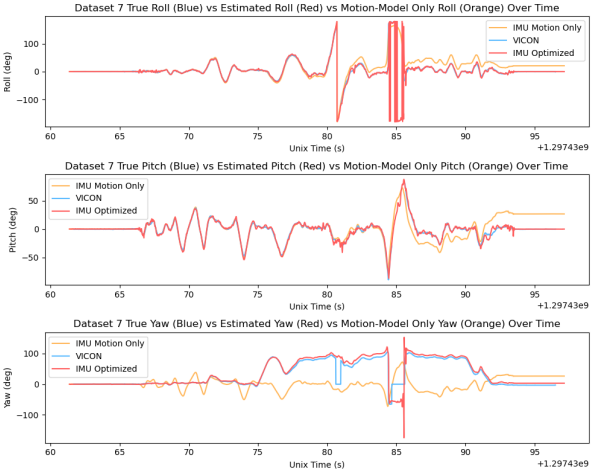

Fig. 8: Dataset 7 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

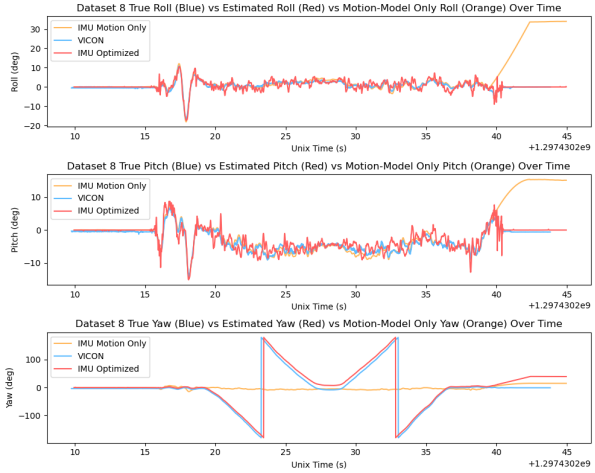

Fig. 9: Dataset 8 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

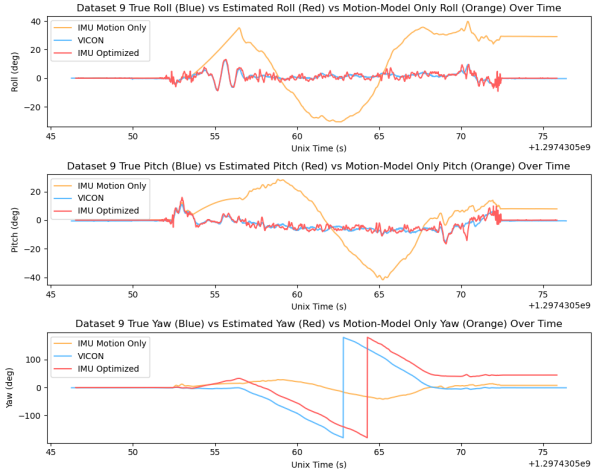

Fig. 10: Dataset 9 (Training) Ground Truth (Blue), Motion Model (Orange), and Estimated (Red) Roll, Pitch, and Yaw

E. Roll, Pitch, and Yaw Testing Plots

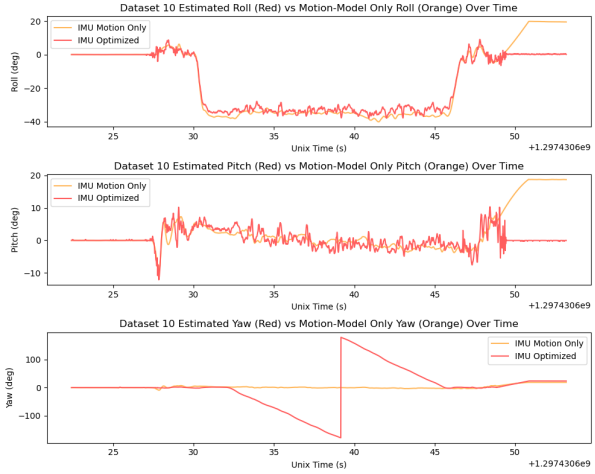

Fig. 11: Dataset 10 (Test) IMU Estimated (Red) vs Motion Model (Orange) Roll, Pitch, and Yaw

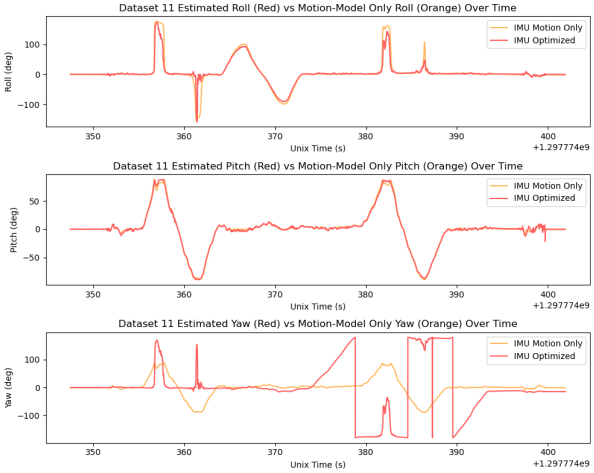

Fig. 12: Dataset 11 (Test) IMU Estimated (Red) vs Motion Model (Orange) Roll, Pitch, and Yaw

F. Training Panoramas

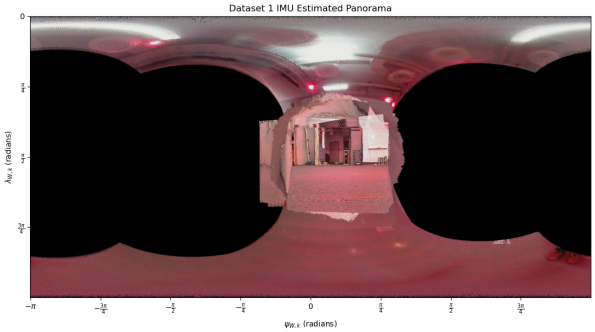

Fig. 13: Dataset 1 (Training) Estimated Orientation Panorama

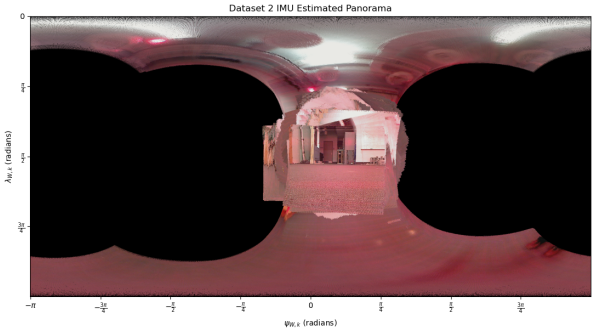

Fig. 14: Dataset 2 (Training) Estimated Orientation Panorama

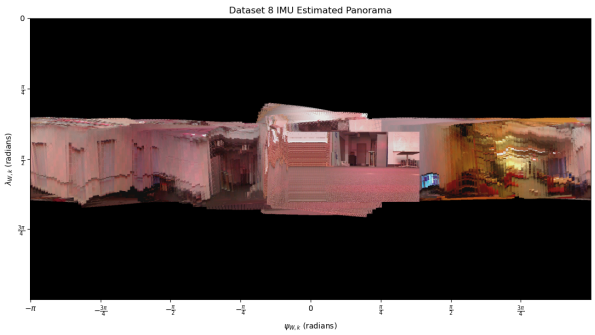

Fig. 15: Dataset 8 (Training) Estimated Orientation Panorama

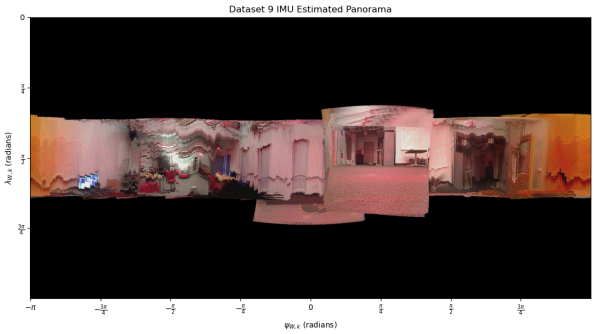

Fig. 16: Dataset 9 (Training) Estimated Orientation Panorama

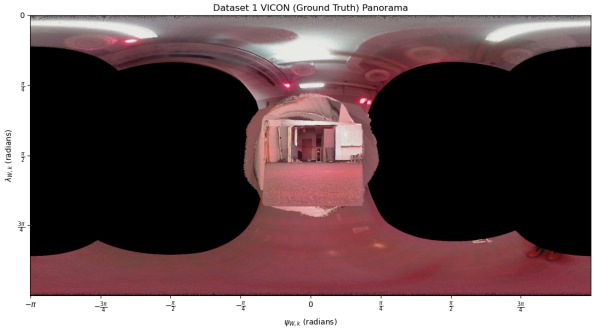

Fig. 17: Dataset 1 (Training) Ground Truth Orientation Panorama

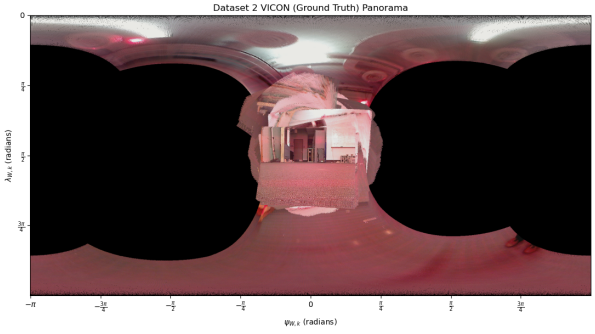

Fig. 18: Dataset 2 (Training) Ground Truth Orientation Panorama

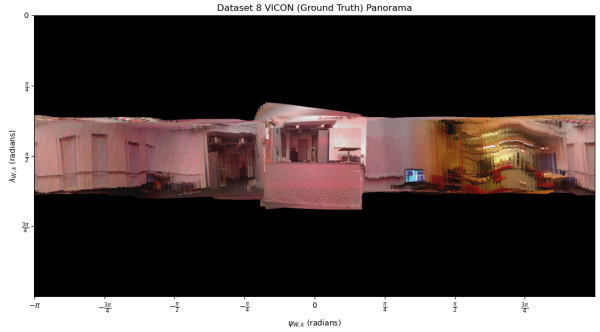

Fig. 19: Dataset 8 (Training) Ground Truth Orientation Panorama

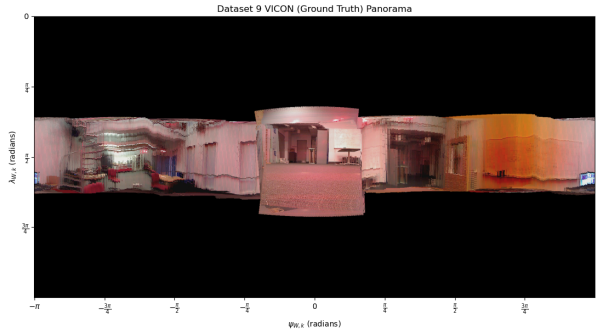

Fig. 20: Dataset 9 (Training) Ground Truth Orientation Panorama

G. Testing Panoramas

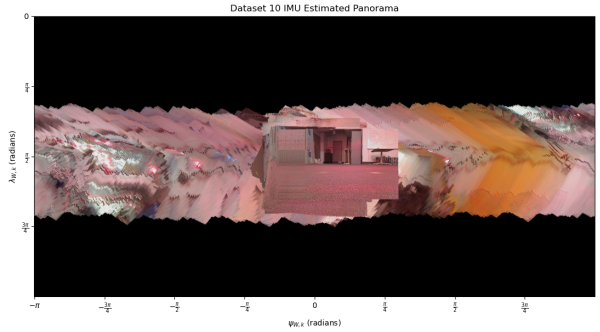

Fig. 21: Dataset 10 (Test) Estimated Orientation Panorama

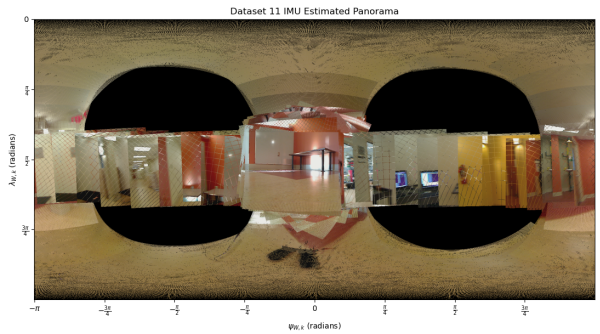

Fig. 22: Dataset 11 (Test) Estimated Orientation Panorama